1 1
OF / DE

CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

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

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

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

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

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

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

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

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

National Library
of CanadaBibliothèque nationale
du Canada

Canadian Theses Division

Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

60011

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

RAJAGOPALAN HARISHA

Date of Birth — Date de naissance

03 Feb, 57

Country of Birth — Lieu de naissance

INDIA

Permanent Address — Résidence fixe

c/o S. Rajagopalan, Station Master,
Ashokapuram, Mysore - 570008,
India.

Title of Thesis — Titre de la thèse

A Diagonalization Technique for the area
reduction of Programmable Logic Arrays.

University — Université

CARLETON UNIVERSITY

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

Master of Engineering

Year this degree conferred — Année d'obtention de ce grade

1983

Name of Supervisor — Nom du directeur de thèse

John P. Knight

Permission is hereby granted to the NATIONAL LIBRARY OF
CANADA to microfilm this thesis and to lend or sell copies of
the filmThe author reserves other publication rights, and neither the
thesis nor extensive extracts from it may be printed or other-
wise reproduced without the author's written permissionL'autorisation est, par la présente, accordée à la BIBLIOTHÈ-
QUE NATIONALE DU CANADA de microfilmer cette thèse et de
prêter ou de vendre des exemplaires du film.L'auteur se réserve les autres droits de publication, ni la thèse
ni de longs extraits de celle-ci ne doivent être imprimés ou
autrement reproduits sans l'autorisation écrite de l'auteur

Date

April 29, 83

Signature

H. Harisha

A DIAGONALIZATION TECHNIQUE FOR THE
AREA REDUCTION OF PROGRAMMABLE LOGIC ARRAYS

by

© Rajagopalan Harisha

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

Master of Engineering

Faculty of Engineering
Carleton University
Ottawa, Ontario

April 1983

The undersigned recommend to the Faculty of Graduate Studies and Research, acceptance of the thesis:

"A Diagonalization Technique for the Area Reduction of Programmable Logic Arrays"

submitted by Rajagopalan Harisha, in partial fulfillment of the requirements for the degree of Master of Engineering.

Thesis Supervisor
Chairman,
Department of Electronics

Department of Electronics

Carleton University

April 1983

ABSTRACT

A novel diagonalization technique to reduce the area requirements of a PLA is described in this thesis. A program to diagonalize a PLA has been developed. More than 40% reduction in the area was typical for most of the ten PLAs considered. Folding techniques can still be applied to the diagonalized PLA for further improvement. Also, it is simpler to perform folding on a diagonalized PLA. The diagonalization problem, like the folding problem, is NP-complete.

ACKNOWLEDGEMENTS

At the very outset, I am deeply indebted to Professor J.P. Knight, my thesis supervisor, for his encouragement, guidance, and moral support.

I place on record my deep sense of gratitude to Mr. Emil Girczyc for his innovative ideas, excellent suggestions, and his constant involvement in my thesis work. The enlightening discussions I had with Mr. K. R. Lakshmi Kumar are worthy of mention. My sincere thanks are due to him.

Professor B.A. Bowen's influence, financial support, and his immaculate and erudite sense of humor have cheered me all along.

Finally, the unblemished and swift typing of Miss Diane Dodds is acknowledged.

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT	i
ACKNOWLEDGEMENTS	ii
TABLE OF CONTENTS	iii
LIST OF FIGURES	iv
LIST OF TABLES	v
 CHAPTER 1 Introduction	 1
 CHAPTER 2 An Overview of Programmable Logic Arrays	 6
2.1 Basic Concepts	6
2.2 Design Methodology	9
 CHAPTER 3 Folding Problem	 15
3.0 Introduction	15
3.1 Folding Techniques	16
3.2 NP-Complete Problems	22
 CHAPTER 4 A Diagonalization Approach to the PLA	 27
4.0 Introduction	27
4.1 Bandwidth Minimization Problem	27
4.2 An Algorithm for Diagonalization	31
4.3 Results	40
 CHAPTER 5 Conclusions	 45
5.1 Summary	45
5.2 Future Work	46
 REFERENCES	 48
 APPENDIX	

LIST OF FIGURES

		Page
Figure 1.1	An Example PLA; the crosspoints represent an AND in the AND plane and an OR in the OR plane	3
Figure 2.1	Basic PLA building block. Each PLA uses two of these; one for input and one for output. Each block consists of cells. The above block has 20 cells.	7
Figure 2.2	Basic PLA	8
Figure 2.3	Clocked PLA state-machine	10
Figure 3.1	Example PLA (a) Without folding; (b) Vertical lines broken into two parts to give column folding	17
Figure 3.1	(c) Horizontal lines broken into two parts to give row folding; (d) A possible folding of the PLA; both rows and columns are foleded	18
Figure 3.2	Example PLA (a) Without folding; (b) Folding rows 1 and 2 makes it impossible to fold rows 3 and 4 (b) Folding rows 3 and 4 makes it impossible to fold rows 1 and 2	21
Figure 4.1	A band matrix with bandwidth equal to 2	29
Figure 4.2	(a) A symmetric matrix and its associated graph; (b) A matrix and its associated directed graph	30
Figure 4.3	A regular PLA matrix and its diagonalized version	32
Figure 4.4	(a) An example PLA matrix which does not cover significantly less area when diagonalized because of the density of row 5 (b) Matrix of (a), diagonalized with row 5 removed and placed beside the diagonal part	33
Figure 4.5	A hypothetical matrix and the steps necessary for reorientation	37
Figure 4.6a	A 40 x 42 PLA matrix	41
Figure 4.6b	Diagonalized reduction of the PLA in Figure 4.6a. The area is 65% of the original	42

LIST OF TABLES

		<u>Page</u>
Table 3.1	Comparison of polynomial and exponential time complexity functions [17]	24
Table 4.1	Weights associated with the elements of a 5 x 5 matrix	35
Table 4.2	Results obtained from the algorithm	43

Chapter 1

INTRODUCTION

With advances in semiconductor technologies, VLSI design has increasingly become complex. It has been widely agreed that the computer is an essential tool for designing VLSI. However, even with the aid of the computer, the complexity of VLSI has grown to the extent that there are doubts about the designers' ability to keep up with the process technology. A more sophisticated set of design aids is necessary to manage the complexity.

The traditional way of implementing a circuit is to use random logic. Random logic provides the highest density and performance. With this technique, the physical layout problems are ignored in the earlier stages of the design. With SSI/MSI complexities, the physical layout problems are not severe. However, with LSI/VLSI circuits, these problems become unmanageable resulting in higher development costs and slow design turnaround.

One possible way to handle the complexity is to use structured design techniques. The popular techniques are gate-arrays, ROMs, and programmable logic arrays (PLAs).

In gate-array based designs, the chip real estate is filled with gate topologies (may include PLAs) in a standard pattern. Standard channels are provided for enabling the interconnection of these gates.

The system designer provides the interconnections of these gates to form the system. Since only interconnections are to be designed, this technique provides a shorter design time. But the chip density suffers since all the gates on the chip cannot be utilized.

In a ROM, the relationship between the input and output codes is relatively fixed, usually alterable only by relatively slow techniques. The major problem with this approach is that many combinational logic functions require only a small fraction of all product minterms for a canonical sum of products implementation. In such cases, a ROM is wasteful of area.

A PLA is a regular structure which can implement Boolean sums of products. PLAs can also be used to implement synchronous finite-state machines by using the PLA form of combinational logic and feeding back some of the outputs to the inputs [1]. An example of a PLA is shown in Figure 1.1. As a fully general logic element, a PLA can replace a block of random logic. The complexity of implementing combinatorial logic functions is simplified to a large extent since the design is reduced to two levels (AND, OR) as opposed to multi-level implementations in random logic. Also, the hazards in two-level logic (PLAs) can be easily found by algorithmic methods. Compared with random logic, PLAs offer the following advantages: fast design turnaround, low design cost, automation of various design tasks, and relative ease in testing and verification techniques. The PLA approach lends itself more readily to engineering changes.

$$F1 = \bar{A} \bar{B} C + \bar{A} B \bar{C}$$

$$F2 = \bar{B} \bar{C} + A B + \bar{A} \bar{B} C$$

Figure 1.1: An Example PLA; the Crosspoints Represent an AND in the AND Plane and an OR in the OR Plane

The PLA is a structure that has all the generalities of a memory for implementing combinational logic functions. However, the key issue is that any specific PLA structure needs a row of circuit elements only for those product terms that are actually required to implement a given logic function. Since it does not contain entries for all possible minterms, it is usually far more compact than a ROM implementation of the same function. The basic concepts of the PLA and the design methodology involved in PLA based systems are discussed in Chapter 2.

Compared with random logic, the silicon area required for PLA implementation is significantly larger. A major portion of the total area is occupied by interconnects and unused devices. This will introduce unnecessary parasitics, thereby degrading the time performance of the PLA. Therefore, techniques to reduce the area are very important. Otherwise, the advantages of the PLAs may be offset by poor silicon utilization. One possible way to reduce the area is to simplify the Boolean algebraic expressions. However, even after this simplification, the area required is still very large since most PLA matrices are very sparse.

