CANADIAN THESES

THÈSES CANADIENNES

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 an inferior 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.

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 qualité inférieure.

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.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

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

THE UNIVERSITY OF ALBERTA

PROGRAMMABLE LOGIC ARRAY (PLA) FOLDING

by

(C) JAMES SAVARIMUTHU

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

EDMONTON, ALBERTA

FALL 1986

ISBN 0-315-32306-X

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR JAMES SAVARIMUTHU

TITLE OF THESIS PROGRAMMABLE LOGIC ARRAY (PLA) FOLDING

DEGREE FOR WHICH THESIS WAS PRESENTED MASTER OF SCIENCE

YEAR THIS DEGREE GRANTED FALL 1986

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(SIGNED) ...S. James.....

PERMANENT ADDRESS:

#611, 17319-69 AVE.....

EDMONTON, ALBERTA

CANADA T5T 3S7

DATED 30 SEPTEMBER 1986.

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled PROGRAMMABLE LOGIC ARRAY (PLA) FOLDING submitted by JAMES SAVARIMUTHU in partial fulfilment of the requirements for the degree of MASTER OF SCIENCE.

...Jack Mowchenko....

Supervisor

.....

.....

Date..Sept..10,.1986.....

Abstract

VLSI circuits and systems are so complex that they demand a structured design approach with a high degree of automation. Programmable Logic Arrays (PLAs) satisfy this demand and because of their functional generality and regular structure, PLAs are extensively used in VLSI circuits to implement combinational and sequential logic. The use of PLAs in VLSI circuits however, presents a major drawback in poor silicon area utilization. This thesis deals with the problem of reducing the PLA area.

One technique which when utilized fully, results in significant reduction of silicon area of PLAs, is folding. The major result presented is a new folding algorithm. Experimental results show that compared to the most popular PLA folding algorithm, our algorithm gives better results on 20 test PLAs while using about the same amount of computer time.

The minor results presented are a dynamic CMOS PLA generator and a quick state assignment method for finite state machines.

Acknowledgement

I would like to thank my thesis supervisor, Dr. Jack Mowchenko, for his academic support, guidance and financial assistance throughout the past two years. I would also like to thank my examination committee members, Dr. Ron Lawson, Dr. N. Durdle and Dr. Mark Green for their comments on my thesis.

I am deeply grateful to my friend Sr. Celine Graf, my sister Leema Rose and my brother-in-law Mr. Arulnayagam for their love and financial help for my studies in Canada. I am also thankful to my friend Lynn Gaetz and her parents for their love, understanding and help in the times of need.

I dedicate this thesis work to Rev. Dr. Justin Diraviam, the former Arch-Bishop of Madurai, India, who has been a great friend and the source of inspiration all my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

As a result of improvements in fabrication technology, Very Large Scale Integrated (VLSI) circuits have become so dense that a single silicon chip may contain hundreds of thousands of transistors. Many VLSI chips, such as microprocessors, now consist of multiple complex subsystems. With the increase in the number of transistors in a single chip, the complexity of designing a VLSI circuit also has increased dramatically. The use of hierarchy in the design processes, a regular layout strategy and the effective use of computer aids can considerably reduce the complexity of VLSI design. One method of implementing circuitry which takes advantage of regular layout and a high degree of automation is the use of array logic.

The term "array logic" refers to a one- or two-dimensional memory-like structure which can be programmed to realize an arbitrary set of combinational functions. The function of an array logic structure is similar to a lookup table in which the input signals are used as "address bits" to extract pre-determined results stored in the array. Examples of array logic include Weinberger arrays [Wein 67], Storage-Logic Arrays (SLAs) [Pati 79], Read-Only-Memory (ROM), and Programmable Logic Arrays (PLAs). The most popular and commonly used of these

is the Programmable Logic Array (PLA).

Figure 1.1 shows a basic PLA structure in schematic form. A PLA has a highly regular structure consisting of two adjacent rectangular arrays called the AND (input) array and the OR (output) array. The OR array is structurally identical to the AND array but rotated 90°. Columns in the AND array represent input signals and their complements. Columns in the OR array represent output signals. Rows in both the AND and OR arrays represent product terms formed by the input signals. The input buffers in the AND array isolate the PLA from the outside circuitry and also provide the complements of the input signals. The inverters attached to the output lines buffer the output signals. Figure 1.1 shows the PLA structure at the symbolic level. This symbolic representation is not bound to any particular IC technology or physical implementation. It does however, define the topological organization of the layout.

PLAs provide a direct implementation of combinational logic expressed in two-level Boolean sums-of-products form. Sequential logic in classical Moore or Mealy style also can be implemented in PLAs with feedback paths. The AND array selectively produces the necessary product terms for a sum-of-products realization of the desired Boolean functions. OR array then, selectively produces sums of product terms, thereby generating the PLA's outputs. These selection processes are called *personalizing* or *programming* the PLA, and are accomplished by placing or connecting

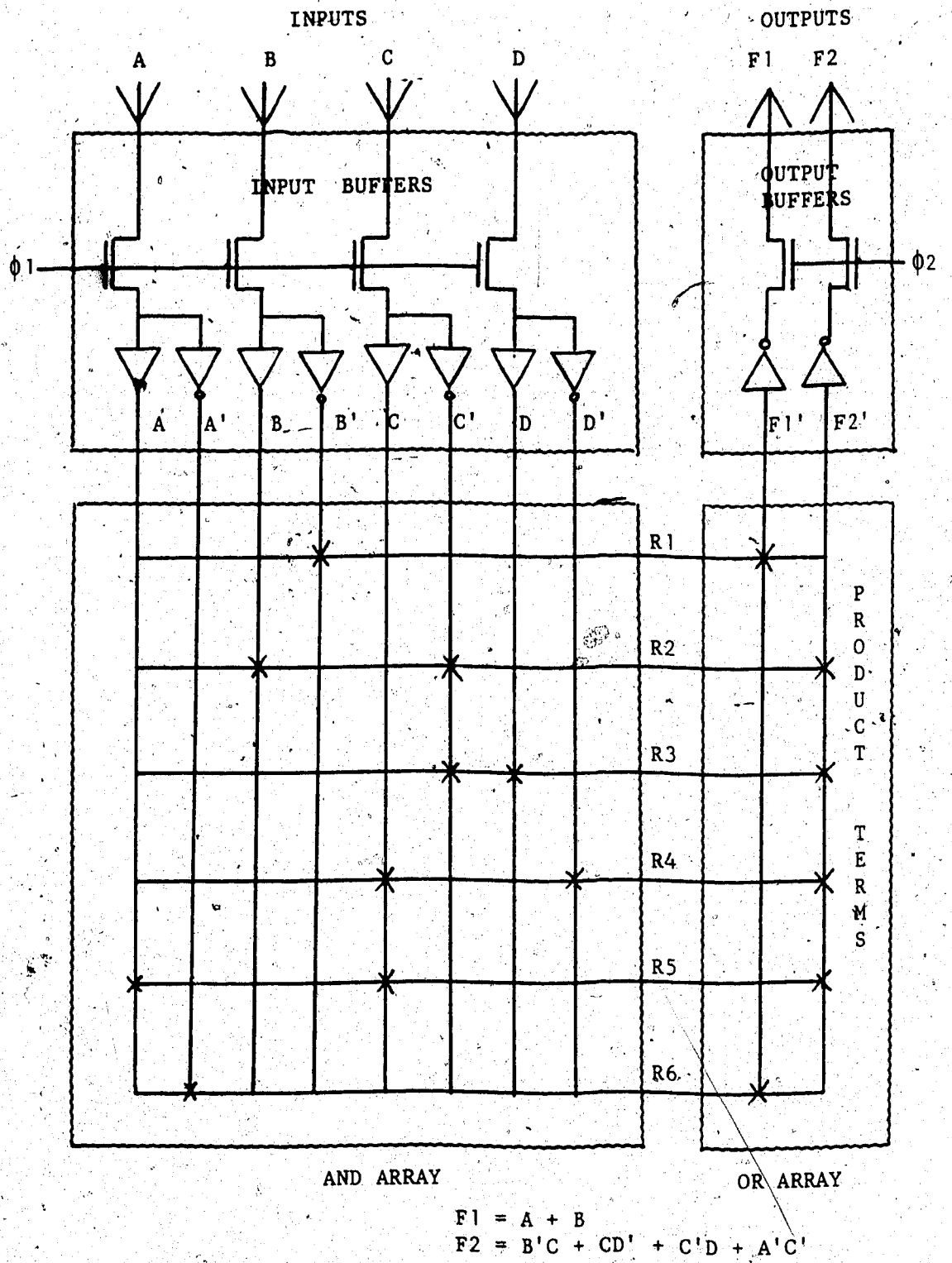


Fig. 1.1 - Basic PLA structure

appropriate devices at the row-column intersections, which are indicated by crosses in Figure 1.1.

A cross (x) in position (i,j) in the AND array indicates that the input represented by the column j is present in the product term represented by the row i. This is implemented by placing a transistor at the row-column intersection (i,j) to which the input j forms the gate. A cross in position (i,j) in the OR array indicates that the product term represented by the row i is present in the output term represented by the column j. This is implemented by placing a transistor at the row-column intersection (i,j) to which the product term i forms the gate.

The way in which a PLA is programmed can be described in symbolic form by a matrix called PLA *personality matrix*. Figure 1.2 shows the personality matrix of the example PLA in Figure 1.1. The left part in Figure 1.2 represents the AND array and the right part represents the OR array. In the

A	B	C	D	F1	F2
-	1	-	-	1	0
-	0	1	-	0	1
-	-	1	0	0	1
-	-	0	1	0	1
0	-	0	-	0	1
1	-	-	-	1	0

F1 = A + B'

F2 = B'C + CD' + C'D + A'C'

Fig. 1.2 - Personality matrix of the PLA in Figure 1.1

AND array, a '1' in the position (i,j) means that the jth input is present in the ith product term, and a '0' in that position means that the complement of the jth input is present in the ith product term. A '-' indicates that the ith product term does not depend on the input j. In the OR array, a '1' in the position (i,j) means that ith product term is present in jth output and a '0' in that position indicates that jth output does not depend on the ith product term. For example, the PLA personality matrix in Figure 1.2 illustrates that the inputs A and B are present in the product terms R1 and R2 respectively and these two product terms are present in the output term F1. Output F2 is the sum of the product terms B'C, CD', C'D and A'C'.

1.1 Advantages and Disadvantages of PLAs

Because of their highly regular layout structure and ease of the design, PLAs offer many advantages compared to custom random logic design. For VLSI applications, fast turn-around and low design cost achieved through design automation make PLAs very attractive. Unlike ROMs, which contain entries for all possible minterms, a specific PLA structure need contain only a row of circuit elements for each of the product terms that are actually required to implement a given set of logic functions. Hence, PLAs implement logic functions in much less area than ROMs. The most significant advantage of the regular structure is that the layout of a PLA can be automatically generated from the

PLA personality matrix.

Industry has availed itself of these advantages. Large PLAs used in industrial applications may have as many as 50 inputs, 50 outputs and 300 product terms and it is anticipated that much larger PLAs will be used in custom VLSI circuits in the near future. Successful applications of PLAs as stand-alone chips [Logu 75] and as constructs within large LSI chips [Cook 79], [Logu 81] have been reported. For example, Signetics Corporation, one of the first to market an uncommitted PLA, describes uses for their 821XX series [Sign 79] as a sequential controller in a parity checker. Large state-of-the-art microprocessor chips, for example the Motorola 68000, Intel 8086, Hewlett-Packard 32-bit and the Bell Mac-32 microprocessors, have many PLA structures implementing control logic. PLAs are also used to implement code conversions, microprogram address conversions, decision tables, bus priority resolvers, counters, decoders [Carr 72] and even small arithmeric logic units [Schm 80].

Unfortunately, the advantages cited above are significantly offset or negated by poor silicon utilization in many applications. PLA *density* is defined as the total number of crosses (transistors) used to personalize the PLA divided by the number product terms times the number of inputs and outputs, *i.e.* the number of "cross-points" in the PLA. PLA *sparsity* is defined as the reciprocal of PLA density. It has been reported that for typical applications, the PLA density is only about 10-20% which means about

80-90% of total PLA area is unused and thus wasted [Wood 79]. This is because a major portion of the PLA area is occupied by interconnections (cross-points without transistors) which do not directly contribute to the logic function of the PLA. Wasted area also degrades the performance of the circuit by introducing parasitic resistances and capacitances. Therefore, methods to reduce the area of the PLAs are of critical interest. In this thesis, we will consider one particular method of reducing PLA area.

1.2 The PLA Design Process

In order to see how PLA area may be reduced, let us first consider the steps in the PDA design process. As shown in Figure 1.3, implementing combinational logic as a PLA can be divided into three major subtasks: functional design, topological design and physical design.

Functional design is the process of translating the designer's specification of the function to be performed by a PLA into a set of two-level sum-of-products Boolean equations. Topological design is the process of transforming the Boolean equations into a toplogical representation of the PLA structure, such as a PLA personality matrix or a stick diagram, which describes the information necessary to program the PLA. The physical design is the translation of the topological representation into the mask layout of the PLA. The final layout will depend on the fabrication

technology being used. PLA area can be reduced in each of the three design steps. At the physical design level the PLA area can be reduced by using minimum dimensions for various mask layers wherever possible. The most significant area reductions can be made in functional and topological designs. In the following paragraph, we will look at the area reduction techniques which can be applied in the functional and topological design steps.

The area of a PLA is directly related to the number of inputs, the number of outputs, and the number of product terms. PLA *functional optimization* is the process of minimizing each of these quantities. Any logic minimization algorithm which properly handles "don't care" conditions should be able to eliminate redundant input and output variables. Therefore separate algorithms to detect redundant input and output variables are not really necessary. Consequently, the main aim of functional optimization is to

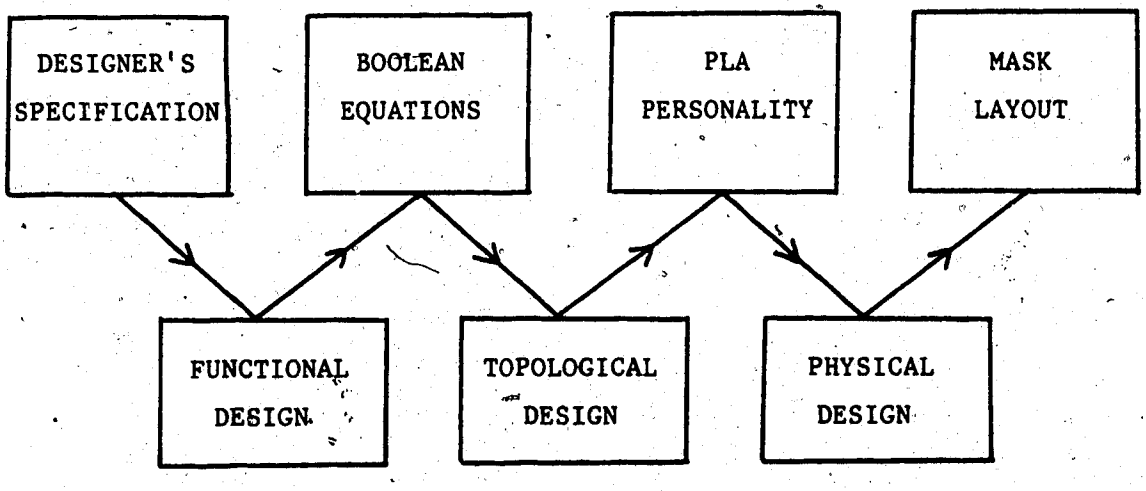


Fig. 1.3 - The PLA design process

minimize the number of product terms.

Product term minimization techniques can be classified into two categories: classical minimization and heuristic minimization. Most classical minimization algorithms are based on one procedure. This procedure first eliminates as many literals as possible from each product term by recursively applying the theorem XY + XY' = X. The resulting terms are called *prime implicants*. Next, the *minimum cover* of these prime implicants is found. The minimim cover is defined as a set of prime implicants which when ORed together is equal to the function being simplified and which contain the minimum number of literals. Detailed description of these methods may be found in [Roth 78].

Finding the minimum number of product terms has been shown to be an NP-complete problem [Gare 79, Mart 83]. That is, the running time of the algorithm above grows exponentially with the number of product terms. Consequently, for a large set of equations with many inputs and outputs, we must rely on heuristic algorithms which give near-optimum solutions in a reasonable amount of time.

Existing heuristic product term reduction algorithms can be classified into two categories: iterative methods and one-pass search methods. Iterative methods seek final minimal product term solutions by iterative improvement, without generating all prime implicants. The product term reduction methods MINI [Hong 74] and PRESTO [Svob 75] are based on this approach. These methods iteratively apply

three processes to the given list of product terms: replace each product term by the smallest product covering its unique part; rewrite pairs of product terms into new forms; find a a prime implicant covering a product term and delete all product terms covered by the prime implicant. The other class of product term reduction methods sequentially approach the final solution through a heuristically guided one-pass search. The product term reduction methods ESPRESSO [Bray 82] and PRONTO [Mart 83], are based on this approach. ESPRESSO, which is an indirect search method, can be summarized as a three step procedure: find a "best" partition tree for the given set logic equations; find a minimal cover for each of these leaves; merge the leaves. PRONTO is a direct search method which finds the minimal product term solution by expanding the given product terms in directions which can possibly cover some other uncovered product terms.

In addition to reducing the product terms, the area of a PLA can be further reduced by preprocessing the input variables. Simple logic operations (AND, OR, NAND, NOR, XOR, EXNOR, etc.) among input variables are sometimes effective. For example, *decoded* PLAs introduced by IBM are shown to have greater compactness than the PLAs discussed so far [Flei 79]. In a decoded PLA, input variables are partitioned into groups and input variables in each group are preprocessed before being connected to a PLA. Since each group need not contain the same number of variables and also

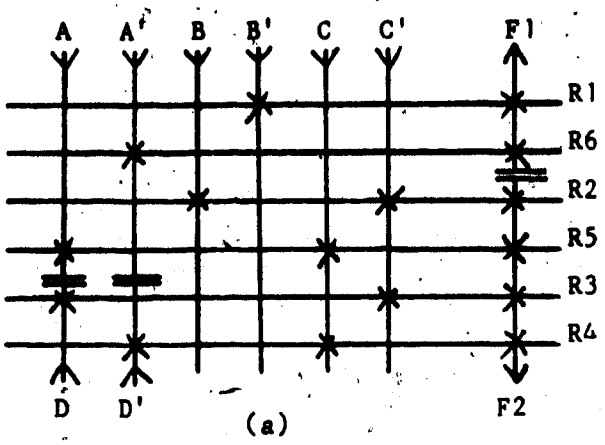
different groups may share some input variables, there are a large number of possible partitions of given input variables. Optimum input partitioning is a complex problem. Hence, the input variables are usually partitioned into disjoint groups of two variables each. For a group that has A and B as its members, the terms A' + B', A' + B, A + B' and A + B are generated outside the PLA and fed into the AND array as input signals. A simpler PLA may also be obtained by introducing additional inverters on output lines, since any function f and its complement f' differ in size.

Even with full exploitation of logic minimization at the functional level, the densities of most PLAs remain very low. Hence, the current research has been directed toward topological optimization methods. *Toplogical optimization* is the process of rearranging the positions and directions of input, output and product term lines in the original PLA, without affecting the logic function being performed, in order to reduce the silicon area occupied by the PLA. A number of toplogical optimization methods have been suggested in the literature [Gree 76, Pail 81]. The most promising method among them is PLA folding.

The concept of PLA folding was originally introduced by Wood [Wood 79] and has since been the topic of numerous research papers. Folding is a technique which aims at the reduction of the area occupied by a given PLA by exploiting its low density. There are three types of folding: column folding, row folding and mixed folding.

Column folding can be applied to the input and the output signal lines. Column folding consists of dividing an input (output) column into two parts so that two inputs (outputs) can share the same physical column. The inputs and outputs which share the same physical column must occupy the two parts of the column. One input (output) is run from the top of the PLA and the other is run from the bottom. The PLA in Figure 1.1 after column folding is shown in Figure 1.4(a). In this Figure, the first, second and seventh columns are divided into two parts by the "breaks" in the columns. The inputs A and D are fed from the top and bottom respectively of the column 1. The inputs A' and D' are fed from the top and bottom respectively of the column 2. The outputs F1 and F2 are available at the top and bottom respectively of the column 7. Note that the rows of the PLA in Figure 1.1 have been reordered in Figure 1.4(a).

Row folding consists of dividing a row of the PLA into two parts so that two product terms can share the same physical row. If all the "cuts" or "breaks" of the rows occur in the AND plane, then one product term on a folded row will be available only on the left hand side of the PLA and the other product term will be available only on the right hand side. Therefore, the OR array is divided into the left hand OR array and the right hand OR array, resulting in a PLA with an OR-AND-OR structure. The PLA in Figure 1.1 after row folding into an OR-AND-OR structure is shown in Figure 1.5(a). In this Figure, the first and the second rows

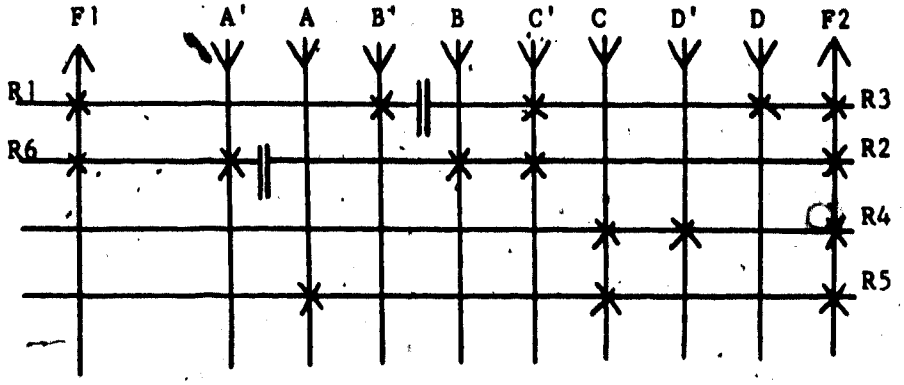


-	1	-	1
1	-	-	4
-	0	1	1
5	-	0	1
0	-	1	1
1	-	0	1

(b)

Fig. 1.4 - (a). PLA in Figure 1.1 after column folding (b). Folded PLA personality

have been divided into two parts by the breaks in the rows. The product terms R1 and R3 of the PLA in Figure 1.1 are formed at the leftside and rightside respectively of the row 1. The product terms R6 and R2 are formed at the leftside and rightside respectively of the row 2. Note that the output F1 is available on the left of the AND array (left OR array) and the output F2 is available on the right hand side of the AND array (right OR array). If all the "cuts" occur in the OR array, then one product term on a folded row will be formed in the left hand side of the PLA and the other product term will be formed on the right hand side. Therefore, the AND array is divided into left hand AND array and right hand AND array resulting in a PLA with an



(a)

1	- 2 1 0	1
1	2 0 1 -	1
0	- - 0 1	1
0	0 - 0 -	1

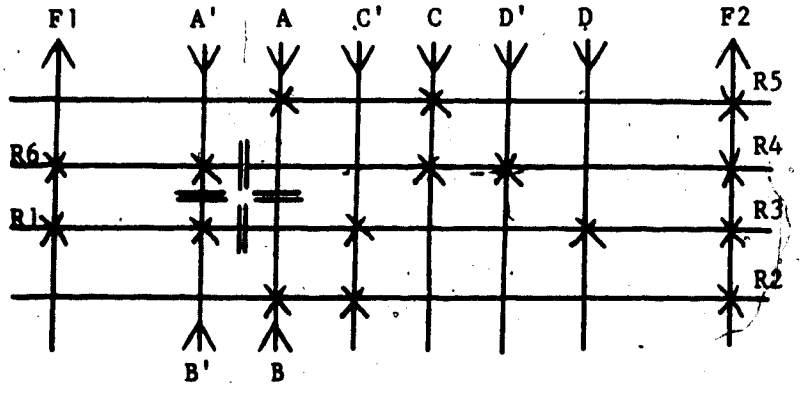
(b)

Fig. 1.5 - (a). PLA in Figure 1.1 after row folding into an OR-AND-OR structure (b). Folded PLA personality AND-OR-AND structure.

Mixed folding is the process of performing both row and column folding on a given PLA, either simultaneously or sequentially. Simultaneous mixed folding allows changeovers from column to row folding, or vice versa, after each selection of a column or row folding pair. In sequential mixed folding, all the column (row) folding pairs are found first and then all the row (column) folding pairs of the column (row) folded PLA are found. Row folding induces constraints on the relative placements of the columns and column folding induces constraints on the relative placements of the rows. Hence, mixed folding has more constraints to satisfy than just row or column folding

alone. The area savings obtained by mixed folding will depend on the sequence of folding applied. Figure 1.6(a) shows the PLA in Figure 1.1 after mixed folding.

In this thesis, we will consider column folding, row folding with OR-AND-OR structure and sequential mixed folding. The way in which a PLA is programmed and folded can be described in symbolic form by a matrix called the *folded PLA personality matrix*. As in the case of PLA personality matrix, the symbols '1', '0' and '-' in a folded PLA personality indicate information necessary to program a PLA. The other symbols indicate the locations of breaks in a folded PLA. A '2' indicates a break in the product term line following a '1' and a '3' indicates a break in the product



(a)

0	0	0	-	1
1	6	0	1	1
1	2	1	0	1
0	0	1	-	1

(b)

Fig. 1.6 - (a). PLA in Figure 1.1 after mixed folding (b). Folded PLA personality

term line following a '0'. These two symbols may be present in row folded PLA personality. A '4' indicates a break in the column following a '1' and a '5' indicates a break in the column following a '0'. These two symbols may be present in a column folded PLA personality matrix. A '6' indicates breaks in the product term line and in the the column following a '1' and a '7' indicates breaks in the product term line and in the column following a '0'. Any of these symbols ('2' to '7') may be present in a PLA personality matrix after mixed folding. The folded personalities of PLAs in Figures 1.4(a), 1.5(a), and 1.6(a) are illustrated in Figures 1.4(b), 1.5(b) and 1.6(b) respectively.

Folding a PLA can potentially reduce its area by as much as 50% if all the columns or all the rows are folded, and by 75% if all the columns and all the rows are folded. In practice, these lower bounds are rarely achieved.

1.3 Thesis Objective and Outline

The major objective of this thesis is to develop and implement a new folding algorithm and to incorporate it into a PLA design system. The minor objective of this thesis is to develop additional programs which, when added to the existing software, will make a complete PLA design system. To this end, a program to make automatic state assignments for finite state machines and a program to generate dynamic CMOS PLAs have been developed.

At the outset of Chapter 2, we explain the operational details of PLAs. A dynamic CMOS PLA generator is described along with the PLA design system at the University of Alberta. Circuit simulation results of the dynamic CMOS PLAs are also presented.

In Chapter 3, we consider PLA folding algorithms. A mathematical formulation of the folding problem is derived. A graph theoretic interpretation of the PLA folding problem and a heuristic folding algorithm by Hachtel, *et al.* are presented.

The drawbacks of the Hachtel's PLA folding algorithm and a new algorithm which overcomes them are discussed in the fourth chapter. The implementation of our algorithm is presented and compared with the results of Hachtel's algorithm. We conclude the chapter with suggestions to further improve the results of PLA folding algorithms and to expand the PLA design system. Chapter 5, summarizes and concludes the thesis.

Chapter 2

Operation and Design of PLAs

We begin this chapter with the operational details of NMOS and CMOS PLAs. Then we describe the PLA design system at the University of Alberta, which includes a dynamic CMOS PLA generator and a program to make quick state assignments for finite state machines. Circuit simulation results which guided our decision regarding the various transistors in the dynamic CMOS PLAs are presented. At the end of this Chapter we discuss the time performance of dynamic CMOS PLAs.

2.1 PLA Operation

PLAs can be implemented both in bipolar and MOS technology. In MOS technology PLAs are typically implemented with NOR-NOR logic. To understand, the logic function of PLAs, let us consider an NMOS PLA as an example.

The stick diagram of an NMOS PLA implementing the logic functions $Z_1 = AB + A'B'$ and $Z_2 = CD + C'D'$ is shown in Figure 2.1. The corresponding transistor schematic diagram is shown in Figure 2.2. In Figure 2.1, the blue lines are metal runs, the red lines are polysilicon runs and the green lines are diffusion runs. The black points indicate contacts. Each input buffer drives two lines running vertically through the AND array, one for an input term and one for its complement. The outputs of the AND array are

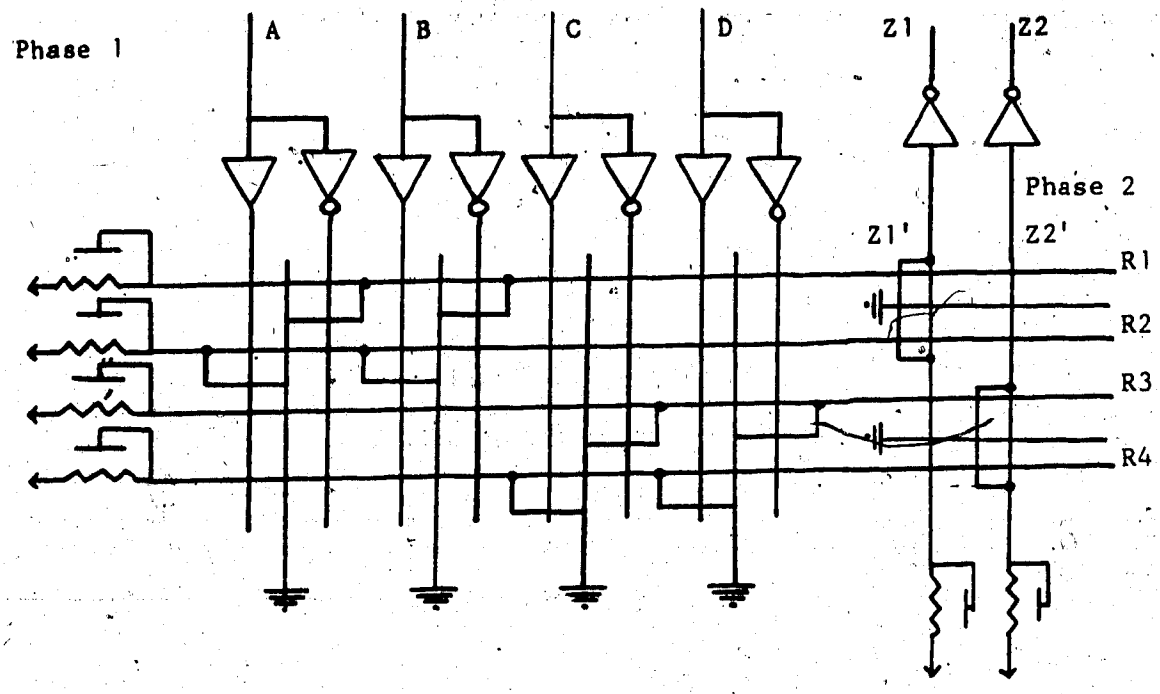


Fig. 2.1 - Stick diagram of an NMOS PLA

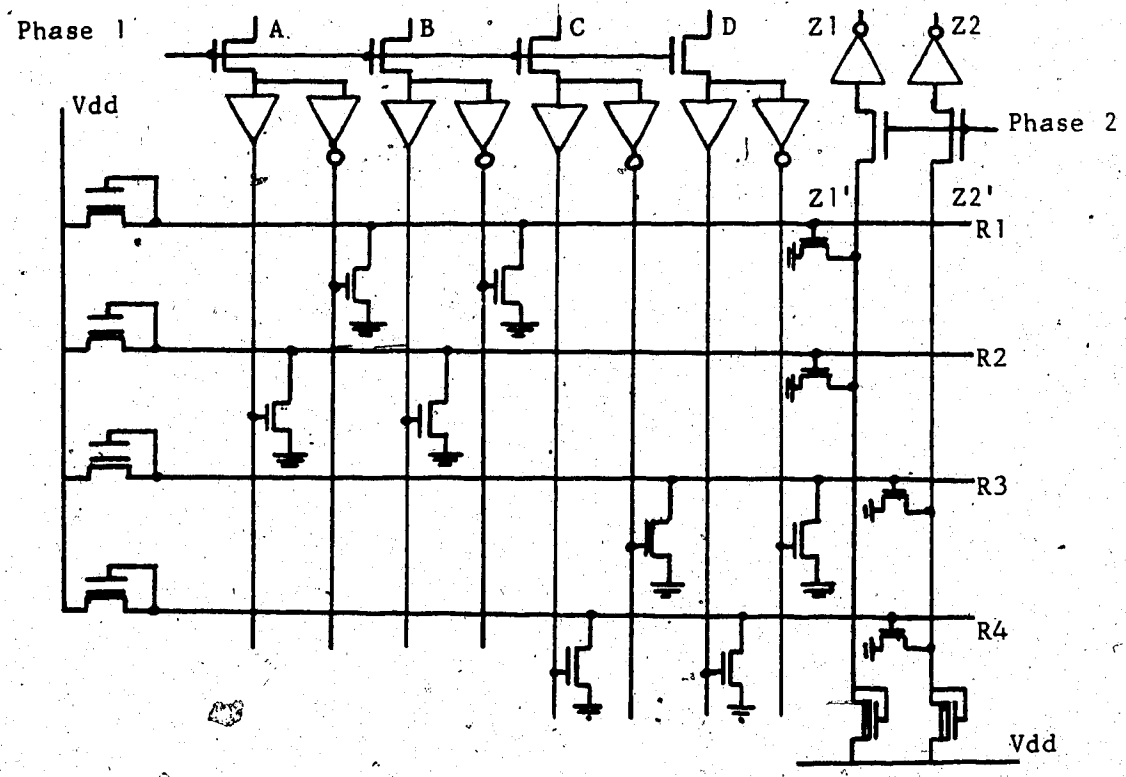


Fig. 2.2 - Transistor schematic diagram of an NMOS PLA

formed by horizontal lines with pull-up transistors at the left-most end. The product terms formed in the AND array are determined by the locations and gate connections of the pull-down transistors connecting the horizontal lines to ground. Each output running horizontally from the AND array carries the NOR combination of all input signals connected to transistors on that output. For example, the horizontal row labelled R_1 in Figure 2.2 has two transistors attached to it in the AND array, one controlled by A' and one by B'. If either of these inputs is *high*, then R_1 will be pulled toward ground and will be at *low* logic level. Thus, $R_1 = (A' + B')' = AB$. Similarly, in the OR array, each output is the NOR of the signals connected to transistors on that output. In Figure 2.2, both R_1 and R_2 lead to the gates of transistors leading from the output line Z_1'. If either the R_1 or R_2 is high, Z_1' will be low. Thus, $Z_1' = NOR(R_1,R_2) = (AB + A'B')'$. Up to this point, the PLA implements the NOR-NOR *canonical form* of Boolean functions of its inputs.

The output lines of the OR array are run through inverting drivers. At this point the output $Z_1 = AB + A'B'$. This expression illustrates why the two PLA arrays, each implementing the NOR functions, are usually referred to as the AND array and the OR array. Following the output inverters, the outputs appear directly as the sum-of-the-products canonical form of Boolean functions of the PLA inputs, that is, the OR of the AND term.