This thesis is mainly concerned with the techniques for reducing the area of sparse PLA matrices. Folding techniques, which have recently become popular, attempt to reduce the area. Folding problem is known to be NP-complete (nondeterministic polynomial-time complete). The folding techniques, and the issues related to NP-complete problems are the topics in Chapter 3.

A different approach to the PLA area reduction problem is investigated in this thesis. This technique, which could be termed as the "diagonalization technique", attempts to arrange the elements of a sparse matrix as close to the diagonal as possible. The diagonalization problem is also NP-complete. However, by using efficient heuristics, the algorithm squeezes the sparse PLA matrix thereby reducing the area. The results obtained with this technique are comparable to the improvements achieved by folding techniques. A 45% reduction in the area was typical for most of the PLAs considered. However, this result is not guaranteed for dense PLAs. A detailed discussion of the diagonalization technique may be found in Chapter 4.

Chapter 5 concludes this thesis by summarizing the work that has been done and suggests further research possible in this area.

Chapter 2

AN OVERVIEW OF PROGRAMMABLE LOGIC ARRAYS2.1 Basic Concepts

The PLA is a regular, read-only structure which can implement irregular combinational functions. PLAs implement Boolean sums of products; any logic equation can be represented as a sum of products.

Figure 2.1 shows the basic PLA building block. In most applications two blocks are connected in series, with the outputs of the first connected to the inputs of the second. The width of a PLA is usually two units for each input signal (a true column and a complemented column) and one unit for each output. The building block shown has 5 columns and 4 rows. A building block consists of cells. All cells are of the same size - usually square. The cells with the X's usually contain an extra transistor. The function F might be AND, OR, NOR, etc. Generally, the first array implements the function AND, and the second array the function OR. However, it is possible to achieve the same result using both arrays to implement the function NOR. A PLA can also be visualized as a table look-up structure, where the AND array forms the look-up library and the OR array forms the resultant output for the operation.

Figure 2.2 shows the basic PLA. In the AND plane, inputs and their complements run vertically while each horizontal line carries the signal corresponding to the product terms of the inputs selected via

Figure 2.1: Basic PLA Building Block. Each PLA Uses Two of These; One for Input and One for Output. Each Block Consists of Cells. The Above Block Has 20 Cells.

Figure 2.2 Basic PLA

personalization. The product term lines are then selectively summed in the OR plane in exactly the same way to form vertical output lines. Matrix personalization is analogous to the programming of a ROM. Thus, the PLA provides a direct implementation of logic functions in the two level Boolean sum-of-products form. In MOS technology, the PLA is typically implemented as a NOR-NOR array [1]. However, it is important to note that the PLA structure is not bound to any particular gate type.

PLAs can be implemented with or without clocks. Figure 2.3 shows a schematic PLA which is clocked and has state registers to implement a synchronous finite-state machine. The function of this circuit is determined by the personalization matrix of its PLA logic. However, it is important to note that an implementation of a synchronous finite-state machine functions correctly only if the delays in the circuit paths are sufficiently short compared to the clock period [1].

2.2 Design Methodology

PLA design is similar to other IC designs. The task of implementing combinational logic as a PLA can be partitioned into five major subtasks [3]. They are:

1. System design
2. Logic design
3. Circuit design
4. Layout
5. Testing

Figure 2.3: Clocked PLA State-Machine

1. System Design

The system design phase involves structuring, partitioning and state assignment [3]. Before any logical function is implemented as a PLA, the silicon system containing the PLA must be specified. The chip floor plan determines which signals and functions are localized. Data flow considerations are important in the preliminary design because it is often not a good idea to run signals long distances when they can be combined where they are generated. This approach might lead to smaller PLAs. In any case, the chip design has strong implications on the individual PLA functional requirements.

A PLA often takes an inappropriately large amount of area; the area of a PLA is the product of I/O count and the minterms. The fact that a PLA implements actually several independent sets of equations can be used to achieve a significant savings in the area. The designer can substitute several smaller PLAs for one large PLA. In the best case, the designer can reduce the amount of area required to implement the PLA by a factor of K by implementing K independent sets of equations as K small PLAs instead of one large PLA [4]. These partitioning issues are decided during system design.

The PLA system design phase produces the functional specifications for all the PLAs used on a chip. These specifications might be represented as logic equations, or state machine descriptions. Alternatively, the specifications might be represented as a set of AND/OR planes. Programs have been developed that translate arbitrarily formed Boolean input-output expressions into AND-OR form [5,6]. In any case, partitioning

the chip's random logic requirements into several PLAs forms the major issue of the system design.

2. Logic Design

In this design phase, each PLA is considered individually. Each PLA's specification requires it to perform certain logical functions. Logic design phase attempts to minimize the cost of providing these requirements. The key aspect to be considered while attempting to minimize the cost is the chip area. In the classical approach to Boolean logic minimization, each equation is considered individually; this means that each PLA output column is considered individually. However, in PLAs, a global minimization is required. All equations in a single block must be minimized together and the number of literals in a product does not significantly affect the PLA cost.

In a PLA, each product term requires one row whether it has one literal, or many literals. Also, if terms are used by other equations, then the minimization techniques are different. It is easy to consider these aspects if the number of input variables is small. However, as the number of input variables increases, it becomes necessary to use a series of heuristic algorithms. There is no guarantee that an absolute minimum will be achieved, as is generally the case when heuristics are employed.

In many PLA designs, an initial inspection is necessary before Boolean logic minimization is attempted; this is to detect null or

redundant inputs, outputs, and products. Several routines have been developed to check and eliminate various inefficiencies from PLAs [3, 6].

3. Circuit Design

The electrical properties of the PLA are optimized during circuit design phase. Parameters such as speed, power, and area are usually considered. A proper design of the power/ground bus is necessary to reduce the noise. The PLA circuit design phase can be classified into two levels. In the first level, the speed, power, and area requirements of the basic PLA cell are considered. In the second level, the array-level design required to interface each particular PLA into its individual chip environment is done. The issues considered include power, ground, clocks, and drivers.

4. Layout

During this phase, the PLA is connected to its external signals from the chip. The flexibility of the PLA structure can be exploited at this time to optimize the layout. PLA layout design is an important area for automatic PLA optimization. Automatic layout optimization can potentially save more area and power than logic and circuit design combined [3]. In Chapters 3 and 4, the techniques related to the optimization of the PLA layout are discussed.

5. Testing

Due to the regular structure of PLAs, it is possible to model and analyze the physical faults and the stuck-at faults (transistors failing to switch state). Physical faults include shorts between the lines, and crosspoint defects (spurious absence or presence). In order to generate tests efficiently for PLAs, the generation process should utilize the regularity, structure, and efficiency of the bit array. The faults to be detected, however, are hardware-dependent. A possible approach is to model the fault being considered as a change in the corresponding bit pattern. Using this approach, it is possible to show that several classes of faults are tested by properly testing all single crosspoint faults of the array [7].

Currently of interest in PLA development, as in general VLSI design, is the incorporation of on-chip testing circuitry to speed the testing phase of production. One such method [8], uses an on-chip test pattern generator and a series of shift registers and enable lines to individually test crosspoint connections for stuck-at faults and short circuits in the array.

Chapter 3

FOLDING PROBLEM3.0 Introduction

The design of LSI/VLSI systems is so complex, that a structured design methodology must be used to ensure the correctness of the design and the feasibility of design changes needed to improve the efficiency of the system. The PLA, because of its regular structure, has proved to be a very effective tool for designing combinatorial functions. Unfortunately, implementing a circuit with a PLA can give poor silicon utilization. For any personalization of the PLA, a large percentage of the total area is occupied by unused devices and is thus wasted. In other words, most PLA matrices are very sparse. Therefore, techniques to reduce the PLA's area are of great importance, and the usefulness of the PLA in LSI/VLSI applications often depends on how successful these techniques are.

Most area reduction efforts, until recently, have been directed towards the simplification of the Boolean expressions. These include optimization of the number of product terms via classical algebraic minimization [9,10], and logic partitioning of the inputs [11]. But, even after logic minimization the resultant PLAs often remain much too sparse.

To further reduce the area, folding techniques [12] may be used. Unfortunately, the optimal folding for a general PLA cannot be found in a reasonable time because the amount of work to be done to find such a folding increases too rapidly with the size of the PLA matrix. Section 3.1 reviews the folding concepts while section 3.2 discusses the significance of NP-complete problems.

3.1 Folding Techniques

PLA folding can be best explained by an example. Consider the PLA matrix shown in Figure 3.1a. The horizontal lines are assigned to the product term T_K (for example, $T_1 = X_1 \cdot X_3$). The vertical lines are assigned to the input variables X_I and output variables Y_J . A crosspoint X_I, T_K is marked if the term T_K depends on X_I , and a crosspoint Y_J, T_K is marked if T_K is a term of Y_J (for example, $Y_2 = T_2 + T_3$). Now, the folding may be done as follows [14]:

- 1) Vertical lines may be broken into an upper and a lower part. Under some conditions two input lines (or two output lines) do not need two complete lines but only the two parts of one line. This is illustrated in Figure 3.1b.
- 2) Horizontal lines may be broken into a left and a right part. Under some conditions two terms do not need two lines but only the two parts of one line. This is illustrated in Figure 3.1c.