We have chosen CMOS fabrication technology to implement PLAs since it is the only fabrication technology available to us through Northern Telecom. The advantages of CMOS PLAs over NMOS PLAs are low power dissipation and high noise immunity. There are two types of CMOS PLAs available: static and dynamic. Due to the inherent nature of static CMOS logic, (for each N-channel transistor there must be a corresponding P-channel transistor), static CMOS PLAs require more silicon area and are slower than the dynamic CMOS PLAs. Therefore, in our implementation, we have used only the dynamic CMOS PLAs.

Dynamic CMOS PLAs have structures similar to NMOS PLAs and they also use NOR-NOR implementation of logic function. A stick diagram of a dynamic CMOS PLA implementation of the logic function described in Figure 2.2 is shown in Figure 2.3. The transistors in both the AND array and the OR array are N-type. The product term lines discharge transistors at the bottom of the AND array and the output lines discharge transistors at the left of the OR array are also N-type. The product term line precharge transistors at the right of the the OR array and the output line precharge transistors at the bottom of the OR array are P-type. Dynamic CMOS PLAs use two clock pulses as illustrated in Figure 2.4. The operation of dynamic CMOS PLAs can be summarized as follows.

When both phase-1 and phase-2 are low, the transmission gates isolate the incoming input signals from the AND array.

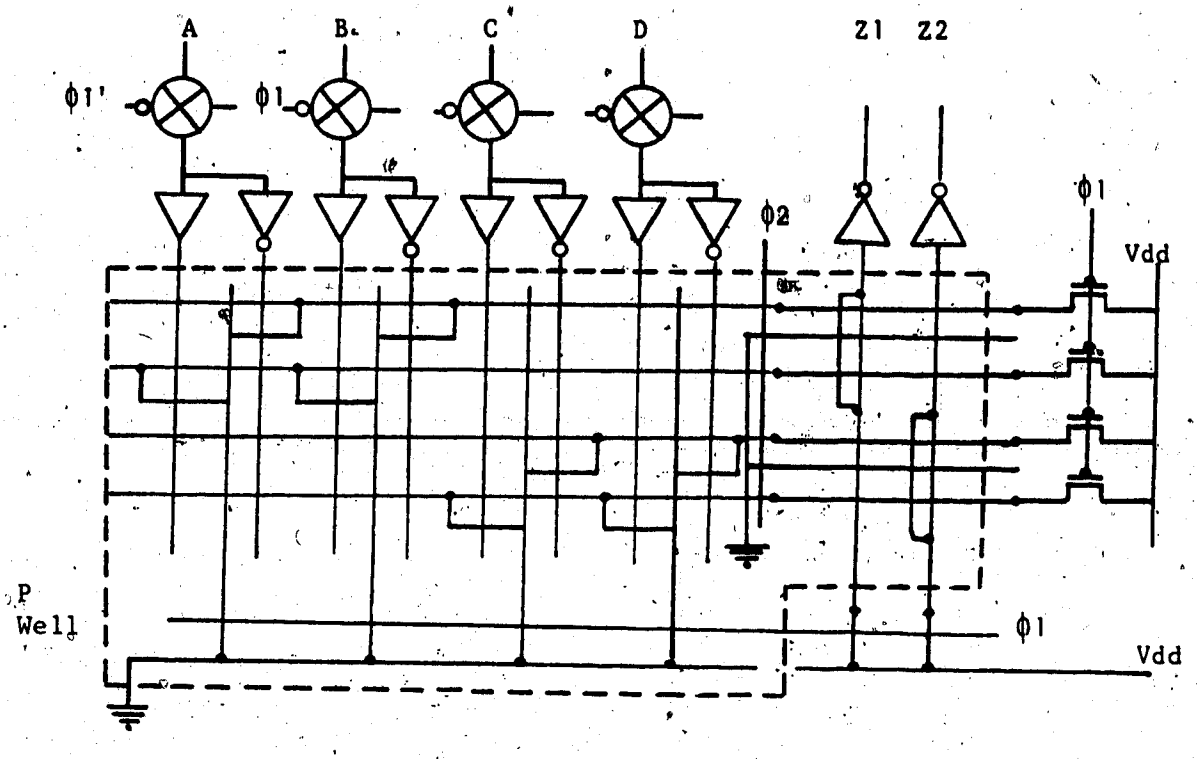


Fig. 2.3 - Stick diagram of a dynamic CMOS PLA

The N-type discharging transistors at the bottom of the AND array and at the left of the OR array are off, disconnecting the diffusion lines from the ground. The P-type precharging transistors at the right and at the bottom of the OR array are on. Therefore, the product term lines are precharged to high logic levels and the output lines are precharged to low logic levels.

When phase-1 is high and phase-2 is low, the transmission gates connect the incoming input signals to the AND array input polysilicon lines. Now, the P-type precharging transistors at the right and at the bottom of

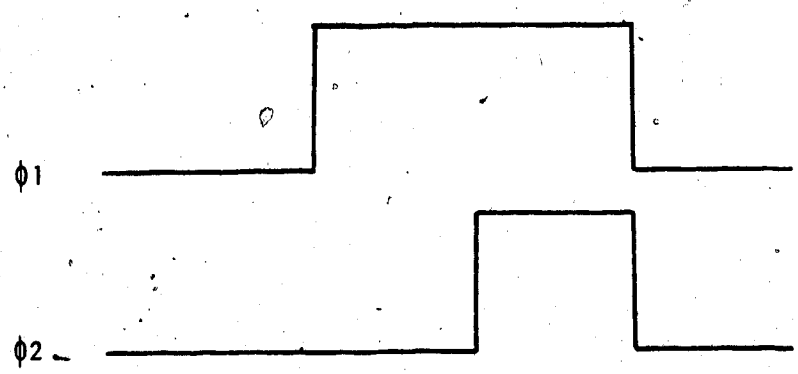


Fig. 2.4 - Clock pulses for dynamic CMOS PLAs

the OR array are off. The N-type discharging transistors at the left of the OR array are also off and the N-type discharging transistors at the bottom of the AND array are on, thus connecting the AND array diffusion lines to ground. During this time, the input signals which are at high logic levels will turn on the N-type transistors in the AND array to whose gates they are connected. Therefore, if any of the input signals connected to a product term is high, then that product term line will be discharged to low logic level.

When phase-1 and phase-2 are high, the N-type discharging transistors at the left of the OR array are also on. During this time, the product term lines still at high logic levels will keep the N-type transistors in the OR array on to whose gates they are connected. Therefore, if any of the product term lines connected to an output line is

at a high logic level, then that output line will be discharged through the N-type transistor at the left of the OR array. In other words, the logic level on the output line above the inverter will be high.

2.2 PLA Design System

Figure 2.5 is a block diagram of the PLA design system at the University of Alberta. The system takes the designer from equation specification to mask layout. The program PARSE-PLA is a an equation translator developed at the University of Waterloo. PARSE-PLA casts the functional descriptions of the PLAs into PLA personalities. Inputs to the PARSE-PLA are of two types: Combinational networks and finite state machines. A combinational network is described as a set Boolean equations. Use of temporary variables to define complex functions is also permitted. An input description for a 4-bit binary to gray code conversion is shown in Figure 2.6(a). The output from PARSE-PLA is shown in Figure 2.6(b). Input descriptions of finite state machines are given as *microcodes*. That is, an order for the states is fixed, and the default transition is assumed to be from one state to the next state in order. This is the microcode view point, where each state may be thought of as a line of microcode; each line may test for certain conditions and jump if they are met. If no jump is called for, then the next state in the selected ordering is executed. A traffic light controller described in microcodes

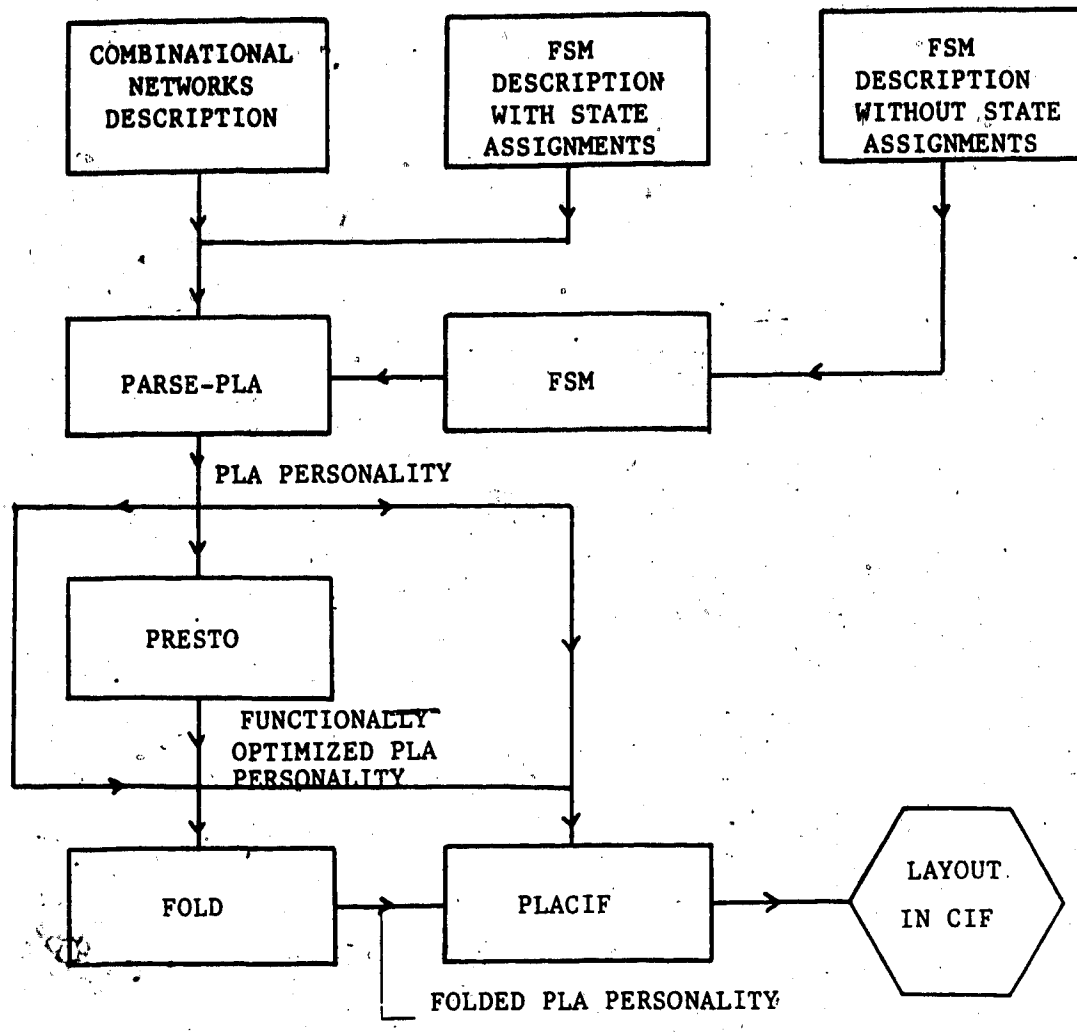


Fig. 2.5 - A PLA design system

is illustrated in figure 2.7.

Note that the assignments of binary codes to the different states must be made before using PARSE-PLA. These state assignments should be made so as to reduce the size of the PLA and to ensure correct circuit operation free of race or hazard problems. For finite state machines with a small number of states, one could try all possible assignments and choose the one which yields the "best" results. This is

```
PLA 4-bit-binary-to-gray;                          .i4

        INPUTS b0,b1,b2,b3;                        .i4
        OUTPUTS g0,g1,g2,g3;                       .p7

    b0-xor-b1 = b0 * b1' + b0' * b1;               -10- 0100
    b1-xor-b2 = b1 * b2' + b1' * b2;               01-- 1000
    b2-xor-b3 = b2 * b3' + b2' * b3;               -01- 0100
    g0 = b0-xor-b1;                                --10 0010
    g1 = b1-xor-b2;                                --01 0010
    g2 = b2-xor-b3;                                ---1 0001
    g3 = b3;                                       10-- 1000

END.                                               .e
```

Fig. 2.6 - (a). An input file to PARSE-PLA (b). Output from PARSE-PLA

known as *manual assignment*. Even for problems of moderate size, however, the number of possible assignments is too large and hence computer aids must be used to make *automatic state assignment*. The optimum state assignment is the one which produces the "lowest cost" realization of the finite state machines, *i.e.* finite state machines with a minimum number of transistors. The optimum state assignment problem is NP-complete [Giov 85]. A number of heuristic optimal state assignment methods have been reported in the literature [Davi 67, Curt 69, Stor 72 and Giov 85]. Even though these methods produce "good" assignments, they usually resort to very complex procedures. Experiments have shown that state assignments do not affect the cost of PLA implementation of finite state machines to a great extent

[Kang 81]. Therefore, we have developed a simpler state assignment algorithm illustrated in Figure 2.8.

This algorithm generates unique, unit-distant Gray codes with a starting state 0. For finite state machines with 2^n ($n > 1$) states, the first and the last codes will also be at unit-distance. Even though, this algorithm does not produce a good state assignment from a classical switching theory point of view, it is better than random state assignment in most cases. The user may wish to use these codes as a starting point for his search for better state assignments.

This algorithm is implemented in the program FSM which appears in Figure 2.5. When designing finite state machines as PLAs, the program FSM takes an input file similar to the one described in Figure 2.7 without the descriptions of state assignments, makes the state assignments and produces the corresponding input file to PARSE-PLA.

PRESTO is the heuristic logic minimization program, written by Svoboda [Svob 75]. It was translated for use on VAX at the University of California, Berkeley. PRESTO takes PLA personalities from PARSE-PLA and produces functionally optimized PLA personalities.

FOLD is the PLA folding program which will be described in detail in Chapter 3 and Chapter 4. It is capable of performing input column folding, output column folding and row folding in any desired sequence. It accepts the PLA personalities from PARSE-PLA or PRESTO and produces folded

```
FSM traffic;

    INPUTS Cars, timeout-short, timeout-long, Start-up;
    OUTPUTS Start-timer, HL0, HL1, FL0, FL1;

    STATES  Highway-green = %00,
            Highway-yellow = %01,
            Farmroad-green = %10,
            Farmroad-yellow = %11;

    RESET Start-up : Highway-green;

    STATE Highway-green > FL0;
        Cars * timeout-long : Highway-yellow > Start-timer;
        OTHERWISE : Highway-green;

    STATE Highway-yellow > HL1, FL0;
        timeout-short : Farmroad-green > Start-timer;
        OTHERWISE : Highway-yellow;

    STATE Farmroad-green > HL0;
        Cars' + timeout-long : Farmroad-yellow >
        Start-timer;
        OTHERWISE : Farmroad-green;

    STATE Farmroad-yellow > HL0, FL1;
        timeout-short : Highway-green > Start-timer;
        OTHERWISE : Farmroad-yellow;

END.
```

Fig. 2.7 - Microcode description of traffic light controller

PLA personalities.

PLACIF accepts PLA personalities as input and produces the mask layout for the corresponding dynamic CMOS PLAs in CIF. This is done by selecting and replicating the components in a PLA cell library. The PLA cell library consists of a transmission gate, buffer, precharging and discharging transistors and a basic cell of dimension

```
Procedure stateassign;
var int S, L, i ,j;
    record
        {   int digit;
            int binary[];
            int gray[];
        }   CODES[];
begin
    Get the number of states (S);
    Calculate the state lines required (L);
    for i := 1 to S CODES[i].digit = i - 1;
    for i := 1 to S
        convert CODES[i].digit to L bit binary and assign it
        to CODES[i].binary;
    for i := 1 to S
        for j := 1 to L
            if (j < L) CODES[i].gray[j]
                = CODES[i].binary[j]  CODES[i].binary[j+1];
            else CODES[i].gray[j] = CODES[i].binary[j];

end;
```

Fig. 2.8 - An algorithm for quick state assignment

70 μm x 35 μm which is manipulated and replicated to make up the AND and OR arrays of the PLAs. The basic cell which is illustrated in Figure 2.9, corresponds to an intersection of a product term row and an input column in the AND array. When rotated 90° it also serves as the intersection between an output column and two product term rows in the OR array. This can be clearly seen in Figure 2.10 which is the plot of the dynamic CMOS PLA in Figure 2.3 produced by PLACIF.

As illustrated in Figure 2.11, the various symbols in a PLA personality matrix can be realized by adding or deleting certain artifacts from the basic cell. To program AND array, a diffusion run from the left or right vertical diffusion

line is used to connect with the metal product term line via a contact. Either the left or right polysilicon run interrupts this path forming the gate of an N-type transistor. If the left polysilicon (uninverted input line) controls the transistor a '0' is formed which provides the signal inversion necessary in the NOR-NOR representation of the CMOS PLA. Right side polysilicon (inverted input line) provides the complementary function (forms a '1'). A product line may connect to an input or its complement or it may not depend on the input at all.

To program the OR array, *i.e.* to connect a product term to an output, a vertical diffusion is used to connect the nearest horizontal diffusion line with the output metal line via a contact. This diffusion run interrupts the product line in polysilicon thus forming a P-type transistor.