Figure 3.1d shows a possible folding of the PLA matrix shown in Figure 3.1a.

(a)

(b)

Figure 3.1: Example PLA

(a) Without Folding; (b) Vertical Lines Broken

into Two Parts to Give Column Folding

(c)

(d)

Figure 3.1: (c) Horizontal Lines Broken Into Two Parts to Give Row Folding;
 (d) A Possible Folding of the PLA; Both Rows and Columns
 Are Folded.

In all the folding strategies the main aim is to reduce the matrix area. There are many kinds of folding possible [13, 15]. This thesis will attempt to aid understanding by introducing the following simple classification:

1. Column folding
2. Row folding
3. Mixed column and row folding
4. Constrained folding

All the above techniques are described below.

1. Column Folding

Column folding consists of dividing an AND plane column into two parts (input folding) so that two inputs can share the same physical column and/or of dividing an OR plane column into two parts (output folding) so that two outputs can share the same physical column. Figure 3.1a shows an example PLA and Figure 3.1b showed a possible column folding. As illustrated in Figure 3.1b, the electrical connections of the two signals are placed on opposite sides of a physical "cut" to be located somewhere in the column.

2. Row Folding

Row folding consists of making a cut between any two columns so that a single physical row in each AND or OR array can be shared by two different product terms, one feeding (AND) or being fed (OR) from the array on the right and the other feeding or being fed from the array on the left.

Folding two rows introduces constraints on the folding of other rows [15]. As an example, consider the PLA shown in Figure 3.2a. Folding rows 1 and 2 makes it impossible to fold rows 3 and 4. This is illustrated in Figure 3.2b. Similarly, folding rows 3 and 4 puts a constraint on folding rows 1 and 2. This is illustrated in Figure 3.2c. Therefore, it is quite clear that these two conditions cannot be satisfied simultaneously. Thus, a necessary condition for row folding to be implementable is that the relative positions of the rows to be folded imply no "cyclic" constraints on the positions of the columns of the PLA.

3. Mixed Column and Row Folding

Both column and row folding can be applied to the same PLA. However, there are interactions between row and column folding. For example, if column folding is performed first, it will introduce additional constraints on a subsequent row folding. A good folding strategy consists of performing mixed row and column folding. Here one performs a certain number of column folds followed by a certain number of row folds, followed by more column folding and so on until no more folds are possible. It has been shown that the area savings one can achieve depend critically on the sequence of folding applied [15].

4. Constrained Folding

A problem shared by the row folding strategies is that inputs and outputs must be reordered to make the fold possible. Reordering of the inputs and outputs may have a considerable impact on the routing of

(a)

(b)

(c)

Figure 3.2: Example PLA

(a) Without Folding

(b) Folding Rows 1 and 2 Makes It Impossible to Fold Rows 3 and 4

(c) Folding Rows 3 and 4 Makes It Impossible to Fold Rows 1 and 2

D

the signals from other parts of the chip. The designer may want to put constraints on the location of inputs and outputs so as not to lose, while routing the signals, the gain in silicon area achieved by folding. In this case, we have a constrained folding.

3.2 NP-Complete Problems

In section 3.1 the basic concepts of folding were presented. The folding problem has been proven to be NP-complete [13]. The bandwidth minimization problem, discussed in Chapter 4, is also proven to be NP-complete [29]. Therefore, before attempting to find a solution it is necessary to understand the important properties of NP-complete problems. However, before discussing NP-complete problems, it is necessary to understand several related concepts which are discussed below [17,18].

In general, one is interested in finding the "most efficient" algorithm for solving a problem. A most efficient algorithm normally means the fastest. Different algorithms possess a wide variety of different time complexity functions. However, basically they can be classified into two broad groups [17]: polynomial time algorithms and exponential time algorithms.

A polynomial time algorithm is defined to be one whose time complexity function is of the order of some polynomial function $p(n)$, where n is the input length. Any algorithm whose time complexity function cannot be so bounded is called an exponential time algorithm. The distinction between these two types of algorithms has particular

significance when considering the solution of large size problems. Table 3.1 [17] illustrates the differences in growth rates among several typical complexity functions of each type. This suggests that exponential time algorithms, in general, are not good algorithms.

It is widely agreed that a problem has not been well solved until a polynomial time algorithm is known for it. Problems for which no polynomial time algorithm is possible are referred to as "intractable". Some are "provably" intractable, others are only "apparently" intractable.

Most of the apparently intractable problems encountered in practice can be solved in polynomial time with the aid of a nondeterministic algorithm which is composed of two stages - a guessing stage and a checking stage. Nondeterministic algorithms are a mathematical fiction since no real computer can execute the guessing stage. A nondeterministic machine may be visualized as a machine that has as many processors as one needs. If one reaches a branch in a program, say a tree search, then both branches can be followed on parallel processors. Problems that can be done in polynomial time on these parallel processors belong to the class NP (nondeterministic polynomial-time) [31].

Certain problems in the class NP have an important property: they embody the difficulty of all other problems in NP. These problems within the class NP are called NP-complete. If anyone of them has an efficient algorithm, then every problem in NP can be solved efficiently.

Time Complexity Function		Size n			
		10	20	30	40
Polynomial	n	0.00001 second	.00002 second	.00003 second	.00004 second
	n^2	.0001 second	.0004 second	.0009 second	.0016 second
	n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minutes
Exponential	2^n	.001 second	1.0 second	17.9 minutes	12.7 days
	3^n	.059 second	58 minutes	6.5 years	3855 centuries

Table 3.1: Comparison of Polynomial and Exponential Time Complexity Functions.
The Functions Express Execution Time in Terms of Microseconds [17].

Rigorously, the class NP consists of all those languages which can be recognized by a polynomial-time bounded nondeterministic Turing machine (NDTM). (The concepts of NDTM and languages can be found elsewhere [17,18]).

One way to prove that a problem is NP-complete is to show that it belongs to the class NP and that every other problem in NP can be polynomially transferred to the given problem (but-not necessarily the reverse). However, to use this approach, we need a problem that is proved to be NP-complete. Such a problem is provided by Cook's fundamental theorem. It is usually referred to as the "Satisfiability" problem and has been proven to be NP-complete [19].

In short, NP-complete problems are characterized by two important properties [16].

1. No NP-complete problem is known to be solvable by a polynomial time algorithm.
2. If a polynomial time algorithm is known for one of the NP-complete problems then one could obtain polynomial time algorithms for all the NP-complete problems and of course, all other problems in NP.

Discovering that a problem is NP-complete is usually just the beginning of the work on that problem. While the theory of NP-completeness does not provide inherent intractability, it does provide straightforward techniques for proving that a given problem is just as difficult as those for which all attempts to find an efficient algorithm have failed. It is then more appropriate to direct the efforts towards more productive

approaches - such as looking for efficient algorithms that solve for individual cases of the general problem. The alternatives to tackle the problem can be roughly divided into two categories [17].

The first alternative is to obtain as much improvement over straightforward exhaustive search as possible. The most widely used approaches include branch-and-bound techniques, dynamic programming, cutting plane methods, and Lagrangian techniques.

The other alternative is to focus on finding a good solution within an acceptable amount of time. Algorithms that do this are usually based on heuristics. The methods used for designing such algorithms tend to be rather problem specific. The most widely used technique is that of neighbourhood search in which a preselected set of local operations is used to repeatedly improve an initial solution, continuing until no further local improvements can be made and a locally optimum solution has been obtained. These approaches are quite successful in practice although it is not often possible to predict how well such algorithms will perform by formally analyzing them beforehand.

To summarize, folding and diagonalization problems (Chapter 4) have been proven to be NP-complete [13,29]. Therefore, instead of looking for an optimal solution, heuristics are employed to get a suboptimal solution. This is why only suboptimal solution has been found for folding, and why we will only look for suboptimal solution to the diagonalization problem of Chapter 4.

Chapter 4

A DIAGONALIZATION APPROACH TO THE PLA4.0 Introduction

In the previous chapter, several folding techniques applicable to PLAs were discussed. In this chapter, a new diagonalization technique to reduce the area of PLA personality matrices is presented. The technique attempts to rearrange the matrix in such a way that all the nonzero elements in the matrix are close to the diagonal. This technique is similar to bandwidth minimization techniques used in connection with sparse matrices [20,26]. However, the algorithms that have been developed are applicable only to symmetric and square matrices. Also, most of the bandwidth minimization algorithms were developed for matrices with smaller bandwidths and fairly regular structures. PLA matrices, in general, are characterized by larger bandwidths and fairly irregular structures [13]. However, the basic idea of the algorithms still remains the same, i.e., to rearrange the matrix by row and column permutations.

4.1 Bandwidth Minimization Problem

In this section several issues related to the bandwidth minimization problem are discussed. A matrix $A = (a_{ij})$, for which $a_{ij} = 0$, for $|i-j| > B$, is known as a band matrix. The bandwidth of a band matrix can then be defined as the maximum value, B , of $|i-j|$,

for which $a_{ij} \neq 0$. An example of a band matrix is shown in Figure 4.1. It is clear from the figure that the bandwidth, B , of the matrix is 2.

The bandwidth minimization problem can be stated with reference to graphs. A graph, $G(V,E)$, is a set of vertices, denoted by V , together with a set of edges, E , connecting the vertices. In particular, if $A = (a_{ij})$ is an $n \times n$ symmetric matrix, then the graph $G(V,E)$ associated with A has vertices numbered from 1 to n , with the vertex i connected from vertex j if and only if $a_{ij} \neq 0$. Figure 4.2a shows an example. A directed graph (or digraph) $G(V,E)$ is a set of vertices V together with a set of directed edges E . If $A = (a_{ij})$ is an $n \times n$ unsymmetric matrix, then there is a directed graph $G(V,E)$ associated with A such that the set of all the vertices in $G(V,E)$ is numbered from 1 to n and there is an edge from vertex i to vertex j if and only if $a_{ij} \neq 0$. Figure 4.2b shows an example.

The bandwidth minimization problem can then be stated as follows:

Let $G(V,E)$ be a directed graph and let $K \leq |V|$ be a positive integer. Then, is there a one-to-one function $f: V \rightarrow \{1, 2, \dots, |V|\}$ such that for all $(p, q) \in E$, $f(p) < f(q)$ and $(f(q) - f(p)) \leq K$?

Comment: Minimizing the bandwidth of a graph corresponds to permuting rows and columns of its corresponding PLA matrix to obtain the narrowest band of 1's around the diagonal.

This bandwidth minimization problem has been shown to be NP-complete [29].

Figure 4.1: A Band Matrix With Bandwidth Equal to 2.

Figure 4.2: (a) A Symmetric Matrix and Its Associated Graph
 (b) A Matrix and Its Associated Directed Graph.

4.2 An Algorithm For Diagonalization

Having known that the bandwidth minimization problem is NP-complete, there is almost no hope of getting an optimal solution to the problem. In other words, there is no guarantee that the bandwidth obtained by the algorithm is the minimum. However, the algorithm makes use of efficient heuristics and attempts to squeeze the sparse matrix into another matrix with a smaller bandwidth.

After having achieved the bandwidth reduction, the bandwidth-minimized PLA is oriented at an angle, say α . The angle depends upon the size of the PLA matrix. For example, for a square matrix α is 45 degrees, and for an $m \times n$ matrix α is equal to $\tan^{-1} [(m-1)/(n-1)]$. Figure 4.3 shows an example of a regular PLA and its diagonalized version.

The two major factors that determine the final bandwidth of the PLA matrix are the density of the matrix, and the choice of the starting row or column. The bandwidth reduction that can be achieved by the algorithm described in [26] is restricted by the maximum density of a row or a column. However, this might not produce a significant saving in the PLA area. For example, consider the matrix shown in Figure 4.4a. In this example, the 5th row is densely packed. Therefore, even with a very efficient algorithm the width of the diagonalized PLA remains almost the same as that of the original PLA. This problem, however, can be solved if one attempts the bandwidth reduction without considering the densely packed row or column. This row or column can be finally

Figure 4.3: A Regular PLA Matrix and Its Diagonalized Version.

The Area of Diagonalized PLA is $L_2 W_2$

The Area Reduction is $\frac{L_1 W_1 - L_2 W_2}{L_1 W_1}$

Figure 4.4: (a) An example PLA matrix which does not cover significantly less area when diagonalized because of the density of row 5.

(b) Matrix of (a), diagonalized with row 5 removed and placed beside the diagonal part.

placed along the side of the diagonalized PLA. This is illustrated in Figure 4.4b. There are two key issues involved in this. First, the interconnection problems become slightly involved. Second, it is important to observe if there is any significant saving in the area. This is left to the discretion of the designer.

As stated earlier, the main idea of the algorithm is to place all the nonzero elements in the matrix as close to the diagonal as possible. To ensure this, the first task should be to assign weights to all the elements in the matrix in such a way that the elements which are farther away from the diagonal have higher weights associated with them. For an $m \times n$ matrix, the weights $W(i,j)$, where $i = 1, \dots, m$, and $j = 1, \dots, n$, are calculated by the following simple formula:

$$W(i,j) = |(m-i) \cos \alpha - (n-j) \sin \alpha|$$

Table 4.1 shows the weights of the elements of a 5×5 matrix.

After assigning the weights to all the elements, the total weight W_T is calculated, i.e.,

$$W_T = \sum_{i=1}^m \sum_{j=1}^m W(i,j)$$

The element which has the highest weight associated with it is chosen to start with. The column (or row) which has this element is iteratively interchanged with another column (or row) in such a way that the difference $\{W_{T(\text{old})} - W_{T(\text{new})}\}$, where $W_{T(\text{old})}$ is the total weight W_T before the columns (or rows) are interchanged and $W_{T(\text{new})}$ is the total weight W_T after the columns (or rows) are interchanged, is maximum. By

	Column				
	1	2	3	4	5
1	0.0	0.707	1.414	2.121	2.828
2	0.707	0.0	0.707	1.414	2.121
3	1.414	0.707	0.0	0.707	1.414
4	2.121	1.414	0.707	0.0	0.707
5	2.828	2.121	1.414	0.707	0.0

Table 4.1: Weights Associated With the Elements of a 5 x 5 Matrix

experimental results, it was found that the algorithm converged faster by the above procedure than the method described in [22]. For example, consider the hypothetical 6x6 matrix shown in Figure 4.5a. A good algorithm should reorient this matrix in minimum number of steps (3 in this case) as illustrated in Figures 4.5b, 4.5c and 4.5d. The algorithm described reorients this hypothetical matrix in 3 steps. At this point, it may be necessary to note that interchanging an input and an output column may not always be desired because of interconnection problems. On the other hand, if input and output columns are mixed then a higher reduction in bandwidth is possible. The algorithm, therefore, provides the designer with the option as to whether the input and output columns can be mixed or not.

The procedure described in the above paragraph is repeated until no further improvement in the bandwidth is possible. Now, a matrix can be divided into two sides, i.e., one on each side of the diagonal. It is not necessary that the bandwidths on the two sides be the same since our main interest is to reduce the width of the diagonalized PLA. Therefore, it is important to reduce the bandwidths on both sides of the diagonal. After finding that it is not possible to reduce the bandwidth on one side, the algorithm attempts to reduce the bandwidth on the other side. This is achieved by again picking the element with the highest weight and repeating the procedure described in the earlier paragraph.

(a)

(b)

(c)

(d)

Figure 4.5: A Hypothetical Matrix and the Steps Necessary for Reorientation

The final step in the algorithm is the double-move minimization. The column (or row) that is making further minimization impossible is located and that particular column (or row) is interchanged with some other column (or row). This is done in such a way that the total weight is minimum. Then it may be possible to further reduce the bandwidth by starting again with the element with the highest weight. The algorithm halts if no improvement in the total weight is obtained.

The algorithm is briefly described as follows:

I. Initialization

Step 1: Assign weights $W(i,j)$, for $i = 1, \dots, m$, and $j = 1, \dots, n$, to all the elements in the $m \times n$ matrix.

Step 2: Calculate the column weights $W_C(j)$, for $j = 1, \dots, n$, where $W_C(j) = \sum_{i=1}^m W(i,j)$ and the row weights $W_R(i)$, for $i = 1, \dots, m$, where $W_R(i) = \sum_{j=1}^n W(i,j)$.

Step 3: Calculate the total weight $W_T = \sum_{i=1}^m \sum_{j=1}^n W(i,j)$.

II. Iteration

Step 1: Pick the element with the highest weight. Let the row and column numbers corresponding to that element be x and y respectively, i.e., $W(x,y) = \max \{W(i,j)\}$.

Step 2: For $j=1, \dots, n$, $j \neq y$, do the following: (perform a limited search within the present bandwidth).
(a) Interchange columns y and j .

(b) Calculate the new column weight, i.e., $W_{C,new}(y)$, and the difference $W_1 = W_{C,old}(y) - W_{C,new}(y)$.

(c) Interchange columns j and y .

(d) Pick the column j for which W_1 is maximum.

Step 3: Repeat step 2 for rows (i.e., x and i) and find the corresponding maximum $W_2 = W_{R,old}(x) - W_{R,new}(x)$.

Step 4: If W_2 is less than W_1 , then interchange the columns y and j , and update $W_C(y)$ and $W_C(j)$; else, interchange rows x and i , and update $W_R(x)$ and $W_R(i)$.

Step 5: Repeat the above steps till a column or row can be found which will improve W_T .

Step 6: If no improvement in W_T is possible, then note on which side of the diagonal the element with the highest weight lies. Pick the element with the highest weight on the other side of the diagonal.

Step 7: Repeat steps 1 to 5 till no further improvement in W_T is possible.