To implement a fold in a row in the AND array, one of the metal strips (marked M1, M2 in Figure 2.11(a)) is

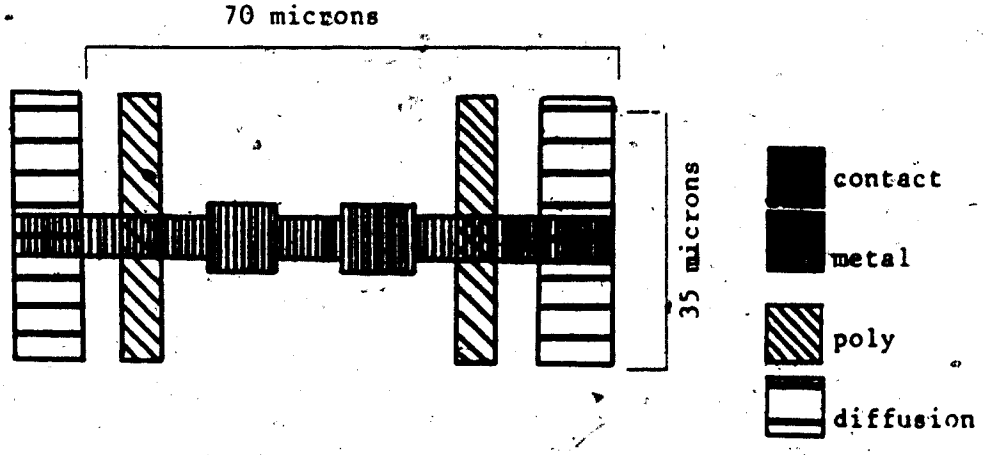


Fig. 2.9 - Basic cell of dynamic CMOS PLAs

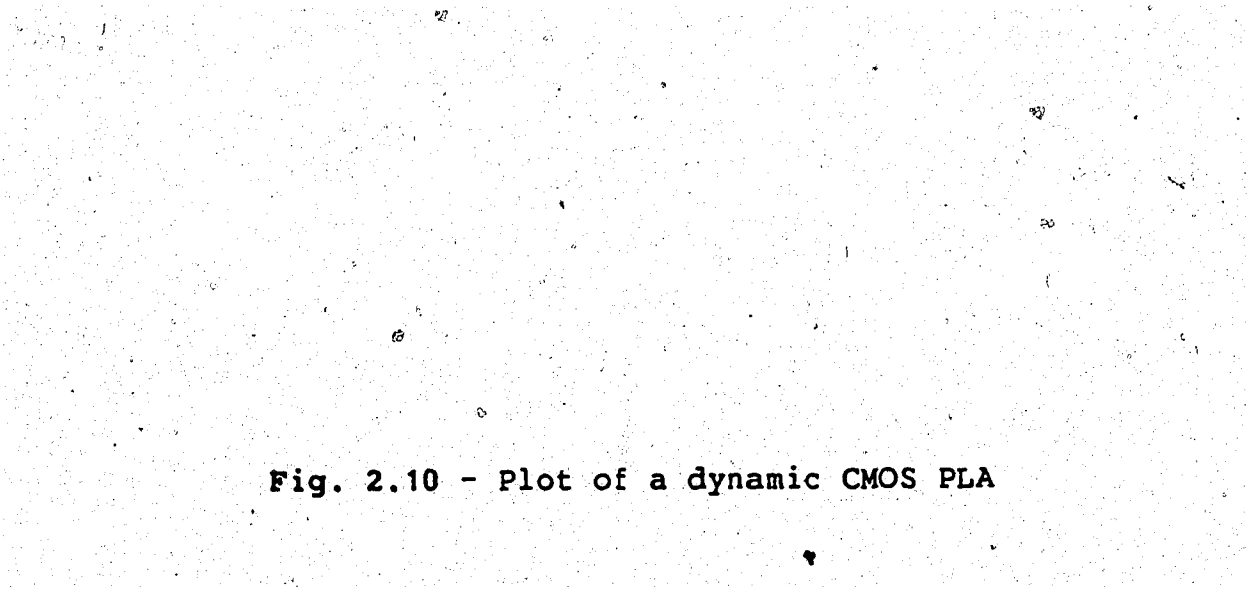
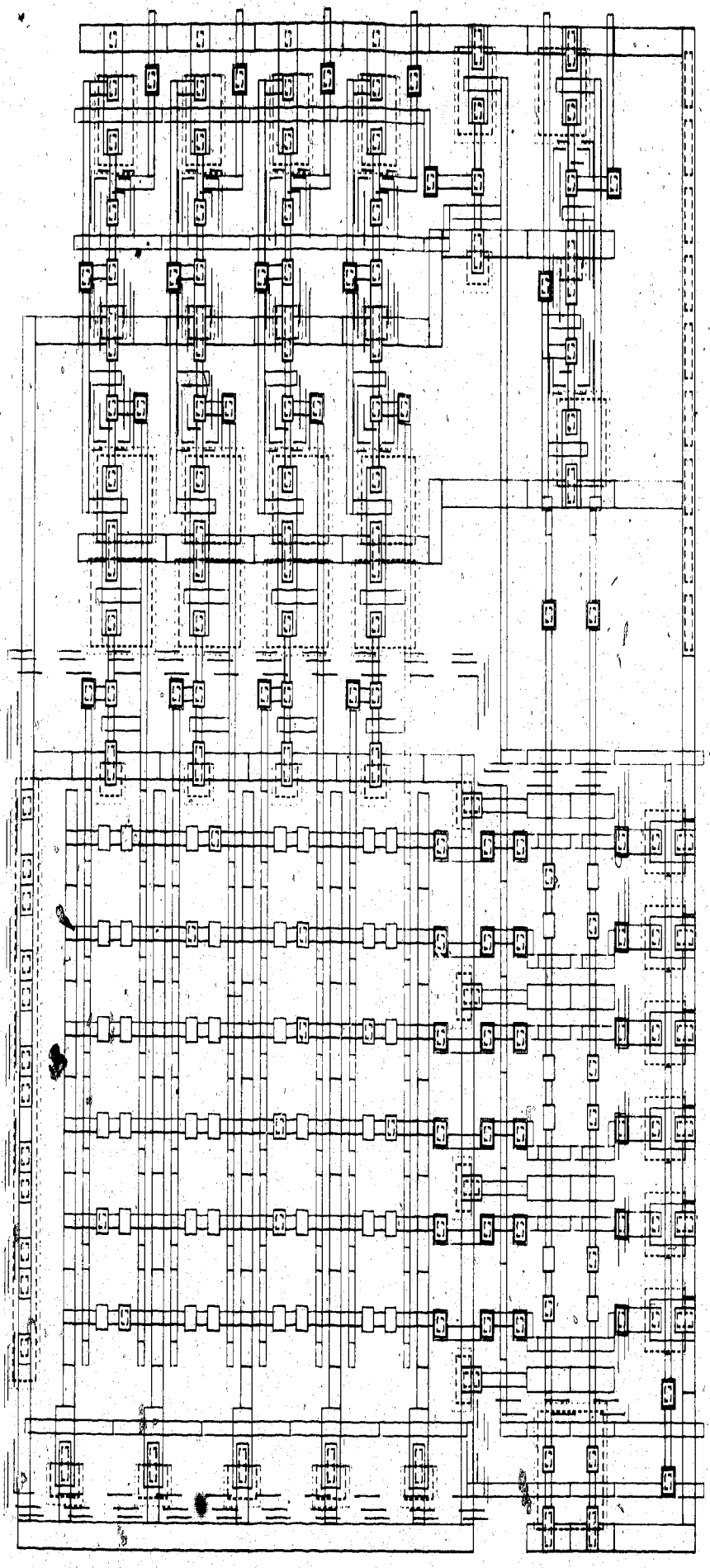
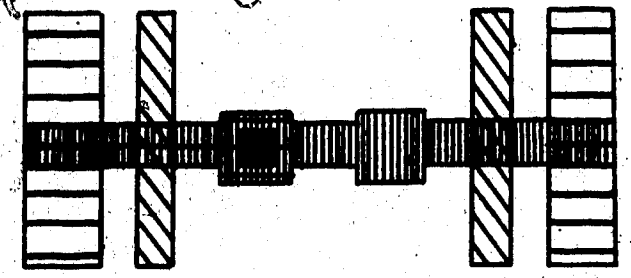
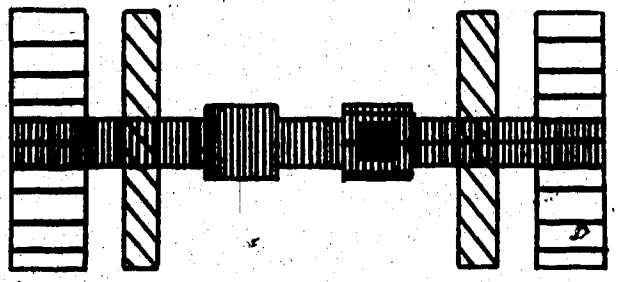


Fig. 2.10 - Plot of a dynamic CMOS PLA

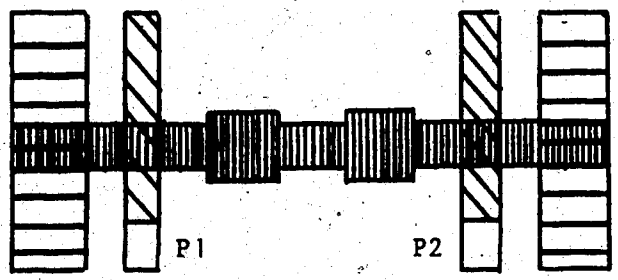




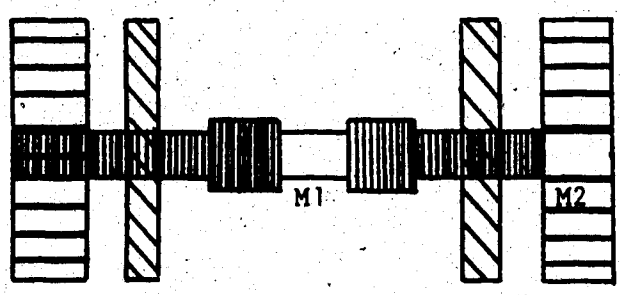
(a) Implementing:
i. a '0' in the AND Array,
ii. when rotated 90°, a '1' in the OR Array.



(b) Implementing:
i. a '1' in the AND Array,
ii. when rotated 90°, a '1' in the OR Array.



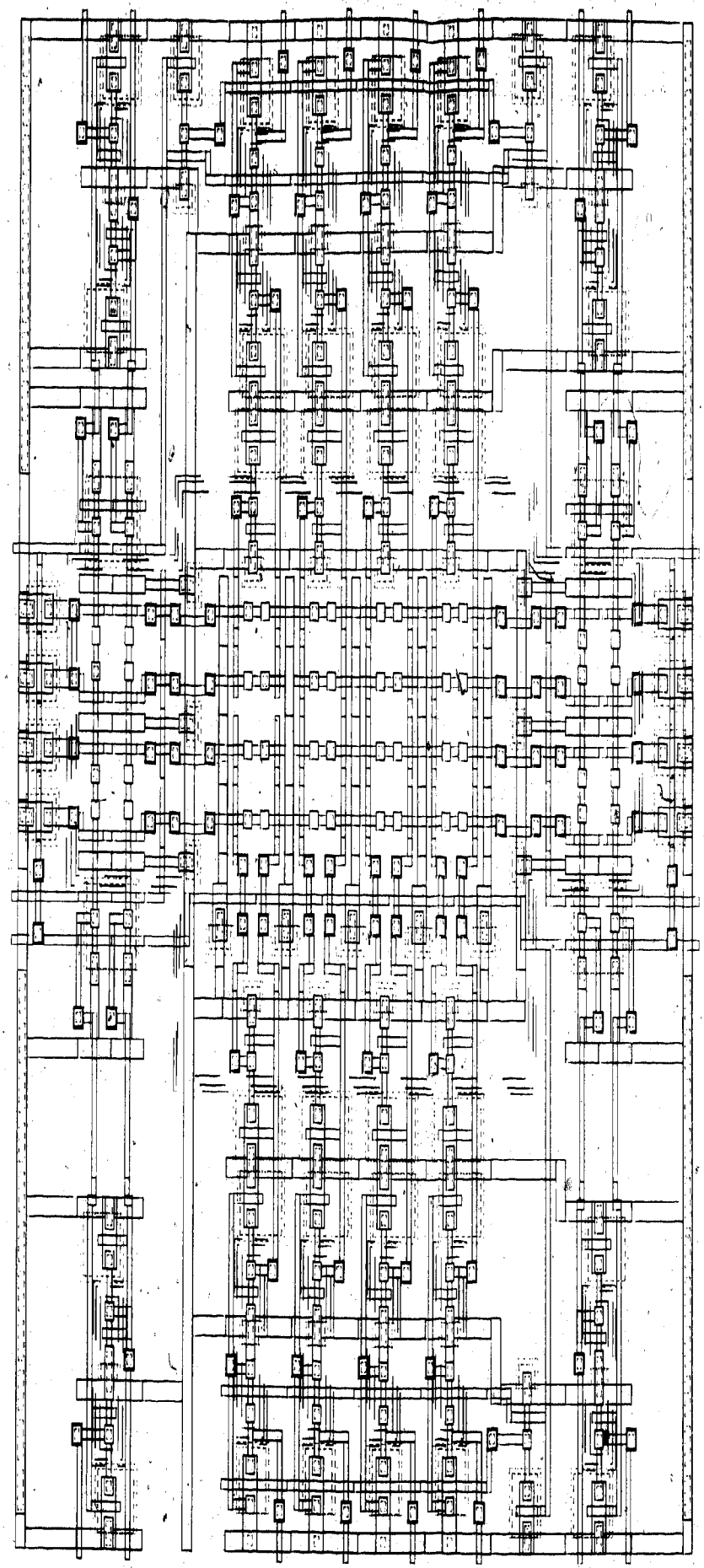
(c) P1 and P2 are removed to implement a fold in a column.



(d) Either M1 or M2 is removed to implement:
i. a fold in the row in the AND Array,
ii. a fold in the OR Array column.

Fig. 2.11 - Manipulation of the basic cell

Fig. 2.12 - Plot of a dynamic CMOS PLA after mixed folding



removed from the basic cell. To implement a fold in a column in the AND array, the bottom parts of the polysilicon lines in the basic cell (marked P1, P2 in Figure 2.11(c)) are removed. A fold in a OR array column can be implemented by removing one of the metal (M1, M2) strips from the basic cell and rotating it by 90°. Figure 2.12 is the plot of the mixed folded PLA personality shown in Figure 1.6(a).

2.3 Circuit Simulation of the Dynamic CMOS PLA

In large PLAs, the resistances and capacitances of signal lines affect the performance of the PLAs to a great extent. The values of these resistances and capacitances must be appropriately included in the circuit simulations in order to accurately evaluate the performance of the PLAs. The dimensions of the various transistors in the PLA must be chosen to compensate their parasitic effects.

In accordance with practical PLA sizes, we assumed a PLA with 25 inputs variables, 50 output terms and 50 product terms, for the simulation. The AND array and OR array transistor densities are assumed to be 50% and 20% respectively. Our approach in designing the PLAs is to use minimum dimension (5 $\mu m \times 5\ \mu m$) transistors to personalize the PLA, and determine the dimensions of the precharging, discharging and buffer transistors based on simulation results. Each signal line has been modelled as a pair of resistance and capacitance as shown in Figure 2.13. The complete PLA model is shown in Figure 2.14. In order to

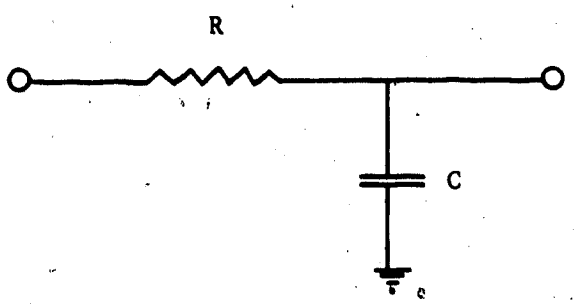


Fig. 2.13 - Transmission line model

simplify the simulation tests, the PLA has been identified as having four sub-circuits as indicated in Figure 2.13. They are:

1. Input buffer circuit,
2. Product term and precharge line,
3. AND array discharge line,
4. Output line.

The variables indicated in Figure 2.14 are the following:

1. RM_1 and CM_1 are the resistance and capacitance of a metal (product term) line in the AND array,
2. RD_1 and CD_1 are the resistance and capacitance of a diffusion (ground) line in the AND array,
3. RP_1 and CP_1 are the resistance and capacitance of a poly (input) line in the AND array,
4. RM_2 and CM_2 are the resistance and capacitance of a metal (output) line in the OR array,
5. RD_2 and CD_2 are the resistance and capacitance of a diffusion (V_{dd}) line in the OR array and

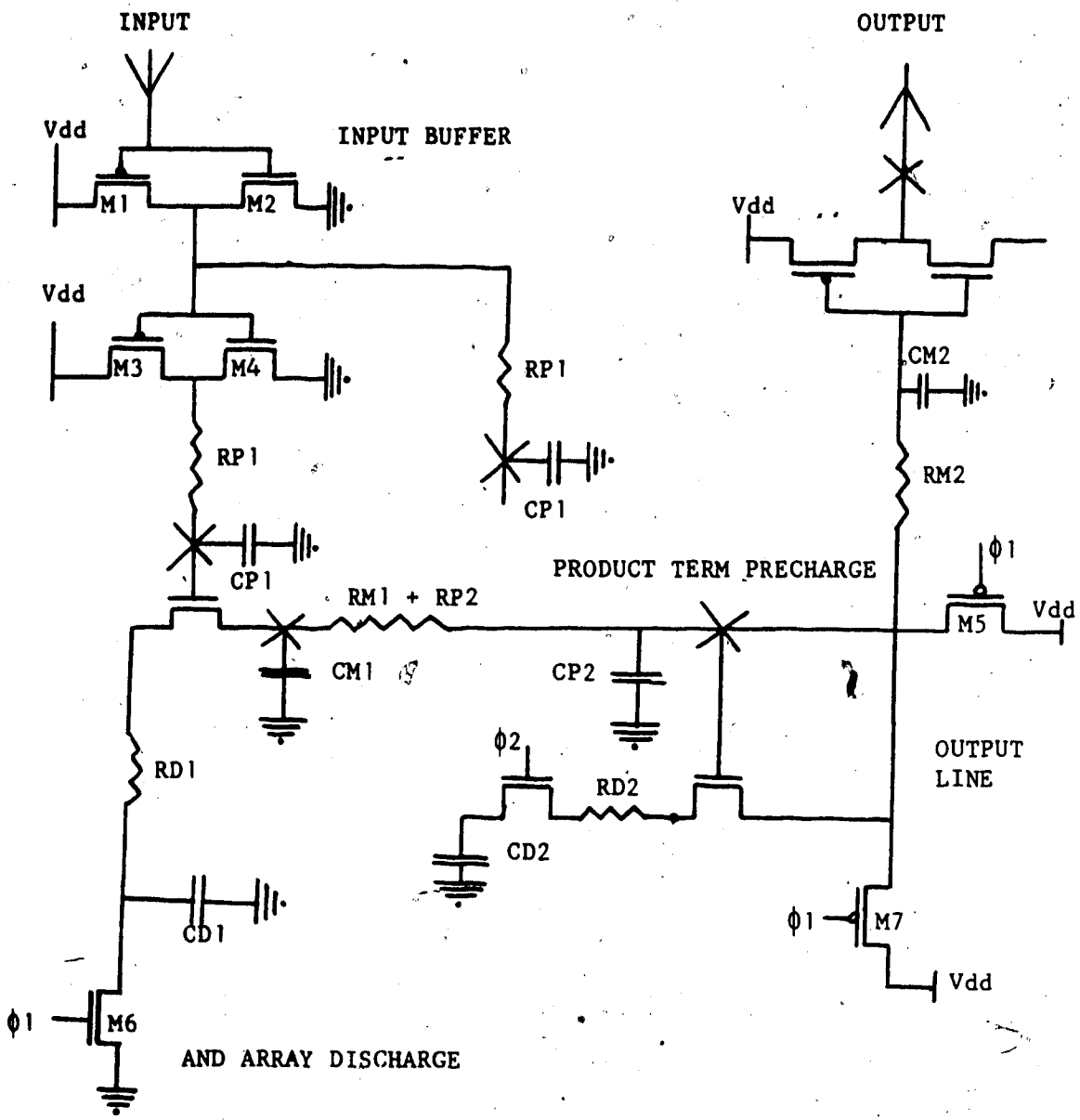


Fig. 2.14 - PLA model for simulation

6. RP_2 and CP_2 are the resistance and capacitance of a poly (product term) line in the OR array.

The following (Northern Telecom CMOS 1B process) constants have been used to calculate these resistances and capacitances.