III. Double-Move Minimization

Step 1: Pick the element with the highest weight. Let it be $W(x,y)$. Make a neighbourhood search and interchange two other columns (say, a and b) so that columns y and a (or y and b) can be interchanged which will result in a maximum improvement in W_T .

Step 2: Interchange y and a (or y and b) and update the column weights. If no improvement is possible with columns, try changing rows. Repeat the above two steps on the other side of the diagonal.

The computation halts when W_T cannot be improved any further. The computation time required for assigning the weights and picking the element with the highest weight is of the order of n^2 for an $n \times n$ matrix. For searching a column (or a row), interchanging the elements in the columns (or rows), and updating the column or row weights the computation time required is of the order n .

4.3 Results

The diagonalization algorithm has been coded in FORTRAN and is listed in the appendix. The algorithm was applied to reduce the areas of 10 PLAs of varying sizes. The smallest PLA considered consists of 9 product terms and 20 input-output variables. A bandwidth reduction of 11 was achieved. The final diagonalized PLA area was 60% of the original area. Figure 4.6a shows a 40×42 PLA matrix and Figure 4.6b shows its corresponding diagonalized version.

The largest PLA considered had 75 product terms and 75 input-output variables. The bandwidth reduction achieved was 45. The area saving achieved was 32%. Table 4.2 summarizes the results obtained from the algorithm. A 40 to 45 percent saving in the area was typical for most of the PLAs considered. However, this result is not guaranteed for dense PLAs.

After a PLA is diagonalized it may then be folded. This will further reduce the area. While this was not extensively studied and no program was written for this second reduction, hand reduction of the diagonalized

ORIGINAL MATRIX

Figure 4.6a: A 40 x 42 PLA Matrix

NEW BAND MATRIX

Figure 4.6b: Diagonalized Reduction of the PLA in Figure 4.6a.

The area is 65% of the original.

PLA Size	Bandwidth Reduction Achieved	Reduced Size (% of the original)
9 x 20	11	60
19 x 27	17	52
22 x 12	14	56
33 x 54	34	50
75 x 75	45	68
40 x 50	30	49
40 x 42	25	65
25 x 25	16	56
60 x 75	46	62
75 x 35	43	53

Table 4.2: Results Obtained From the Algorithm

PLA was attempted.

It was noted that not only was further reduction possible, but also the diagonalization made the folding simpler.

Chapter 5

CONCLUSIONS5.1 Summary

A diagonalization technique to reduce the area of sparse PLA matrices was described in this thesis. An overview of PLAs was presented in Chapter 2. The design methodology involved in the design of PLA based systems was also discussed.

Folding is one technique to reduce the area of PLAs. Many kinds of folding are possible. To aid understanding, the folding techniques were classified into four different categories in Chapter 3. They are: column folding, row folding, mixed column and row folding, and constrained folding. All these different techniques were discussed. The major folding problem is to find maximum number of folds. This problem is known to be NP-complete. The concepts of NP-complete problems were also described.

A diagonalization technique, which attempts to reduce the area of PLAs, was presented in Chapter 4. This technique attempts to arrange all the elements of a PLA matrix as close to the diagonal as possible. In mathematical terminology, the technique attempts to minimize the bandwidth of a PLA matrix. The problem of minimizing the bandwidth of a matrix is NP-complete. Since no polynomial-time algorithms are known for the solution of NP-complete problems, several heuristics were employed in the algorithm.

A FORTRAN program to diagonalize a PLA was developed. The main features of the program include assigning weights to all the non-zero elements, interchanging the rows or columns that will maximize the difference between the total weights before and after iteration, and double-move minimization. The program was applied to 10 PLAs of varying sizes. An average reduction of 40 to 45 percent was achieved. Maximum area reduction achieved was 51% and the minimum was 32%. However, because of the nature of the problem, these results are not guaranteed for dense PLAs.

Folding techniques can be applied to a diagonalized PLA. Although no program was developed to perform folding on a diagonalized PLA, hand reduction was attempted. Based on this, two conclusions were drawn: First, results obtained by folding after diagonalization were comparable to those obtained by other folding techniques, and second, folding becomes simpler after diagonalization.

The maximum area reduction that can be achieved is limited to 50% in row folding and column folding, and 75% in mixed row and column folding. However, there is no such limit in the diagonalization technique.

The results (pertaining to area reduction) shown in Table 4.1 indicate the ideal case situation. However, this may not be the case in the real world since additional area may be required due to interconnections, power supply lines, load resistors etc. This is, of course, true with other folding techniques.

5.2 Future Work

The algorithm described in this thesis considers the AND plane matrix and the OR plane matrix as one single matrix and attempts to minimize the bandwidth. However, for PLAs where the output columns are very few in number compared to the input columns and densely packed, it might be a good idea to first diagonalize the AND plane and then the OR plane. To do this, it becomes necessary to put clear bounds on the number of inputs, the number of outputs, and the density of output columns. This might be an interesting area for further research.

As mentioned earlier in section 5.1, it was found that folding became simpler after diagonalization. Developing an algorithm to perform the same is another possible area for future research.

Finally, since the algorithm employs heuristics, it is possible to employ additional heuristics (for example, three-step look-ahead, better weighting scheme) which should eventually improve the overall performance of the algorithm.

REFERENCES

1. C.A. Mead, and L.A. Conway, Introduction to VLSI Systems, Addison Wesley, 1980.
2. W.N. Carr, and J.P. Maize, MOS/LSI Design and Application, Texas Instruments Electronics Series, New York: McGraw-Hill, 1972.
3. D. Destreicher, "PLASYS: Final Report", California Institute of Technology Computer Science Department Technical Report #3655, May 1980.
4. R. Ayres, "Silicon Compilation - A Hierarchical Use of PLAs", in Proceedings of the 16th ACM Design Automation Conference, pp.314-326, June 1979.
5. A.R. Newton, et al., "Design Aids for VLSI: The Berkeley Perspective", IEEE Trans. Circuits, Syst., Vol.CAS-28, No.7, pp.666-680, July 1981.
6. M.W. Stebinsky, et al., "A Fully Automatic Technology-Independent PLA Macrocell Generator", Proceedings of International Conference on Circuits and Computers, pp.156-160, 1982.
7. D.L. Ostapko, and S.J. Hong, "Fault Analysis and Test Generation for PLAs", IEEE Trans. on Computers, Vol.C-28, No.9, pp.617-627, September 1979.
8. W.W. Chu, "Automated Welding Customizes Programmable Logic Arrays", Electronics, pp.159-162, July 14, 1982.
9. S.J. Hong, et al., "MINI: A Heuristic Approach for Logic Minimization", IBM J. Res. Develop., Vol.18, pp.443-457, September 1974.
10. J.P. Roth, "Programmed Logic Array Optimization", IEEE Trans. on Comp., Vol.C-27, pp.174-176, February 1978.
11. H. Fleisher, and L. Maissel, "Introduction to Array Logic", IBM J. Res. Develop., Vol.19, pp.98-109, March 1975.
12. R.A. Wood, "A High Density Programmable Logic Array Chip", IEEE Trans. on Comp., Vol.C-28, No.9, pp.602-608, September 1979.
13. M. Luby, et al., "Some Theoretical Results on the Optimal PLA Folding Problem", Proc. of International Conference on Circuits and Computers, pp.165-170, 1982.

14. W. Grass, "A Depth-First Branch-And-Bound Algorithm for Optimal PLA Folding", Proc. of 19th ACM Design Automation Conference, pp.133-140, 1982.
15. G.D. Hachtel, et al., "Techniques for Programmable Logic Array Folding", Proc. of 19th ACM Design Automation Conference, pp.147-155, 1982.
16. M.R. Garey, and D.S. Johnson, "Strong NP-Completeness Results: Motivation, Examples, and Implications", J. ACM, Vol.25, No.3, pp.499-508, July 1978.
17. M.R. Garey, and D.S. Johnson, "Computers and Intractability - A Guide to the Theory of NP-Completeness", W.H. Freeman and Co., San Fransico, 1979.
18. A.V. Aho, et al., "The Design and Analysis of Computer Algorithms", Addison-Wesley, Reading, Mass., 1974.
19. S.A. Cook, "The Complexity of Theorem Proving Procedures", Proc. 3rd Annual ACM Symposium on Theory of Computing, pp.151-158, 1971.
20. K.Y. Cheng, "Minimizing the Bandwidth of Sparse Symmetric Matrices", Computing, Vol.11, pp.103-110, 1973.
21. R.P. Tewarson, "Row-Column Permutation of Sparse Matrices", Computer Journal, Vol.10, pp.300-305, 1967.
22. H.R. Grooms, "Algorithm for Matrix Bandwidth Reduction", J. of the Structural Div., Proc. Amer. Soc. Civil Eng., pp.203-214, January 1972.
23. E. Cuthill, and J. McKee, "Reducing the Bandwidth of Sparse Symmetric Matrices", Proc. ACM National Conference, pp.157-172, 1969.
24. G.G. Alway, and D.W. Martin, "An Algorithm for Reducing the Bandwidth of a Matrix of Symmetrical Configuration", Computer Journal, pp.264-272, 1965.
25. R. Rosen, "Matrix Bandwidth Minimization", Proc. ACM National Conference", pp.585-595, 1968.
26. P.T.R. Wang, "Bandwidth Minimization, Reducibility, Decomposition, and Triangularization of Sparse Matrices", Computer and Information Science Research Center, Ohio State University, Rep.OSU-CISRC-TR-73-5, 1973.
27. M.R. Garey, et al., "Some Simplified NP-Complete Problems", Proc. 6th Annual ACM Symposium on Theory of Computing, pp.47-63, 1974.

28. C.H. Papadimitrou, "The NP-Completeness of the Bandwidth Minimization Problem", Computing, Vol.16, pp.263-270, 1976.
29. M.R. Garey, et al., "Complexity Results for Bandwidth Minimization", SIAM J. Appl. Math., Vol.34, No.3, May 1978.
30. A. George, and J.W.H. Liu, "Computer Solution of Large Sparse Positive Definite Systems", Prentice-Hall Inc., N.J., 1981.
31. P. Cornell, "Cook's Theorem", The Graduate, University of Toronto, pp.15-17, January 1983.

APPENDIX

```

C *****
C *
C *   P L A   D I A G O N A L I Z A T I O N   P R O G R A M   *
C *
C *****
C
C THIS PROGRAM DIAGONALIZES A PLA MATRIX BY REDUUCING THE
C BANDWIDTH. THE PROGRAM FIRST READS THE NUMBER OF ROWS AND
C THE NUMBER OF COLUMNS AS NROWM AND NCOLM. THEN IT READS THE
C ELEMENTS OF THE PLA MATRIX. CHARACTERS (NOT BLANKS) ARE USED
C TO REPRESENT THE 1'S IN THE MATRIX. BLANK CHARACTERS
C ARE USED IN THE INPUT DATA FILE TO REPRESENT THE 0'S IN THE
C PLA MATRIX. THE MATRIX IS STORED IN THE ARRAY ELEMENTH. ROW
C AND COLUMN POINTERS ARE STORED IN THE ARRAYS ROWPH AND COLIH
C RESPECTIVELY. THE WEIGHTS FOR THE ELEMENTS IN THE MATRIX (WHETHER
C IT IS A 1 OR A 0) ARE CALCULATED AND STORED IN THE ARRAY STORE.
C THEN THE WEIGHTS FOR THE 1'S IN THE MATRIX ARE STORED IN WEIH.
C THIS IS DONE SIMPLY BY ASSIGNING THE WEIGHTS FROM STORE TO THE
C NONZERO ELEMENTS. THE TOTAL WEIGHTS FOR ALL THE ELEMENTS IN A ROW
C AND A COLUMN ARE CALCULATED AND STORED IN THE ARRAYS ROWEIH
C AND COWEIH RESPECTIVELY.
C
C READ(5,1) NROWM,NCOLM
1  FORMAT(2I2)
C CHARACTER *1 ELEMENTH(75,75),BUFFER(75),TEMP
C INTEGER ROWPH(75),COLIH(75),COUNTM,SIDEM,RDENS,CDENS,ROUNTH
C REAL WEIH(75,75),COWEIH(75),ROWEIH(75),STORE(75,75)
C LOGICAL COMPLETE
C
C NOW READ THE INFUT MATRIX AND STORE IT IN 'ELEMENTH'
C
C DO 10 I=1,NROWM
C READ(5,11) (BUFFER(J),J=1,NCOLM)
11  FORMAT(75A1)
C DO 12 K=1,NCOLM
C TEMP=BUFFER(K)
C ELEMENTH(I,K)=TEMP
12  CONTINUE
10  CONTINUE
C
C INITIALIZE THE ROW AND COLUMN FOINTERS
C
C DO 17 I=1,NROWM
17  ROWPH(I)=I
C DO 25 I=1,NCOLM
25  COLIH(I)=I
C WRITE(108,292)
292 FORMAT(1H1,40X,'O R I G I N A L   M A T R I X',//,10X,80(1H-),2X)

```

```

DO 285 I=1,NROWH
WRITE(108,286) ROWPH(I),(ELEMENTH(I,J),J=1,NCOLM)
286 FORMAT(10X,I2,1X,7SA1)
285 CONTINUE
C
C
40 CDENS=(NCOLM/2)
RDENS=(NROWH/2)
C
C
NOW PLACE THE DENSELY PACKED ROWS AND COLUMNS TO A SIDE.
THESE ROWS OR COLUMNS ARE NOT CONSIDERED FOR BANDWIDTH REDUCTION.
IF THERE IS NOT A DENSE ROW OR COLUMN THEN ROUNTH AND COUNTH WILL
BE EQUAL TO 1. THE VARIABLES RDENS AND CDENS INDICATE THE MAXIMUM
DENSITY ALLOWABLE IN A ROW OR A COLUMN FOR BANDWIDTH REDUCTION.
HOWEVER, THIS COULD BE CHANGED TO ANY DESIRED VALUE DEPENDING UPON
THE CONSTRAINTS.
C
CALL MANYC(ELEMENTH,NROWH,NCOLM,CDENS,COUNTH)
CALL MANYR(ELEMENTH,NROWH,NCOLM,RDENS,ROUNTH)
C
C
INITIALIZE ROW AND COLUMN WEIGHT ARRAYS
C
DO 165 I=1,NROWH
165 ROWEIM(I)=0.0
DO 164 I=1,NCOLM
164 COWEIM(I)=0.0
C
C
CALCULATE THE WEIGHTS
C
DO 83 I=ROUNTH,NROWH
DO 84 J=COUNTH,NCOLM
WEIM(I,J)=0.0
84 CONTINUE
83 CONTINUE
C
C
CALCULATE THE WEIGHTS OF ALL THE ELEMENTS IN THE MATRIX AND STORE
IT IN THE ARRAY STORE.
C
RN=NROWH-ROUNTH
CN=NCOLM-COUNTH
TANAL=RN/CN
ANGLE=ATAN(TANAL)
DO 209 J=COUNTH,NCOLM
MM=J-COUNTH
DO 209 I=ROUNTH,NROWH
II=I-ROUNTH
209 STORE(I,J)=ABS((RN-II)*COS(ANGLE)-(CN-MM)*SIN(ANGLE))
208 CONTINUE
C
C
THE ARRAY WEIM CONTAINS THE WEIGHTS FOR THE 1'S IN THE PLA MATRIX.
C
DO 283 I=ROUNTH,NROWH
DO 284 J=COUNTH,NCOLM
IF (ELEMENTH(I,J).EQ.' ') GO TO 284
WEIM(I,J)=STORE(I,J)
284 CONTINUE
283 CONTINUE
C
C
FIND THE COLUMN WEIGHTS.
C

```

```

DO 166 J=COUNTM,NCOLM
DO 167 I=ROUTM,NROWM
COWEIM(J)=COWEIM(J)+WEIM(I,J)
167 CONTINUE
166 CONTINUE
C
C FIND THE ROW WEIGHTS.
C
DO 168 J=ROUTM,NROWM
DO 169 I=COUNTM,NCOLM
ROWEIM(J)=ROWEIM(J)+WEIM(I,J)
169 CONTINUE
168 CONTINUE
C
COMPLETE=.FALSE.
SIDEH=3
CALL DIAGONAL(ELEMENTM,NROWM,NCOLM,ROUTM,COUNTM
$,WEIM,COWEIM,ROWEIM,ROWPM,COLIM,COMPLETE,JCM,LCH,SIDEH,STORE)
215 IF(COMPLETE) GO TO 999
C
C TRY TO MINIMIZE THE BANDWIDTH FROM THE OTHER SIDE.
C
CALL WHICHSIDE(TANAL,ROUTM,COUNTM,JCM,LCH,SIDEH)
IF (SIDEH.EQ.2) GO TO 240
SIDEH=2
GO TO 241
240 SIDEH=1
241 CALL DIAGONAL(ELEMENTM,NROWM,NCOLM,ROUTM,COUNTM
$,WEIM,COWEIM,ROWEIM,ROWPM,COLIM,COMPLETE,JCM,LCH,SIDEH,STORE)
999 WRITE(108,271) (COLIM(I),I=1,NCOLM)
271 FORMAT(1H1,10X,/,10X,' THE ARRANGED COLUMN NUMBERS ARE ',/,
$,10X,25I3,/,10X,25I3,/,10X,25I3)
WRITE(108,294)
294 FORMAT(1H1,40X,'N E W B A N D M A T R I X',/,10X,80(1H-),2X)
DO 287 I=1,NROWM
WRITE(108,286) ROWPM(I),(ELEMENTM(I,J),J=1,NCOLM)
287 CONTINUE
STOP
END
C
C
C THE SUBROUTINE DIAGONAL DIAGONALIZES THE ELEMENTS OF THE MATRIX
C STORED IN ELEMENT,NROW AND NCOL REPRESENT THE NUMBER OF ROWS AND
C COLUMNS RESPECTIVELY.WEI CONTAINS THE WEIGHTS ASSOCIATED WITH THE
C ELEMENTS IN THE MATRIX.COWEI AND ROWEI CONTAIN THE COLUMN AND
C ROW WEIGHTS.ROWP AND COLI ARE ROW AND COLUMN POINTER ARRAYS.JC,LC
C ARE THE COLUMN AND ROW NUMBERS OF THE ELEMENT WITH THE HIGHEST
C WEIGHT.SIDE INDICATES ON WHICH SIDE OF THE DIAGONAL THE ELEMENT
C WITH THE HIGHEST WEIGHT LIES.
C
C
C SUBROUTINE DIAGONAL(ELEMENT,NROW,NCOL,ROUT,COUNT
$,WEI,COWEI,ROWEI,ROWP,COLI,THEEND,JC,LC,SIDE,STORE)
C *****
C CHARACTER *1 ELEMENT(75,75)
C INTEGER ROWP(75),COLI(75),COUNT,ROUT,MARO(75),MACO(75),SIDE
C REAL WEI(75,75),MAX,WEIM,WENEW,COWEI(75),ROWEI(75)
C $,REMP,CEMP,XEMP,ZEMP,DUMMY(75,75),STORE(75,75)
C LOGICAL LOGIC,THEEND