(i) Capacitances

Diffusion	= 3.0×10^{-8} Farad/cm^2
Poly	= 3.2×10^{-8} Farad/cm^2
Metal	= 2.3×10^{-8} Farad/cm^2
Gate Capacitance	= 4.06×10^{-8} Farad/cm^2

(ii) Resistances

Diffusion (N+)	= 15 ohms per square
Poly (N+)	= 25 ohms per square
Metal	= 0.03 ohms per square

As noted earlier, the AND and OR array in the dynamic CMOS PLA are made up by replicating a simple cell of dimension 70 μm x 35 μm. We will use this dimension to calculate the lengths of the signal lines and hence the values of the resistances and capacitances in Figure 2.12. These calculations are summarized below.

AND array diffusion line:

Area for one product term

= $9 \times 35 \times 10^{-8}$ cm^2 = 315×10^{-8} cm^2

Area for 50 product terms

$= 315 \times 10^{-8} \times 50\ cm^2 = 15750 \times 10^{-8}\ cm^2$

$\therefore CD_1 = 15750 \times 10^{-8} \times 3 \times 10^{-9}\ F = 0.4725\ pF$

$RD_1 = (50 \times 35 \times 15)/9\ ohms = 2917\ ohms$

AND array metal line:

Area for one input variable

$= (52 \times 5 + 18 \times 9) \times 10^{-8}\ cm^2 = 422 \times 10^{-8}\ cm^2$

Area for 25 input variables

$= 422 \times 10^{-8} \times 25\ cm^2 = 10550 \times 10^{-8}\ cm^2$

$\therefore CM_1 = 10550 \times 10^{-8} \times 2.3 \times 10^{-9}\ F = 0.2426\ pF$

$RM_1 = 25 \times (52/5 + 18/9) \times 0.03\ ohms = 9.3\ ohms$

AND array poly line:

Area for one product term

$= 5 \times 35 \times 10^{-8}\ cm^2 = 175 \times 10^{-8}\ cm^2$

Area for 50 product terms

$= 175 \times 10^2 \times 50\ cm^2 = 8750 \times 10^{-8}\ cm^2$

Line capacitance

$= 8750 \times 10^{-8} \times 3.2 \times 10^{-9}\ F = 0.28\ pF$

With 50% density in the AND array, each input or its complement is present in about 13 (12.5 to be exact) product terms.

Minimum gate area $= 25 \times 10^{-8}\ cm^2$

Gate area for 13 transistors $= 325 \times 10^{-8}\ cm^2$

Gate capacitance

$= 325 \times 10^{-8} \times 4.06 \times 10^{-8}$ F = 0.132 pF

$\therefore CP_1 = 0.28$ pF + 0.132 pF = 0.412 pF

$RP_1 = (50 \times 35 \times 25)/5$ ohms = 8750 ohms

OR array diffusion line:

Area for one output term

$= 35 \times 9 \times 10^{-8}$ cm² $= 315 \times 10^{-8}$ cm²

Area for 50 output terms

$= 315 \times 10^{-8} \times 50$ cm² $= 15750 \times 10^{-8}$ cm²

$\therefore CD_2 = 15750 \times 10^{-8} \times 3.0 \times 10^{-8}$ F = 0.4725 pF

$RD_2 = (50 \times 35 \times 25)/9$ ohms = 4861 ohms

OR array metal line:

Area for 2 product terms

$= (52 \times 5 + 18 \times 9) \times 10^{-8}$ cm² $= 422 \times 10^{-8}$ cm²

Area for 50 product terms

$= 422 \times 10^{-8} \times 25$ cm² $= 10550 \times 10^{-8}$ cm².

$\therefore CM_2 = 10550 \times 10^{-8} \times 2.3 \times 10^{-8}$ F = 0.2427 pF

$RM_2 = 25\ (52/5 + 18/9) \times 0.03$ ohms = 9.3 ohms

OR array poly line:

Area for one output term

$= 35 \times 5 \times 10^{-8}$ cm² $= 175 \times 10^{-8}$ cm²

Area for 50 output terms

$= 175 \times 10^{-8} \times 50\ cm^2 = 8750 \times 10^{-8}\ cm^2$

Line capacitance

$= 8750 \times 10^{-8} \times 3.2 \times 10^{-9}\ F = 0.28\ pF$

With 20% density in the OR plane, for 50 output terms, each product term is present in 10 output terms.

Gate capacitance

$= 25 \times 10^{-8} \times 10 \times 4.06 \times 10^{-8}\ F = 0.1015\ pF$

$\therefore CP_2 = 0.28\ pF + 0.1015\ pF = 0.3815\ pF$

$RP_2 = (50 \times 35 \times 25)/5\ ohms = 8750\ ohms$

In the following paragraphs, the dimensions of transistors are expressed by W/L where W and L are the width and length of a transistor in microns (μm) respectively. Rise time is defined as the time taken for the voltage at a node to rise from 10% to 90% of the maximum supply voltage (V_{dd}). Fall time is defined as the time taken for the voltage at a node to fall from 90% to 10% of the maximum voltage. With the supply voltage of 5.0V, the 10% - 90% range is from 0.5V to 4.5V. The nodes where the voltages are measured locations are indicated by N, in the following Figures.

At the start of simulation tests, minimum dimensions (5/5) were used for the transistors (M1 - M8) under consideration. In places where this choice did not give faster rise or fall time, the widths of the transistors were increased in steps of 5 μm keeping the length at 5 μm.

Input buffer design:

The input buffer circuit with the resistance and capacitance values calculated is illustrated in Figure 2.15. This circuit was simulated to determine the dimensions of the transistors M1, M2, M3 and M4 in the input buffer which would give equal rise and fall times both for an input variable and its complement. The response times were measured at N_1 and N_2. Buffers with M1 = 15/5, M2 = 5/5, M3 = 15/5 and M4 = 5/5 gives the following results:

rise time of an input signal = 21.9 ns

fall time of an input signal = 23.8 ns

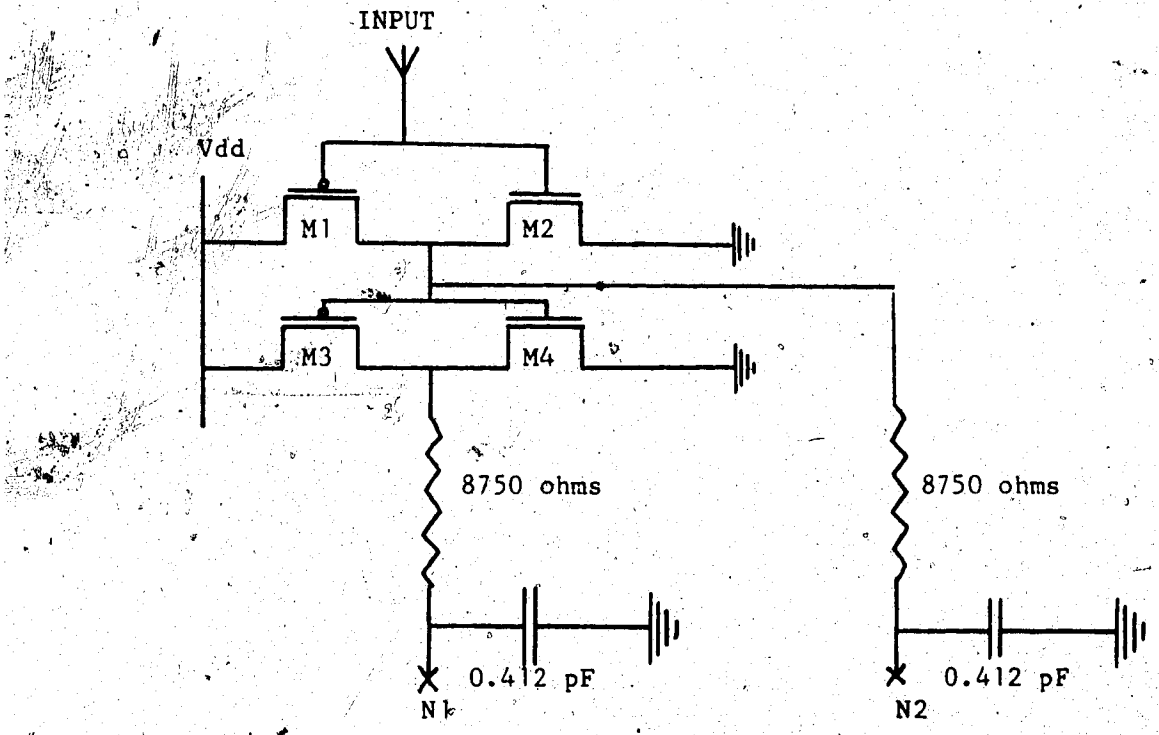


Fig. 2.15 - Input buffer circuit

rise time of the complement of an input signal = 16.6 ns

fall time of the complement of an input signal = 19.2 ns

Product term precharge transistor design:

The circuit which was simulated to determine the product term precharge transistor size is shown in Figure 2.16. The response time was measured at N_3. Based on a number of tests, we decided to use precharge transistors of dimensions M5 = 15/5 which takes 27.1 ns to precharge a product term line.

AND array discharge transistor design:

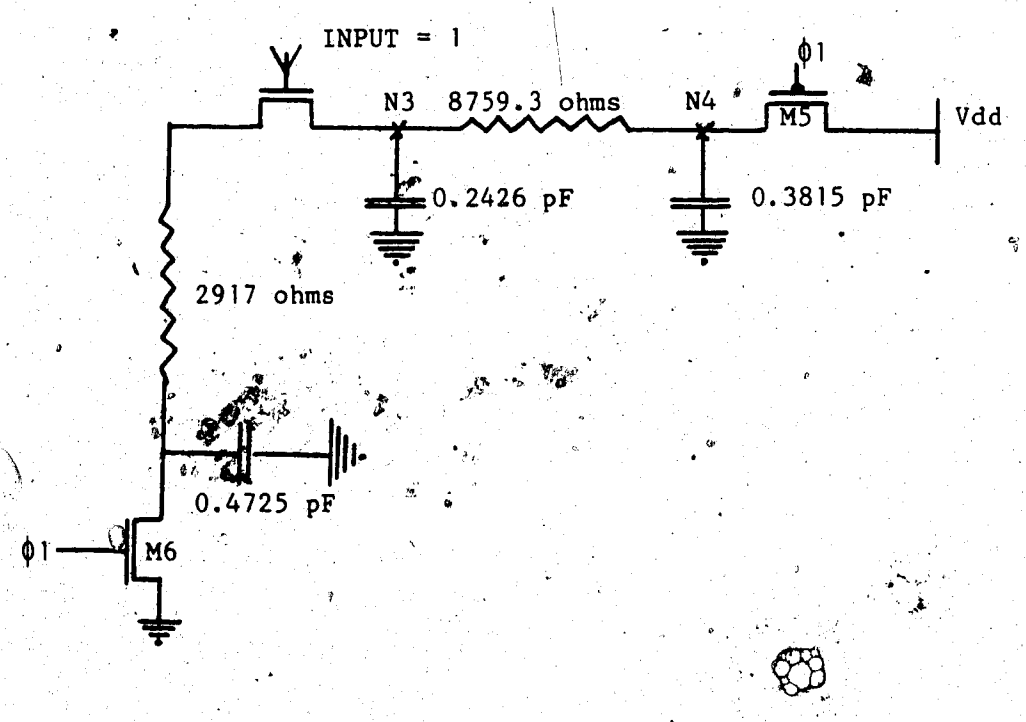


Fig. 2.16 - Product term precharge circuit

The circuit in Figure 2.16 was once again simulated with M5 = 15/5 to determine the dimension of the AND array discharge transitors (M6) which would give the fastest time to conditionally discharge product term lines. The response time was measured at N_4. M6 = 15/5 takes 32.5 ns to discharge a product term line.

Output line discharge transistor design:

The circuit used for this simulation test is depicted in Figure 2.17. With a minimum dimension M7, it took 18.5 ns to precharge an output line. With a minimum dimension M8, it took 14.2 ns to discharge an output line. The response times were measured at N_5.

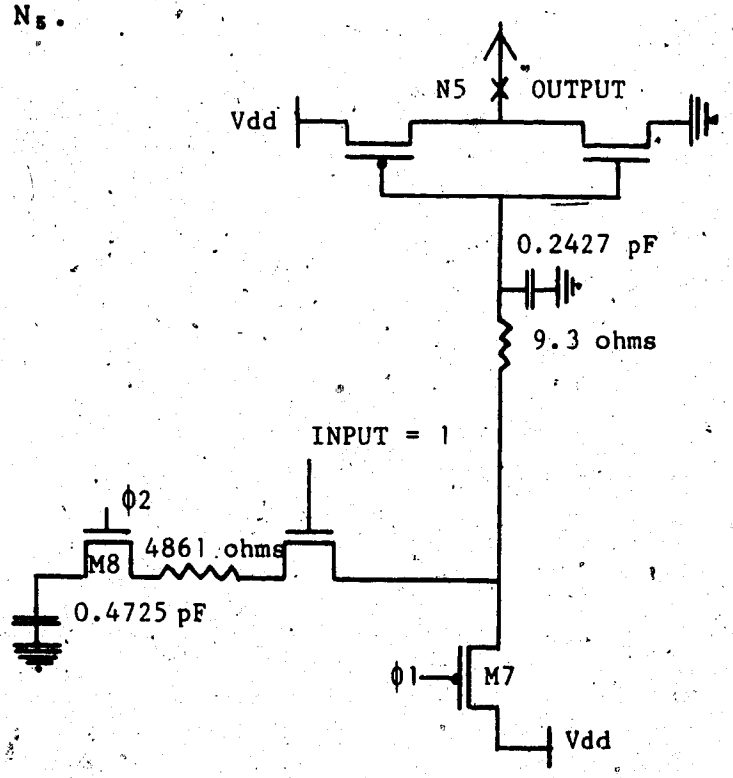


Fig. 2.17 - Output line precharge and discharge transistors design

Timing Analysis:

In the worst case, the time taken by a dynamic PLA of the dimensions described above to respond to a change in input signals is given by $T = T_1 + T_2 + T_3$, where:

T_1 = MAX
{ time taken to precharge a product term line,
time taken to precharge an output line
},

T_2 = time taken by the input buffers to charge up the input poly lines and the time taken to conditionally discharge a product term line,

T_3 = time taken to discharge an output line.

With the transitor dimensions we have chosen,

T_1 = MAX {27.1 ns, 18.5 ns}
= 27.1 ns

T_2 = 23.8 + 32.5 = 56.8 ns

T_3 = 14.2 ns

$\therefore T$ = 98.1 ns.

Chapter 3

PLA Folding

The objective of a PLA folding algorithm is to find a maximal set of column (row) pairs in a PLA where each pair can be implemented in the same physical column (row) with all the technological and design constraints satisfied. In order to mathematically formulate the PLA folding problem, we must consider the constraints that PLA folding in its various forms must satisfy. To accomplish this, let us consider column folding of a PLA in detail.