```

```

C
C FIND THE ELEMENT WITH MAX WEIGHT, IF THERE IS MORE THAN ONE
C ELEMENT WITH THE SAME MAXIMUM WEIGHT, THEN STORE THE ROW AND
C COLUMN NUMBERS OF THOSE ELEMENTS IN MAROW AND MACOL. IF NO ROW
C OR COLUMN IS FOUND FOR INTERCHANGING, THEN TRY THE NEXT ELEMENT
C WITH THE SAME MAXIMUM WEIGHT.
C
  KK=1
232 CALL MAXIM(ELEMENT, NROW, NCOL, ROUNT, COUNT, MAROW, MACOL, MAX,
  WEI, TANALP, SIDE, ICONST)
132 JC=MACOL(KK)
  LC=MAROW(KK)
  LOGIC=.FALSE.
  MSTORE=0
  WEMIN=999999.0

C
C FIND THE COLUMN WHICH WHEN INTERCHANGED MAXIMIZES THE DIFFERENCE
C BETWEEN THE OLD AND THE NEW COLUMN WEIGHTS. STORE THE SAME IN MSTORE.
C JC IS THE COLUMN NUMBER OF THE ELEMENT WITH THE HIGHEST WEIGHT.
C
  DO 126 JK =COUNT, NCOL
    IF (JK, EQ, JC) GO TO 126
    CEMP=COWEI(JC)
    XEMP=COWEI(JK)
    CALL CHACOL(JC, JK, ELEMENT, NROW)
    CALL COWAIT(ELEMENT, NROW, ROUNT, JK, COWEI, DUMMY, STORE, WM1)
    IF (WM1, GE, MAX) GO TO 205
    CALL COWAIT(ELEMENT, NROW, ROUNT, JC, COWEI, DUMMY, STORE, WM3)
    IF (WM3, GE, MAX) GO TO 205
    WENEW=COWEI(JC)+COWEI(JK)
    IF (WEMIN, GT, WENEW) GO TO 128
205 CALL CHACOL(JC, JK, ELEMENT, NROW)
    COWEI(JC)=CEMP
    COWEI(JK)=XEMP
    GO TO 126
128 WEMIN=WENEW
    MSTORE=JK
    CALL CHACOL(JC, JK, ELEMENT, NROW)
    COWEI(JC)=CEMP
    COWEI(JK)=XEMP
126 CONTINUE

C
C FIND THE ROW WHICH WHEN INTERCHANGED MAXIMIZES THE DIFFERENCE
C BETWEEN THE OLD AND THE NEW ROW WEIGHTS. STORE THE SAME IN MSTORE.
C LC IS THE ROW NUMBER OF THE ELEMENT WITH THE HIGHEST WEIGHT.
C
  DO 127 LK=ROUNT, NROW
    IF (LK, EQ, LC) GO TO 127
    REMP=ROWEI(LC)
    ZEMP=ROWEI(LK)
    CALL CHAROW(LC, LK, ELEMENT, NCOL)
    CALL ROWAIT(ELEMENT, NCOL, COUNT, LK, ROWEI, DUMMY, STORE, WM2)
    IF (WM2, GE, MAX) GO TO 206
    CALL ROWAIT(ELEMENT, NCOL, COUNT, LC, ROWEI, DUMMY, STORE, WM4)
    IF (WM4, GE, MAX) GO TO 206
    WENEW=ROWEI(LC)+ROWEI(LK)
    IF (WEMIN, GT, WENEW) GO TO 129
206 CALL CHAROW(LC, LK, ELEMENT, NCOL)
    ROWEI(LC)=REMP
    ROWEI(LK)=ZEMP

```



```

GO TO 127
129 WEMIN=WENEW
    MSTORE=LK
    LOGIC=,TRUE.
    CALL CHAROW(LC,LK,ELEMENT,NCOL)
    ROWEI(LC)=REMP
    ROWEI(LK)=ZEMP
127 CONTINUE
C
C IF NO ROW OR COLUMN IS FOUND FOR INTERCHANGING, THEN TRY
C DOUBLE-MOVE MINIMIZATION.
    IF (MSTORE.NE.0) GO TO 258
    CALL WHICHSIDE(TANALP,ROUT, COUNT, JC, LC, MYSIDE)
    IF (MYSIDE.EQ.1) GO TO 256
C THEREFORE IT IS ON SIDE 2
    INIT1=JC-1
    IFIN1=COUNT
    ISTEP1=-1
    INIT2=ROUT
    IFIN2=LC-1
    ISTEP2=1
    INIT3=LC+1
    IFIN3=NROW
    ISTEP3=1
    IFIN4=NCOL
    ISTEP4=-1
256 GO TO 257
    INIT1=JC+1
    IFIN1=NCOL
    ISTEP1=1
    INIT2=NROW
    IFIN2=LC+1
    ISTEP2=-1
    INIT3=LC-1
    IFIN3=ROUT
    ISTEP3=-1
    IFIN4=COUNT
    ISTEP4=-1
257 DO 250 I=INIT1,IFIN1,ISTEP1
    IF (ELEMENT(LC,I).NE.' ') GO TO 250
    INCR=0
    DO 251 J=INIT2,IFIN2,ISTEP2
    IF (ELEMENT(J,I).EQ.' ') GO TO 251
    INCR=INCR+1
    IF (INCR.GT.1) GO TO 250
    JROW1=J
251 CONTINUE
    IF (INCR.EQ.0) GO TO 250
    DO 252 KR=INIT3,IFIN3,ISTEP3
    DO 253 KC=I,IFIN4,ISTEP4
    IF (ELEMENT(KR,KC).NE.' ') GO TO 252
253 CONTINUE
    KROW1=KR
    CALL CHAROW(JROW1,KROW1,ELEMENT,NCOL)
    CALL ROWAIT(ELEMENT,NCOL,COUNT,JROW1,ROWEI,DUMMY,STORE,W1)
    IF (W1.LT.MAX) GO TO 254
    CALL CHAROW(JROW1,KROW1,ELEMENT,NCOL)
    CALL ROWAIT(ELEMENT,NCOL,COUNT,JROW1,ROWEI,DUMMY,STORE,W1)
    GO TO 252

```

```

254 CALL ROWAIT(ELEMENT,NCOL,COUNT,KROW1,ROWEI,DUMMY,STORE,W2)
    IF (W2.GE.MAX) GO TO 255
    CALL CHAROW(JROW1,KROW1,ELEMENT,NCOL)
    CALL ROWAIT(ELEMENT,NCOL,COUNT,JROW1,ROWEI,DUMMY,STORE,DUM)
    CALL ROWAIT(ELEMENT,NCOL,COUNT,KROW1,ROWEI,DUMMY,STORE,DUM)
    MSTORE=JROW1
    LC=KROW1
    LOGIC=.TRUE.
    GO TO 247
255 CALL CHAROW(JROW1,KROW1,ELEMENT,NCOL)
    CALL ROWAIT(ELEMENT,NCOL,COUNT,JROW1,ROWEI,DUMMY,STORE,DUM)
    CALL ROWAIT(ELEMENT,NCOL,COUNT,KROW1,ROWEI,DUMMY,STORE,DUM)
252 CONTINUE
    GO TO 247
250 CONTINUE
247 IF (MSTORE.NE.0) GO TO 258
    KK=KK+1
    IF (KK.GE.ICNST) GO TO 220
    GO TO 132

```

```

C
C ROW OR COLUMN IS FOUND FOR INTERCHANGING.
C

```

```

258 IF (LOGIC) GO TO 130
    CALL CHACOL(MSTORE,JC,ELEMENT,NROW)
    ICO=COLI(MSTORE)
    COLI(MSTORE)=COLI(JC)
    COLI(JC)=ICO
    CALL COWAIT(ELEMENT,NROW,ROUNT,MSTORE,COWEI,WEI,STORE,DUM)
    CALL COWAIT(ELEMENT,NROW,ROUNT,JC,COWEI,WEI,STORE,DUM)
    GO TO 131
130 CALL CHAROW(MSTORE,LC,ELEMENT,NCOL)
    IRO=ROWP(MSTORE)
    RWP(MSTORE)=ROWP(LC)
    RWP(LC)=IRO
    CALL ROWAIT(ELEMENT,NCOL,COUNT,MSTORE,ROWEI,WEI,STORE,DUM)
    CALL ROWAIT(ELEMENT,NCOL,COUNT,LC,ROWEI,WEI,STORE,DUM)
131 CALL CHECK(ELEMENT,NROW,NCOL,MO,ROUNT,COUNT)
    KK=1
215 IF (MO.GT.0) GO TO 232
    THEEND=.TRUE.
220 RETURN
    END

```

```

C
C THE FOLLOWING SUBROUTINE INTERCHANGES TWO COLUMNS.
C ITE,ICO ARE COLUMN NUMBERS;ELE IS THE CHARACTER MATRIX;
C NR IS THE NUMBER OF ROWS.
C

```

```

SUBROUTINE CHACOL(ITE,ICO,ELE,NR)
*****
C
CHARACTER *1 ELE (75,75), STEM
DO 30 I=1,NR
    STEM=ELE(I,ITE)
    ELE(I,ITE)=ELE(I,ICO)
    ELE(I,ICO)=STEM
30 CONTINUE
RETURN
END
C

```

```

C THE FOLLOWING SUBROUTINE INTERCHANGES TWO ROWS.IRO1,IRO2 ARE
C ROW NUMBERS.ELM IS THE CHARACTER MATRIX.NCL IS THE NO.OF COLUMNS.
C
C SUBROUTINE CHAROW(IRO1,IRO2,ELM,NCL)
C *****
C
C CHARACTER *1, ELM(75,75),STU
C DO 54 JR=1,NCL
C STU=ELM(IRO1,JR)
C ELM(IRO1,JR)=ELM(IRO2,JR)
C ELM(IRO2,JR)=STU
54 CONTINUE
C RETURN
C END
C
C THIS SUBROUTINE CHECKS THE ELEMENTS OUTSIDE THE DIAGONAL,IF THERE
C IS ANY ELEMENT WHICH IS NOT THE DIAGONAL,THEN MOUT WILL BE 1.
C
C SUBROUTINE CHECK(LEM,KROW,KCOL,MOUT,JROUN,JCOUN)
C *****
C
C CHARACTER *1 LEM(75,75)
C TR=FLOAT(KROW)
C TC=FLOAT(KCOL)
C ER=FLOAT(JROUN)
C EC=FLOAT(JCOUN)
C VAG=(TC-EC)/(TR-ER)
C DO 60 I=JROUN,KROW
C DO 61 J=JCOUN,KCOL
C IF(LEM(I,J).EQ.' ') GO TO 61
C Y=FLOAT(I)
C X=EC+(Y-ER)*VAG
C IF (FLOAT(J).NE.X) GO TO 68
61 CONTINUE
60 CONTINUE
C MOUT=0
C GO TO 110
68 MOUT=1
110 RETURN
C END
C
C SUBROUTINE TO FIND AND PUT THE DENSELY PACKED COLUMNS TO A SIDE.
C
C SUBROUTINE MANYC(MENT,NIRO,NICO,JWHAT,ICOUNT)
C *****
C CHARACTER*1 MENT(75,75)
C ICOUNT=1
C DO 29 K=1,NICO
C ISUM=0
C DO 28 I=1,NIRO
C IF (MENT(I,K).NE.' ')ISUM=ISUM+1
28 CONTINUE
C IF(ISUM.LE.JWHAT) GO TO 29
C ITEMP=K
C CALL CHACOL(ITEMP,ICOUNT,MENT,NIRO)
C ICOUNT=ICOUNT+1
29 CONTINUE
C RETURN

```

```

      END
C
C
C      SUBROUTINE TO FIND AND PUT DENSELY PACKED ROWS TO A SIDE.
C
C      SUBROUTINE MANYR(MEN,NIR,NIC,MWHAT,LCOUNT)
C      *****
C
C      CHARACTER*1 MEN(75,75)
C      LCOUNT=1
C      DO 71 K1=1,NIR
C      KSUM=0
C      DO 72 K2=1,NIC
C      IF (MEN(K1,K2).NE.' ')KSUM=KSUM+1
72  CONTINUE
C      IF (KSUM.LE.MWHAT) GO TO 71
C      KEMP=K1
C      CALL CHAROW(LCOUNT,KEMP,MEN,NIC)
C      LCOUNT=LCOUNT+1
71  CONTINUE
C      RETURN
C      END
C
C      THE SUBROUTINE MAXIM PICKS THE ELEMENT WITH THE HIGHEST WEIGHT.
C      IF THERE IS MORE THAN ONE ELEMENT WITH THE SAME MAXIMUM WEIGHT
C      THEN THE ROW AND COLUMN NUMBERS OF THE SAME WILL BE STORED IN
C      THE ARRAYS NOTER AND NOTEC.
C
C      SUBROUTINE MAXIM(CEMENT,KYRO,KYCO,ROWNT,COLNT,NOTER,NOTEC,WMAX,
C      $WAIT,TANPI,JMSIDE,K)
C      *****
C
C      CHARACTER *1 CEMENT(75,75)
C      INTEGER NOTER(75),NOTEC(75),ROWNT,COLNT
C      REAL WAIT(75,75)
C      DO 116 I=ROWNT,KYRO
116  NOTER(I)=0
C      DO 117 J=COLNT,KYCO
117  NOTEC(J)=0
C      WMAX=0.0
C      K=1
C      DO 113 I=ROWNT,KYRO
C      DO 114 J=COLNT,KYCO
C      IF (JMSIDE.EQ.3) GO TO 270
C      CALL WHICHSIDE(TANPI,ROWNT,COLNT,J,I,JSIDE)
C      IF (JSIDE.NE.JMSIDE) GO TO 114
270  IF (WAIT(I,J).LT.WMAX) GO TO 114
C      IF (WAIT(I,J).NE.WMAX) GO TO 115
C      WMAX=WAIT(I,J)
C      NOTER(K)=I
C      NOTEC(K)=J
C      K=K+1
C      GO TO 114
115  K=1
C      NOTER(K)=I
C      NOTEC(K)=J
C      WMAX=WAIT(I,J)
C      K=K+1
114  CONTINUE
113  CONTINUE

```

```

      RETURN
      END
C
C   THE SUBROUTINE COWAIT CALCULATES THE COLUMN WEIGHT OF COLUMN MK.
C
C   SUBROUTINE COWAIT(CELMEN,ISRO,KROUT,MK,WECO,DCAIT,STOREC,WEHC)
C   *****
C
C   CHARACTER *1 CELMEN(75,75)
C   REAL WECO(75),DCAIT(75,75),WEHC,WAM,STOREC(75,75)
C   DO 170 I=KROUT,ISRO
170  DCAIT(I,MK)=0.0
C   WECO(MK)=0.0
C   WEHC=0.0
C   DO 140 I=KROUT,ISRO
C   IF(CELMEN(I,MK).EQ.' ') GO TO 140
C   DCAIT(I,MK)=STOREC(I,MK)
C   WAM=DCAIT(I,MK)
C   IF(WAM.GT.WEHC) WEHC=WAM
C   WECO(MK)=WECO(MK)+DCAIT(I,MK)
140  CONTINUE
C   RETURN
C   END
C
C   SUBROUTINE ROWAIT CALCULATES THE ROW WEIGHT OF ROW IK.
C
C   SUBROUTINE ROWAIT(RELMEN,ITCO,KCOWT,IK,WERO,DRAIT,STORER,WEHR)
C   *****
C
C   CHARACTER *1 RELMEN(75,75)
C   REAL WERO(75),DRAIT(75,75),WEHR,STORER(75,75)
C   WERO(IK)=0.0
C   DO 171 I=KCOWT,ITCO
171  DRAIT(IK,I)=0.0
C   WEHR=0.0
C   DO 144 I=KCOWT,ITCO
C   IF(RELMEN(IK,I).EQ.' ') GO TO 144
C   DRAIT(IK,I)=STORER(IK,I)
C   WWH=DRAIT(IK,I)
C   WERO(IK)=WERO(IK)+DRAIT(IK,I)
C   IF(WWH.GT.WEHR) WEHR=WWH
144  CONTINUE
C   RETURN
C   END
C
C   SUBROUTINE WHICHSIDE TELLS ON WHICH SIDE OF THE DIAGONAL THE
C   ELEMENT WITH ROW AND COLUMN NUMBERS LCS AND JCS LIES.
C
C   SUBROUTINE WHICHSIDE(TANTH,ROUNTS,COUNTS,JCS,LCS,SIDES)
C   *****
C
C   INTEGER ROUNTS,COUNTS,SIDES
C   YAXIS=TANTH*(JCS-COUNTS)+ROUNTS
C   IF (YAXIS.LT.LCS) GO TO 245
C   SIDES=2
C   GO TO 246
245  SIDES=1
246  RETURN
C   END

```

END

0	6	1	0	9	1	8	4
---	---	---	---	---	---	---	---

FIN