An example PLA is shown in Figure 3.1. An input or output variable c_i is known as a "primary variable". The set C of primary variables c_i is the union of set X of input

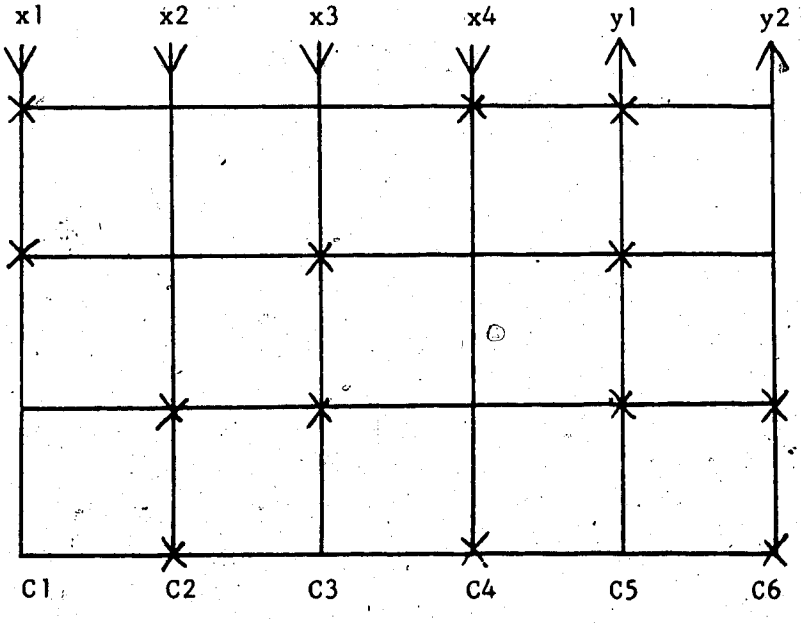


Fig. 3.1 - An example PLA

variables x_i and the set Y of output variables y_i. R is the set of the product term rows r_i. Associated with each primary variable c_i is a set $R(c_i) \subseteq R$. $R(c_i)$ contains all product term rows r_k which depend on x_i or which build up y_i. In other words, element (r_k, c_i) in the PLA personality is a '1' or a '0'. Similarly, associated with each row r_i is a set $C(r_i)$, the union of the set of input variables present in the product term row r_i and the set of output variables in which the product term r_i is present.

Two columns c_i and c_j are *disjoint* if $R(c_i) \cap R(c_j) = \phi$. More specifically, two input columns are disjoint if the logic variable they represent does not occur in the same product term for any of the output functions and two output columns are disjoint if the logic function they represent does not have common product terms. A *column folding pair* is defined as an unordered pair $f = (c_i, c_j)$, where c_i and c_j are disjoint columns. The *column folding set* $F = \{f_1, \ldots, f_k\}$, is a set of all possible column folding pairs in a PLA.

An *ordered column folding pair* is as an ordered pair $o = (c_i, c_j)$, in which the signal line corresponding to c_i is assigned to be on top and the signal line corresponding to c_j is assigned to the bottom of a folded column in the folded PLA. An *ordered column folding set* $O = \{o_1, \ldots, o_k\}$, is a set of ordered column folding pairs.

In order to implement an ordered column folding pair (c_i, c_j) in the same column with c_i at the top and c_j at the

bottom, without affecting the logic being performed by the PLA, all the rows in $R(c_i)$ must be above all rows in $R(c_j)$. In other words, each ordered column folding pair $o_1 = (c_i, c_j)$, induces a partial order (irreflexive and asymmetric) relation $Q_1 = R(c_i) \times R(c_j) \subset R \times R$ on the rows of the PLA. For notational purposes, we will write this relation as $Q_1 = R(c_i) < R(c_j)$ which indicates all the rows in $R(c_i)$ are above all the rows in $R(c_j)$. We will also write each pair $(r_m, r_n) \in Q_k$ as $r_m < r_n$. Since partial ordering can be embedded into a linear ordering, the rows of the PLA can be reordered so that c_i and c_j can share one vertical column.

In order to assign a second column folding pair (c_x, c_y) to the top and bottom respectively of a column in the PLA, the rows of the PLA must be reordered once again so that $Q_2 = R(c_x) < R(c_y)$ also holds. Let us define the relation Q(R) to be the union of the relations Q_i induced by the ordered folding pairs of O. Not all ordered folding sets O for a particular PLA can be implemented because of conflicting ordering among rows. This is because Q(R) is not a partial ordering on R since it is not transitive. Therefore, we define $Q^*(R)$ as the transitive closure of Q(R). The *implementability* of an ordered column folding set O is determined by the following theorem.

Theorem 1: A set CFP of ordered column folding pairs $\{(c_{i1}, c_{i2}), i = 1...m\}$, can be assigned to m vertical lines if and only if $Q^*(R)$ is a partial order on R, the

set of the rows in the PLA, *i.e.* $Q^*(R)$ is asymmetric.

Proof:

Necessity:

$Q^*(R)$ defines the ordering of the rows of the PLA which will enable us to implement the column folding. It is physically impossible to reorder the rows of the PLA if $Q^*(R)$ is symmetric.

Sufficiency:

By definition, if $Q^*(R)$ is asymmetric then the relation Q_i induced by each ordered column folding pair o_i in CFP holds. Therefore, the rows of the PLA can be reordered so that each ordered folding pair in CFP can be implemented. ■

For the PLA shown in Figure 3.1 the set $F = \{(c_1,c_2),(c_4,c_3)\}$, is the column folding set since (c_1,c_2) and (c_4,c_3) are disjoint column pairs. The set $O = \{(c_1,c_2),(c_3,c_4)\}$, is an ordered column folding set in which we specify that c_1 has to be on top of c_2, and c_3 has to be on top of c_4 in the folded PLA. The ordered column folding pair $o_1 = (c_1,c_2)$ induces the relation $Q_1 = R(c_1) < R(c_2)$, and the ordered column folding pair $o_2 = (c_3,c_4)$ induces the relation $Q_2 = R(c_3) < R(c_4)$. Q_1 consists of $r_1 < r_3$, $r_1 < r_4$, $r_2 < r_3$ and $r_2 < r_4$. Q_2 consists of $r_2 < r_1$, $r_2 < r_4$, $r_3 < r_1$ and $r_3 < r_4$. The ordered folding set $O = \{(c_1,c_2),(c_3,c_4)\}$ of this PLA is not implementable. This is because Q_1 contains $r_1 < r_3$ and Q_2 contains $r_3 < r_1$ and therefore $Q^*(R)$ is non-asymmetric. Such non-asymmetric relations are also known as *alternating*

cycles.

We can now state the optimum column folding problem mathematically as follows:

Given the personality of a PLA, find an implementable ordered column folding set of maximum cardinality.

The constraint for row folding can be derived in a similar way. Let RFP = $\{(r_{i1}, r_{i2}), i = 1...n\}$ be the set of disjoint ordered row folding pairs in which row r_{i1} and row r_{i2} are assigned to the left and right respectively of the row i. Each ordered row folding pair (r_{i1}, r_{i2}), induces a relation $C(r_{i1}) \times C(r_{i2}) \subset C \times C$ on the columns of the PLA. We will represent this relation by $Q_i = C(r_{i1}) < C(r_{i2})$, which indicates that all the columns in $C(r_{i1})$ are on the left side of all the columns in $C(r_{i2})$. Let Q(C) be the union of the relations Q_i, induced by the ordered row folding pairs of RFP and let $Q^*(C)$ be the transitive closure of Q(C). The constraint for the implementability of an ordered row folding set RFP can be stated as follows:

Theorem 2: A set RFP of ordered row folding pairs $\{(r_{i1}, r_{i2}), i = 1....n\}$, can be assigned to n horizontal lines if the transitive closure $Q^*(C)$ of Q(C) is asymmetric.

The proof of this theorem is similar to the proof of Theorem 1. The optimum row folding problem can be stated mathematically as follows:

Given the personality of a PLA, find the implementable ordered row folding set of maximum cardinality.

Mixed folding consist of finding the maximal, implementable sets of row and column folding pairs. The constraints for the implementability of such sets is stated in the following theorem.

Theorem 3: A set CFP of ordered column folding pairs $\{(c_{i1},c_{i2}),\ i = 1....m\}$ can be assigned to m vertical lines and a set RFP of ordered row folding pairs $\{(r_{i1},r_{i2}),\ i = 1....n\}$ can be assigned to n horizontal lines if and only if:

a) $Q^*(R)$ is asymmetric,

b) $Q^*(C)$ is asymmetric,

c) $\forall\ (c_i,c_j) \in CFP$, $(c_i,c_j) \notin Q^*(C)$ and $(c_j,c_i) \notin Q^*(C)$, and

d) $\forall (r_i,r_j) \in RFP$, $(r_i,r_j) \notin Q^*(R)$ and $(r_j,r_i) \notin Q^*(R)$.

Proof:

Necessity:

The necessity of conditions a) and b) follows from Theorems 1 and 2. The necessity of condition c) can be proved by contradiction. Let (c_i,c_j) be a column folding pair in CFP. This means only one line is provided for this pair of variables. However, if (c_i,c_j) or (c_j,c_i) is an element of the partial ordering $Q^*(C)$, then c_i and c_j must be assigned to different lines in order to fulfill the ordering and hence, the contradiction. The necessity of condition d) can be shown in a similar way.

Sufficiency:

If $(c_i,c_j) \in Q^*(C)$ and $(c_j,c_i) \in Q^*(C)$, then c_j and c_i can be ordered in any way. They can be assigned to the same line and can therefore be a column folding pair. Because of condition b), $Q^*(C)$ is a partial ordering of the columns. Both conditions b) and c) together allow a linear ordering of elements in the column folding pair. The sufficiency of the conditions a) and d) can be shown in a similar way. ■

The mathematical description of the optimum, mixed folding problem of a PLA can be stated as:

Given the personality of a PLA, find the implementable ordered row folding set and the implementable ordered column folding set of maximum total cardinalities.

All three types of optimum PLA folding problems have been shown to be NP-complete [Hach 80]. The fact that this problem is NP-complete suggests a need for heuristic algorithms. A number of heuristic PLA folding algorithms have been reported in the literature. The algorithm proposed by Hachtel *et al.*, [Hach 80, Hach 82] gives good, fast results. The running time of this algorithm is only in the order a of few minutes even for large PLAs with about 50 inputs, 50 outputs and 100 product terms. Grass [Gras 82] presented a branch and bound algorithm which has been reported as giving slightly better results than Hachtel's algorithm. However, as reported in Grass' paper, testing of this algorithm was limited to PLAs with no more than 30 inputs, 20 outputs and 72 product terms because of the enormous computation time and storage requirements of this

algorithm. Another heuristic PLA folding algorithm is bipartite folding. A *bipartite* column folding is a folding in which the breaks in all the folded columns occur at the same point (row). Because of the single break level for columns, the PLA is divided into two parts: the *upper folding region* which contains those folded input and output columns that are above the break and the *lower folding region* which contains the folded input and output columns below the break. The bipartite column folding of an example PLA in Figure 3.2(a) is shown in Figure 3.2(b). Similarly, in bipartite row folding all the breaks in the folded rows occur at the same point (column). A bipartite folding algorithm developed by Egan *et al*. [Egan 82, Egan 84], has

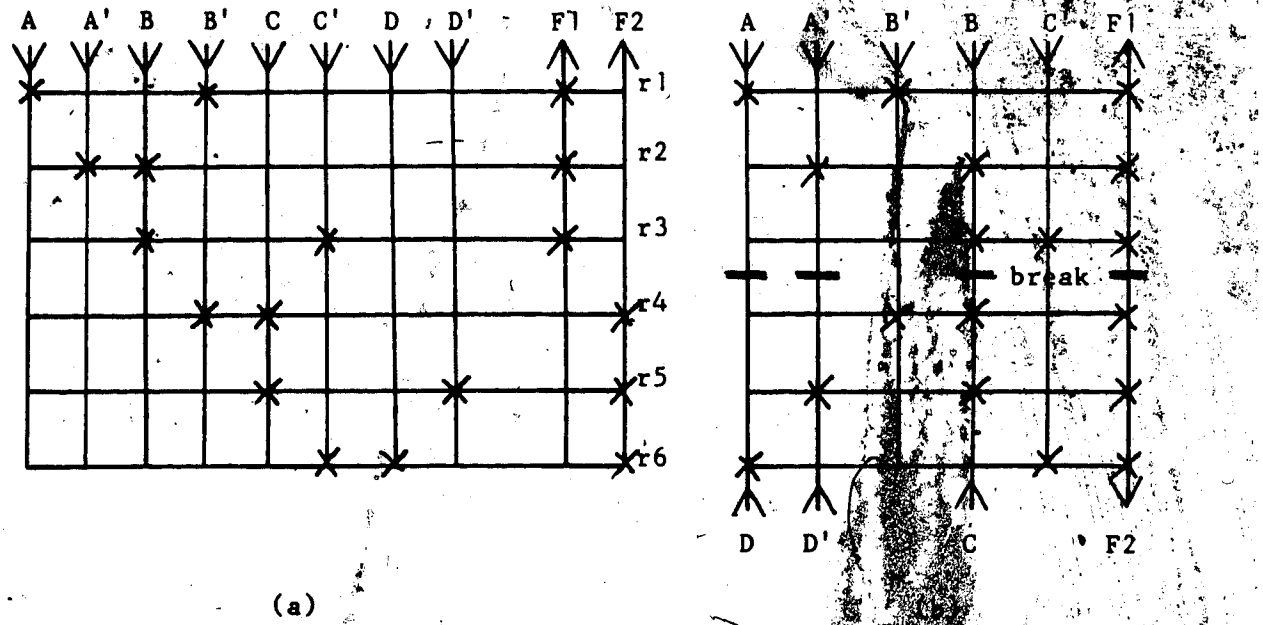


Fig. 3.2 - (a). An example PLA (b). Bipartite folding

been shown to give results comparable to that of Hachtel's, for simple row or column folding. Egan's papers stress that the nature of their folding method facilitates better results in the case mixed folding. The disadvantage of his algorithm is its running time, which could be in the order of a few hours for large PLAs.

In selecting a folding algorithm for implementation, we were looking for the following properties; first, it should be able to efficiently handle large PLA input data, and secondly it must be fast. Of all these algorithms, the algorithm given by Hachtel *et al.*, seems to give good, fast results even for large PLAs. Therefore, we have implemented their algorithm as a first step in studying PLA folding.

3.1 A Graph Theoretic Interpretation of PLA Folding

Hachtel, *et al.*, have developed a graph theoretic interpretation of the folding problem which is useful in understanding their heurisitic PLA folding algorithm.

They define a *column intersection graph* of a PLA, $G = (V,E)$, in which each node v_i represents column c_i and E is the set of undirected edges given by

$$E = \{e = (v_i,v_j) \mid R(c_i) \cap R(c_j) \neq \phi\}.$$

In other words, each edge in E represents a pair of columns which are not disjoint and therefore unfoldable. This definition implies that any pair of nodes which are not adjacent (connected) represents a folding pair. The column intersection graph of the PLA personality of Figure 3.1 is

shown in Figure 3.3(a).

Hachtel, *et al.*, also define a *mixed graph* G(O) = (V,E,A(O)) for a given ordered folding set O. G(O) is a graph with two sets of edges: a set of undirected edges E and a set of directed edges A. V and E are the same as for the column intersection graph and A(O) is defined as a set of directed edges O, such that no two edges share a common node. It should be noted that for a given PLA personality, A(O) is not unique.

A matching is defined as a set of edges such that no two edges share a common node. Therefore, by definition A(O) is a *matching* in graph G(O). The two nodes defining each edge of A identify a disjoint column pair. In addition, the direction of the edges identifies the relative position of the two columns in the folded PLA. The graph G(O) associated

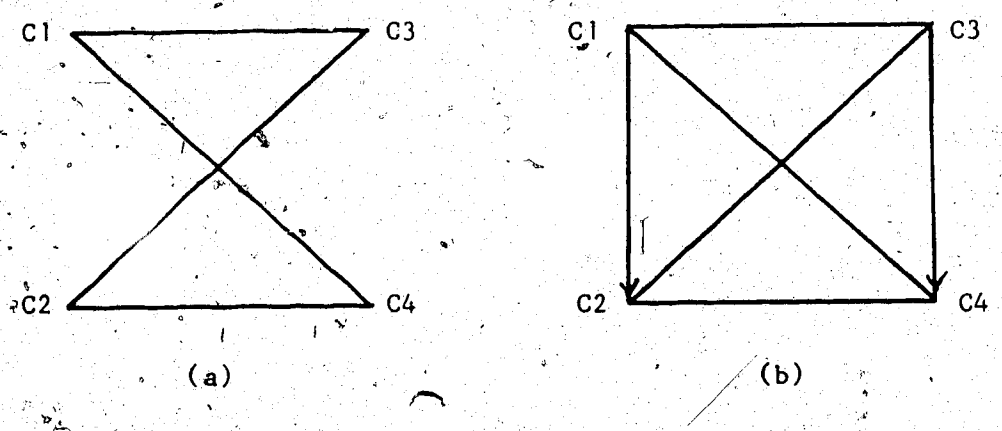


Fig. 3.3 - (a). Column intersection graph (b). Mixed Graph

with the ordered folding set O = {(c_1,c_2) [illegible])} of the personality of the PLA of Figure 3.[illegible] is shown in Figure 3.3(b)

In order to characterize an implementable ordered folding set in graphical terms, Hachtel, *et al.*, introduce the following definitions and lemma.

Alternating Path: A sequence of vertices $\pi \equiv [v_1, v_2, .., v_{2k}]$ of the mixed graph G(O) = (V,E,A(O)) is an alternating path if $(v_{2k-1}, v_{2k}) \in A(O)$, for k = 1,2,...,n and $(v_{2k}, v_{2k+1}) \in E$, for k = 1,2,...,n-1.

Alternating Cycle: Given G(O) = (V,E,A(O)), let $\pi \equiv [v_1, v_2, .., v_{2k}]$, $v_1 \in V$ be an alternating path. Then the sequence $\pi_c \equiv [\pi, v_1]$ is an alternating cycle if $[v_{2k}, v_1] \in E$.

Lemma: Given G(O) = (V,E,A(O)) where G = (V,E) is the column intersection graph of a PLA and where A(O) is a matching of G, the induced ordered folding set O is implementable if and only if G(O) does not contain alternating cycles.

Proof: This lemma can be proved by contradiction.

Sufficiency:

Assume A(O) is implementable. For the sake of contradiction, assume that G(O) has an alternating cycle $\pi_c \equiv [c_1, c_2, .., c_n, c_n, c_n, c_1]$. By definition of alternating cycle, $(c_1, c_2) \in A$, hence $(c_1, c_2) \in O$. This implies that

$$R(c_1) < R(c_2). \qquad (1)$$

Again by definition of alternating cycle, $(c_2, c_3) \in E$. Now consider a row $r \in R(c_2) \cap R(c_3)$. From (1),

$$R(c_1) < r. \tag{2}$$

Now by definition of r we have $r \in R(c_3)$, and by definition of alternating cycle, we have

$$R(c_3) < R(c_4). \tag{3}$$

Thus from (2) we have $R(c_1) < r < R(c_4)$.

Hence the relation $R(c_1) < R(c_4)$ is in $Q^*(R)$ since $Q^*(R)$ is transitive. By the same argument we may conclude that

$$R(c_1) < R(c_{n-2}) \tag{4}$$

Now, let $\hat{r} \in R(c_{n-1}) \cap R(c_{n-2})$.

From (4) we have $R(c_1) < \hat{r}$. (5)

Finally, let $r' \in R(c_n) \cap R(c_1)$. Since by the definition of alternating cycle, $R(c_{n-1}) < R(c_n)$,

$$\hat{r} < r'. \tag{6}$$

But, since $r' \in R(\hat{c})$, from (4) we must have

$$r' < \hat{r} \tag{7}$$

which contradicts the hypothesis that A(O) is implementable since this implies $Q^*(R)$ is a partial order.

Necessity:

Assume that G(O) = (V,E,A(O)) has no alternating cycles. For the sake of contradiction, assume that $Q^*(R)$ is not a partial order. Therefore, there exists two rows $\hat{r}_1$ and r'_1 such that

$$\hat{r}_1 < r'_1, \tag{8}$$

and

$$r'_1 < \hat{r}_1. \tag{9}$$

By (8) and by definition of transitive closure, there exits a sequence

$$[\hat{r}_1, \hat{r}_2, .. \hat{r}_n, r'_1] \quad (10)$$

such that

$\hat{r}_1 \in R(s_{x1})$, $\hat{r}_2 \in R(t_{x1})$,

$\hat{r}_2 \in R(s_{x2})$, $\hat{r}_3 \in R(t_{x2})$, ...

$r'_1 \in R(t_{xn})$ and

for $i = 1, ...n$,

$$o_{xi} = (s_{xi}, t_{xi}) \in A(O). \quad (11)$$

This implies that

$$R(t_{xi}) \cap R(s_{xi}) \neq \phi, \; i = 1, ...n-1. \quad (12)$$

Furthermore, by (9) and by definition of transtive closure, there exits a sequence

$$[r'_1, r'_2, ...r'_n, \hat{r}_1] \quad (13)$$

such that

$r'_1 \in R(s_{y1})$, $r'_2 \in R(t_{y1})$,

$r'_2 \in R(s_{y2})$, $r'_3 \in R(t_{y2})$, ...

$\hat{r}_1 \in R(t_{ym})$ and

for $i = 1, ...m$,

$$o_{yi} = (s_{yi}, t_{yi}) \in A(O). \quad (14)$$

This implies that

$$R(s_{yi}) \cap R(t_{yi}) \neq \phi, \; i = 1, ...m. \quad (15)$$

By (12) and (15) and by definition of E, we have that, for $i = 1, ..., n-1$,

$$\{t_{xi}, s_{xi}\} \in E \quad (16)$$

and that, for $i = 1, ...m-1$,

$$\{t_{yi}, s_{yi}\} \in E. \quad (17)$$

Thus

$$[s_{x1}, t_{x1}, s_{x2}, ..., s_{y1}, t_{y1}, ..., s_{ym}, t_{ym}, s_{x1}] \quad (18)$$

is an alternating cycle in G(O) = (V,E,A(O)) which contradicts the hypothesis.■

The ordered folding set O induced by the set of directed edges in the graph of Figure 3.2(b) is not implementable since the sequence [1,2,3,4] is an alternating cycle.

The fundamental theorem of Hachtel, *et al.*, pertaining to the graph theoretic interpretation of the PLA folding problem is stated as follows.

Theorem 4:

Let P be the following graph problem: Given the column intersection graph G = (V,E) for a PLA personality, find a maximum cardinality set A such that:

1. A(O) is a matching of G = (V,E);
2. G(O) = (V,E,A(O)) has no alternating cycles.

P is equivalent to the optimum PLA folding problem.

This theorem is a direct consequence of the lemma stated above.

The heuristic PLA folding algorithm developed by Hachtel, *et al.*, constructs a set A satisfying properties 1 and 2 of the problem P. The construction is done incrementally, *i.e.* by adding directed edges to the set A one at a time, without backtracking.

3.2. The Algorithm and the Heuristics

The structure of Hachtel's algorithm for column folding is described in *Pidgin Algol* in Figure 3.4. The uppercase letters indicate sets of columns and the lowercase letters represent individual columns. Procedure VSELECT finds columns to be on top of a folded pair of columns and USELECT finds the columns to be on bottom of the folded pair of columns. Procedure CYCLE checks if column folding pairs creates alternating cycles. Procedure TRANS keeps track of relations induced by selected column folding pairs in a data structure called ROW-MATRIX and TRANSCLOS finds the transitive closure of such relations. Procedure SORT sorts the relations stored in ROW-MATRIX into a linear ordering. A detailed description of these procedures will be given later.

In practical PLAs, the number of implementable folding pairs is limited only by the creation of alternating cycles and not by the availability of disjoint pairs. Hence, the selection procedures VSELECT and USELECT must aim at minimizing the number of alternating cycles for the pairs which remain to be considered for folding. To show how this is done, first let us define the degree of a column or row to be the number of "cares" (transistors) in that column or row. The number of pairs in $R(c_i) \leq R(c_j)$ induced by folding an ordered column pair (c_i, c_j) is the product of the degree of c_i and the degree of c_j. These pairs in turn influence the implementability of other column pairs by building up

```
Program FOLD1;
begin
  A ← φ;
  V' ← U' ← V;
  Label : while (V' ≠ φ)
    begin
      v ← VSELECT(V');
      U'(v) = {u | u ∈ U',u ≠ v,{u,v} ∉ E};
      while (U'(v) ≠ φ) do
      begin
        u ← USELECT(U'(v));
        if (CYCLE(u,v,ROW-MATRIX))
           U'(v) ← U'(v) - {u};
        else begin
          ROW-MATRIX ← TRANS(ROW-MATRIX,v,u);
          ROW-MATRIX ← TRANSCLOS(ROW-MATRIX);
          A  ← A  {(v,u)};
          V' ← V' - {v,u};
          U' ← U' - {v,u};
          goto Label;
        end
      end
      V' ← V' - {v};
    end
    Write A;
    SORT(ROW-MATRIX);
end.
```

Fig. 3.4 - Hachtel's PLA folding algorithm

alternating cycles. Therefore, particular attention must be paid to the degrees of the columns being considered for folding. Selecting v and u with maximum degree will induce a large number of pairs in the relation initially and therefore, potentially introduce alternating cycles for most of the later column pairs. Selecting v and u with minimum degree will leave columns of larger degree to be folded later on. However, chances of pairs of these columns being disjoint are slight and the number of folding pairs will be

reduced. Therefore, the remaining strategy is to either select v with minimum degree and u with maximum degree, or to select v with maximum degree and u with minimum degree. It has been reported in Hachtel's papers that the former strategy gives better results for practical PLAs which are usually very sparse. Our experiments show that the latter strategy also gives better results in some instances. Hence, in our implementation the user is allowed to choose either one of these strategies.

3.3 Implementation

The algorithm described above has been implemented as the program FOLD1 in the C language under Berkeley Unix 4.2 operating system on a VAX 11/780. This program is capable of performing column folding, row folding and mixed folding. Input columns are folded only with input columns and output columns are folded only with output columns. Row folding produces only the OR-AND-OR structure, since the dynamic CMOS PLA structure described in Chapter 2 does not allow the AND-OR-AND structure. For state machines, the feedback paths are only allowed on top of the right OR array.

We have used a simple and efficient matrix-like data structure suggested by Grass [Gras 82] to keep track of the relations, to check for alternating cycles, to find the transitive closures of relations, and to sort the relations to find the final placements of rows and columns. The nature of this data structure will be discussed in connection with

the procedure TRANS in the next section because TRANS maintains this data structure.

Procedure TRANS:

This procedure keeps track of the relations induced by selected folding pairs. The column relations induced by row folding are entered in a matrix called COL-MATRIX of size m x m, where m is the number of columns in the PLA. The row relations induced by column folding are entered in a matrix called ROW-MATRIX of size n x n, where n is the number of rows in the PLA. Procedure TRANS maintains COL-MATRIX and ROW-MATRIX. Consider the column folding pair (c_1, c_2) of the PLA in Figure 3.1. This pair induces the relation Q_1 = {r1 < r3, r1 < r4, r2 < r3 and r2 < r4}. This relation is entered in ROW-MATRIX as shown in Figure 3.4. A '1' (TRUE) in position (i,j) in ROW-MATRIX indicates the relation $r_i < r_j$. A '0' (FALSE) in that position indicates a "do not care". The column relations induced by row folding

	r1	r2	r3	r4
r1	0	0	1	1
r2	0	0	1	1
r3	0	0	0	0
r4	0	0	0	0

1 = TRUE
0 = 'DO NOT CARE'

Fig. 3.5 - Data structure for ROW-MATRIX

are similarly entered in COL-MATRIX.

Besides entering the actual relations induced by the folding pairs, it is necessary to include additional relations to ensure that row folding pairs conform to either the AND-OR-AND or OR-AND-OR PLA structure, to maintain the column (row) folds already made when row (column) folding in the case of mixed folding, and to avoid crossovers in the feedback connections in the case of folding PLAs which implement sequential logic. This will be elaborated upon.

Consider the case of row folding under OR-AND-OR structure. In order to maintain this structure and also for ease of sorting the relations at the end of the folding operation, we could make all the outputs produced by left rows "less than" all the inputs and all the inputs "less than" all the outputs produced by right rows.

In mixed folding, the final positions of input, output and product term lines are determined only after finding all the column and row folding pairs. Therefore, additional relations which will maintain the integrity of folds made up to the point should be entered appropriately. Consider a situation where row folding has been performed first, inducing a relation $c_1 < c_3$. Folding the columns c_i and c_j implies that the position of c_i and c_j will be the same in the final PLA structure. Now, suppose column folding is performed and (c_1, c_4) is selected as a column folding pair. In order for COL-MATRIX to indicate that the position of the c_1 is the same as the position of c_4, we should enter the

additional relation $c_a < c_b$. In general, for column folding after row folding, we should make the following additional relations. If $(c_i,c_j) \in$ CFP and if $c_i < c_k$ is in COL-MATRIX then $c_j < c_k$ should be added to COL-MATRIX or if $(c_i,c_j) \in$ CFP and if $c_j < c_k$ is in COL-MATRIX then $c_i < c_k$ should be added to COL-MATRIX. Similar adjustments must be made in ROW-MATRIX when performing row folding after column folding.

When folding sequential logic implemented as PLAs, it is advisable to keep all the inputs and outputs being connected by the feedback connections as close as possible and ordered so that there are no crossovers in the feedback connections. Sometimes the designer may want to keep all the feedback connections together on one side of the PLA, for example, on the top side of the right OR array. In such cases, additional relations should be entered before folding in order to achieve the required structure. Our data structure can easily incorporate such requirements.

Procedure CYCLE:

Procedure CYCLE checks whether a folding pair would create alternating cycles when added to the folding pairs already selected. For simple column folding, the only constraint for a disjoint folding pair to be implementable is that the relations induced by the set of ordered folding pairs should not contain alternating cycles on the rows. In the matrix representation of the relations, this constraint can be checked easily and efficiently. If (c_i,c_j) is the

pair being considered for folding, and any of the relations induced by the reversed ordered pair (c_j,c_i) is already present, then (c_i,c_j) creates an alternating cycle. As an example, consider column folding of the example PLA shown in Figure 3.1 when (c_1,c_2) has been folded first, inducing the relations $r_1 < r_3$, $r_1 < r_4$, $r_2 < r_3$ and $r_2 < r_4$. If we are checking to see if (c_3,c_4) is implementable, we note that of the reversed pair (c_4,c_3), will induce a relation $r_2 < r_4$ which is already present, and therefore (c_3,c_4) is not implementable. Checking for alternating cycles for simple row or column folding can be done in constant time in this manner.

In the case of mixed folding, the procedure for checking for alternating cycles is more complex. For example, consider performing column folding after row folding. Row folding can be performed with the only constraint that it does not create any alternating cycles in the columns of the PLA, *i.e.* $Q^*(C)$ remains asymmetric. Now, in order to be an implementable column folding pair (c_i,c_j) must satisfy the following conditions:

1. $(c_i,c_j) \notin Q^*(C)$ and $(c_j,c_i) \notin Q^*(C)$,
2. (c_i,c_j) should not create any alternating cycles among the rows, that is, $Q^*(R)$ should remain asymmetric, and
3. (c_i,c_j) should not create any relations involving the two elements in any of the row folding pairs.

In our implementation, the first condition is taken care of by eliminating all the column folding pairs which

are part of the relation in $Q^*(C)$, after the completion of row folding. This also reduces the number of pairs being considered for column folding. The second condition is checked for, in the same way as in simple column folding. If this condition is satisfied, then we check for the third condition. The relation in ROW-MATRIX is copied into a temporary matrix, the relations induced by the column folding pairs are added to the temporary matrix and the transitive closure of the temporary matrix is found. Then the matrix is checked for the presence of any relations involving the two elements in any of the row folding pairs. If any such relation is found, the column pair being considered is not implementable. The complexity of checking alternating cycles for mixed folding is $O(n^3)$. This is because, in the worst case, the transitive closure of the temporary matrix should be found.

Procedure TRANSCLOS:

Every time a folding pair is added to the folding set, the relations induced by that pair are entered in the corresponding matrix or matrices and the transitive closure of the relations is found before adding another pair. We have used Warshall's algorithm shown in Figure 3.5 to find the transitive closure of the relations. The complexity of this algorithm is $O(n^3)$.

Procedure SORT:

After the folding process has been completed it is necessary to sort the relations into a linear ordering so as

```
Procedure TRANSCLOS(var A : array[1..n,1..n] of boolean);
var i,j,k : integer;
begin
    for k := 1 to n do
      for i := 1 to n do
        for j := 1 to n do
          if (A[i,j] == FALSE) then
            A[i,j] = A[i,k] and A[k,j];
end.
```

Fig. 3.6 - Warshall's algorithm for transitive closure

```
Procedure SORT(var A : array[1..n,1..n] of boolean);
var C : array[1...n] of boolean;
    i : integer;
begin
   for i := 1 to n do C[i] = FALSE;
   for i := 1 to n do
      if (C[i] = FALSE)
         topsort(i,C,n,A);
end.

procedure topsort(i,C,n,A);
var j : integer;
begin
   C[i] = TRUE;
   for j := 1 to n do
      if ((A[i][j] = TRUE) and (C[j] = FALSE))
         topsort(j,C,n,A);
   print(i);
end.
```

Fig. 3.7 - An algorithm for topological sorting

to determine the final folded PLA structure. Topological sorting is a process of assigning a linear ordering to the vertices of a directed graph so that if there is an arc from vertex i to vertex j, then i appears before j in the linear ordering. The relations in COL-MATRIX and ROW-MATRIX are

sorted using the procedures in Figure 3.7.

When topsort finishes searching all vertices adjacent to a given vertex x, it prints x. The effect of calling *topsort* with i = x is to print in a reverse order all vertices of a directed graph accessible from x by a path in the graph. In our programs, since the relations being sorted are asymmetric (no back arcs), this method works efficiently. The complexity of this algorithm is $O(n^2)$.

3.4 Complexity Analysis

For the analysis of the complexity of this PLA folding algorithm, let us consider column folding. Let c, r be the number of columns and rows respectively of a PLA. The worst case complexity of procedures VSELECT and USELECT are O(c), since all the columns should be examined to select a column with minimum or maximum degree. As explained earlier, alternating cycles can be checked (procedure CYCLE) in constant time. The relations induced by each selected column folding pair can be entered in constant time (procedure TRANS). This constant is given by the degree of the top column multiplied by the degree of the bottom column. Procedure TRANSCLOS is an $O(r^3)$ algorithm and procedure sort is an $O(r^2)$ algorithm. The number of passes through the algorithm is c^2 since there can be c^2 folding pairs in the worst case. Therefore, the overall worst case complexity of the column folding algorithm is $O(c^2r^3)$. In our implementation the number of passes through this algorithm

c^2, is minimized by separating input column folding from output column folding.

The algorithm for row folding is identical to that of the column folding. Hence, the complexity of the row folding algorithm is $O(r^2c^3)$. Since we are considering only the sequential mixed folding, the complexity of mixed folding also will be the sum of the complexities of column and row folding. Howeyer, it should be noted that the procedure CYCLE, requires only constant time in row folding alone but $O(c^3)$ time when performing row folding after column folding. Therefore, the constant factor for the row (column) folding after column (row) folding will be higher than row (column) folding alone.

Storage requirements

The largest amount of storage required is for the matrices COL-MATRIX and ROW-MATRIX. The storage required for COL-MATRIX is $Z \times Z$, where Z is the sum of the inputs and the outputs in the PLA. The storage required for ROW-MATRIX is $R \times R$, where R is the number of product terms in the PLA. Program FOLD1 can handle PLAs with upto 125 inputs, 125 outputs and 250 product terms.

3.5 Results

We have tested this algorithm on 20 example PLAs. Ten of them are combinational networks and the other ten are synchronous state machines. The combinaltional logic PLAs are networks used for code conversions. For example, the

test PLA #8 is a network for 32-bit binary to grey code conversion. The state machine examples have been taken from various textbooks, and they are extensively used in practical applications. For example, the test PLA #16 describes a vending machine. The results of row folding are listed in Table 3.1, the results of column folding are listed in Table 3.2 and the results of mixed folding are listed in Table 3.3.

Table 3.1 - Results of column folding using algorithm FOLD1

No.	PLA SIZE (IxOxP)	# OF TRANS	# OF STATES	CFP	TIME† sec	% AREA SAVING
1	8x8x15	44	0	7	2.8	44
2	10x9x18	54	0	6	6.3	32
3	12x20x24	81	0	13	13.8	41
4	16x16x31	92	0	15	22.5	47
5	8x10x12	100	0	5	2.8	28
6	10x20x30	122	0	10	24.5	34
7	26x20x43	131	0	20	87.6	44
8	32x32x63	188	0	31	315.6	49
9	5x32x32	192	0	16	29.0	44
10	8x8x107	812	0	4	295	25
11	4x5x4	22	5	2	1.3	23
12	6x7x9	62	4	2	2.8	16
13	11x5x12	82	7	4	5.5	25
14	14x12x17	132	7	9	9.9	35
15	14x14x19	167	10	9	11.5	33
16	11x9x24	197	14	5	23.7	25
17	11x10x25	226	12	5	20.5	24
18	15x13x26	234	16	10	22.6	36
19	12x20x26	275	14	9	23.6	29
20	15x14x33	311	17	10	39.2	35

† : includes time taken by PARSE-PLA and PRESTO

Table 3.2 -Results of row folding using algorithm FOLD1

No.	PLA SIZE (IxOxP)	# OF TRANS	# OF STATES	RFP	TIME† sec	% AREA SAVING
1	8x8x15	44	0	3	2.3	20
2	10x9x18	54	0	5	6.0	28
3	12x20x24	81	0	5	13.0	21
4	16x16x31	92	0	14	19.6	46
5	8x10x12	100	0	0	2.4	0
6	10x20x30	122	0	4	19.0	14
7	26x20x43	131	0	11	66.6	26
8	32x32x63	188	0	30	299.6	48
9	5x32x32	192	0	0	8.6	0
10	8x8x107	812	0	0	102.7	0
11	4x5x4	22	5	0	1.1	0
12	6x7x9	62	4	0	2.7	0
13	11x5x12	82	7	0	5.0	0
14	14x12x17	132	7	0	7.5	0
15	14x14x19	167	10	0	8.9	0
16	11x9x24	197	14	0	21.1	0
17	11x10x25	226	12	0	18.0	0
18	15x13x26	234	16	0	15.6	0
19	12x20x26	275	14	0	17.1	0
20	15x14x33	311	17	0	25.4	0

† : includes time taken by PARSE-PLA and PRESTO

Table 3.3 -Results of mixed folding using algorithm FOLD1

No.	PLA SIZE (IxOxP)	# OF TRANS	# OF STATES	CFP	RFP	TIME† sec	% AREA SAVING
1	8x8x15	44	0	7	5	[illegible]	[illegible]
2	10x9x18	54	0	6	4	[illegible]	47
3	12x20x24	81	0	13	5	[illegible]	53
4	16x16x31	92	[illegible]	[illegible]	4	44.0	54
5	8x10x12	100	[illegible]	[illegible]	0	3.0	28
6	10x20x30	122	[illegible]	10	2	31.8	38
7	26x20x43	131	0	20	11	198.9	58
8	32x32x63	188	0	31	16	751.4	62
9	5x32x32	192	0	16	0	29.3	44
10	8x8x107	812	0	4	0	292.2	25
11	4x5x4	22	5	2	0	1.2	23
12	6x7x9	62	4	2	0	3.0	16
13	11x5x12	82	7	4	0	5.3	25
14	14x12x17	132	7	9	0	9.8	35
15	14x14x19	167	10	9	0	11.5	33
16	11x9x24	197	14	5	0	23.8	25
17	11x10x25	226	12	5	0	20.6	24
18	15x13x26	234	16	10	0	22.5	36
19	12x20x26	275	14	9	0	23.8	29
20	15x14x33	311	17	10	0	38.8	35

† : includes time taken by PARSE-PLA and PRESTO

Chapter 4

A New PLA Folding Algorithm

The main drawback of Hachtel's PLA folding algorithm is that each time a folding pair is accepted, the algorithm specifies the particular positions to the elements in the folding pair. This in turn fixes a particular direction to the ordering of the relations induced by the folding pair. In Hachtel's column folding algorithm described in Chapter 3, procedure VSELECT specifically searches for a column to be on top of a folded column and USELECT searches for a column to be on the bottom of that folded column. If this pair does not create alternating cycles, it is added to the folding set A and an ordering of the rows is fixed immediately which will enable the implementation of that folding pair. A number of such arbitrary decisions about the directions of input (output) signals in the folded columns will restrict the ability to make further folds in the future. This can be explained using an example PLA shown in Figure 4.1.

If (c_1, c_2) is selected as the first column folding pair with c_1 on top and c_2 on the bottom of a folded column, this will induce a relation $r_1 < r_2$ on the rows of the PLA. If (c_3, c_4) is selected as the second column folding pair with c_3 on top and c_4 on the bottom of a folded column, this will induce a relation $r_3 < r_4$. With this arrangement, it is

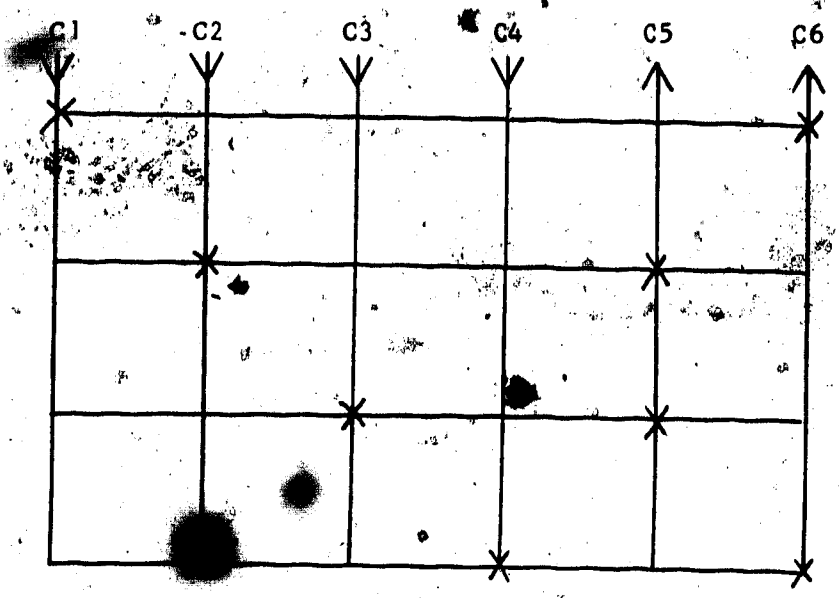


Fig. 4.1 - An example PLA

impossible to fold columns c_5 and c_6 as illustrated in Figure 4.2(a). However, if the direction of the folding pair (c_3, c_4) is reversed so that c_4 will be on top and c_3 on the bottom of a folded column, then it is quite easy to fold columns c_5 and c_6 as shown in Figure 4.2(b).

A PLA folding algorithm should not make arbitrary decisions about the relative placements of elements in the folding pairs. Instead, the relative positions should be established only when a particular folding pair forces the program to decide the relative positions of the elements in the already selected folding pairs. In the example given, the relative positions of columns c_3 and c_4 could have been

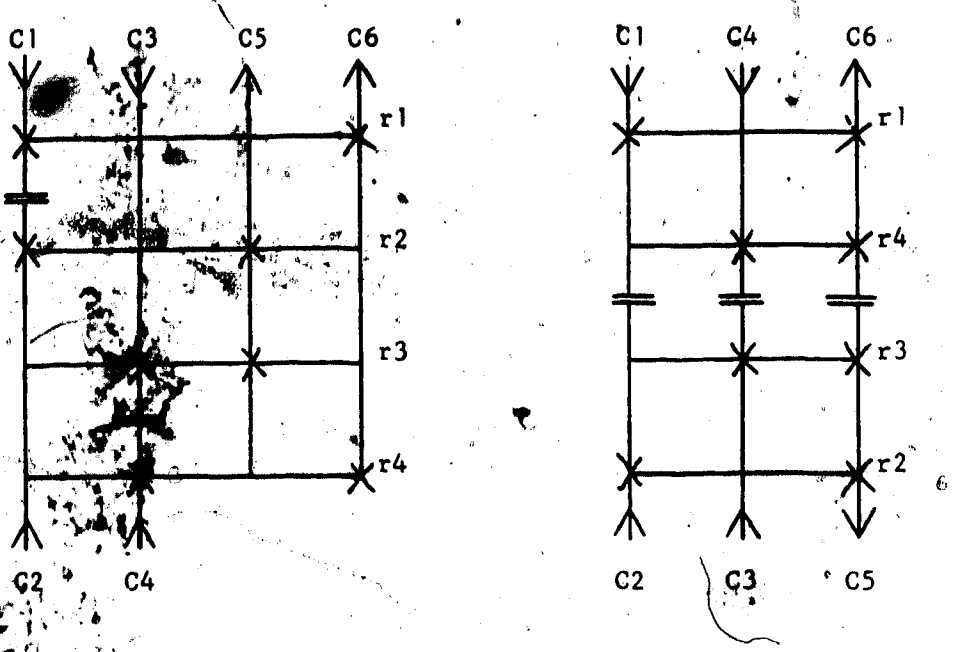


Fig. 4.2 - PLA in Figure 4.1 after column folding

left unspecified until the pair (c_s, c_e) forces the program to state that c_s is on top and c_e is on the bottom of a folded column. This idea forms the basis for our folding algorithm.

This observation about the positions of the elements in the folded columns can be restated in terms of the graph model discussed in Chapter 3. The example PLA in Figure 4.1 can be represented by a mixed graph G = (V,E,A) shown in Figure 4.3(a). In this graph, each vertex in V represents a column, the dashed lines represent the adjacent columns that are not foldable and the solid lines represent disjoint columns *i.e.* the ordered folding set A. In this graph, conflicts in the ordered folding set are indicated by alternating cycles. An alternating cycle is a cycle in which

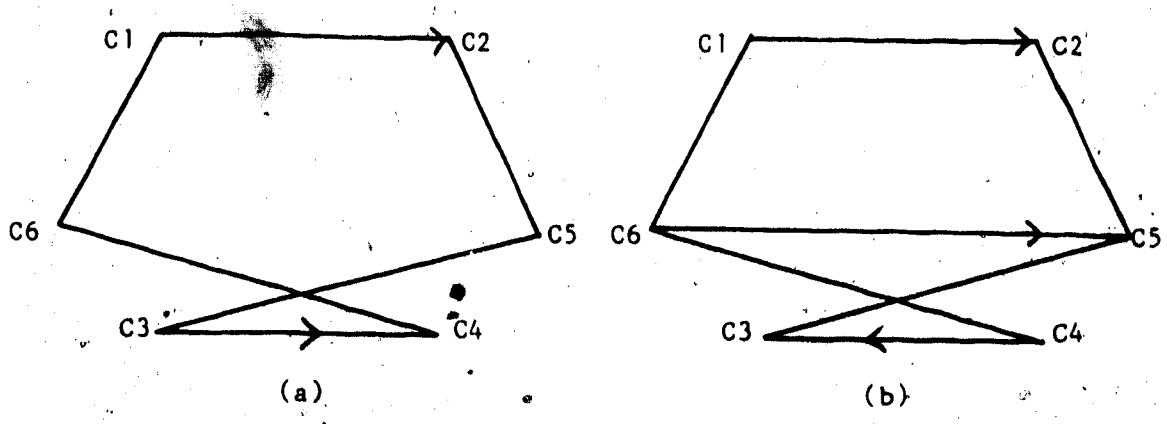


Fig. 4.3 - Mixed graphs of PLA in Figure 4.1

edges alternate between the members of E and members of A. Note that in Figure 4.3(a) there can not be a directed edge between vertices c_3 and c_4 because, no matter which way it points, an alternating cycle will be created. However, if the direction of directed edge c_3c_4 is reversed, then c_4c_3 could be joined by a directed edge without creating alternating cycles as shown in Figure 4.3(b).

4.1 A New Strategy

The observation made in the example PLA, suggests a possible new strategy for maximizing the number of folding pairs.

1. Initially, choose any folding pair and form a set of *independent folding pairs* of maximum cardinality. Two

folding pairs (c_i, c_j) and (c_x, c_y) are independent if:

$(c_i, c_x) \notin E$,

$(c_i, c_y) \notin E$,

$(c_j, c_x) \notin E$, and

$(c_j, c_y) \notin E$.

These independent folding pairs form the folding set A^0. Each folding pair in A^0 is independent of all other folding pairs in A^0 and all edges in A^0 are undirected except for the first one.

2. Choose additional folding pairs which:
   a. do not create alternating cycles, and
   b. determine the direction of as few existing edges in A^m (A after mth edge has been added to A^0) as possible.

Note that adding edges to A^0 to produce A^m may determine the direction of some edges in A^m. For example, suppose A^m is as shown in Figure 4.4(a). Here, the direction of (c_1, c_2) is specified but the direction of (c_3, c_4) is so far unspecified. Suppose we want to add (c_5, c_6) to A^m to form A^m. To do this, we must specify the direction of both (c_5, c_6) and (c_3, c_4), in order to avoid alternating cycles. The final result will be as shown in Figure 4.4(b).

The reason for selecting the independent folding pairs in the first step is this. According to the definition of the independent folding set, each member of a folding pair in A^0 is disjoint from each member in other folding pairs in

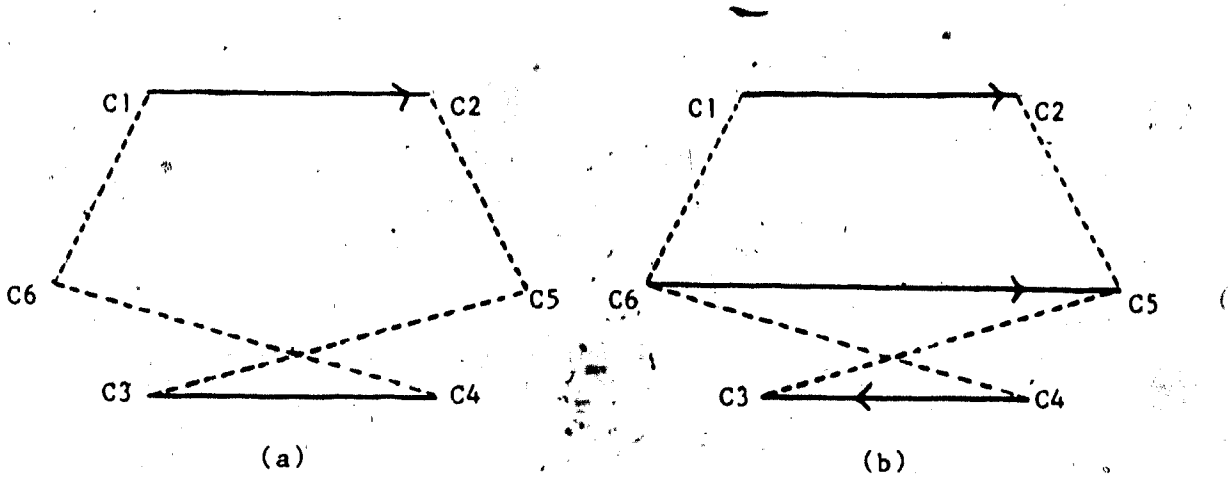


Fig. 4.4 - Mixed graphs of PLA in Figure 4.1

A°. Therefore, the set of induced relations by each folding pair in A° will be disjoint from the set of relations induced by the other folding pairs in A°. This gives us the freedom to alter the direction of the folding pairs in A° (except the first pair) and thereby the direction of the ordered relations induced by them in such a way as to maximize the number of folding pairs obtained. In the above example, the folding pair (c_3, c_4) is independent of the first folding pair (c_1, c_2). We chose to assign a direction to the folding pair (c_3, c_4) which enabled us to fold columns c_5 and c_3.

For the same reason, in step 2, other folding pairs are selected in such a way that we can delay the decision regarding the direction of the folding pairs in A, as long

as possible. The directions of the folding pairs are decided only when the particular selection of folding pairs forces us to do so.

These two steps are used in our algorithm. It should be noted that the first folding pair in A^0 critically influences the cardinality of the set A^0. Therefore, particular attention must be paid to the selection of the first folding pair in order to maximize the cardinality of the set A^0. Our experimental results suggests that using Hachtel's method of selection described in Chapter 3 results in better results in most cases. Therefore his method has been used in our algorithm. However, we feel that more research need to be done in order to establish a criterion for the selection of the first folding pair that will yield a maximum cardinality set A^0.

As an example, the column folding algorithm is described in *Pidgin Algol* in Figure 4.5. The following variables are used in this algorithm. P is the set of all possible folding pairs. A, the set of implementable folding pairs is initially empty. A_i, P_i refer to the ith folding pair in A and P respectively. N1 and N2 are the cardinalities of the sets A and P respectively.

In the implementation of this algorithm we also used the data structures ROW-MATRIX and COL-MATRIX described in Chapter 3 to keep track of the row and column relations induced by column and row folding pairs respectively. Procedures TRANS, CYCLE and TRANSCLOS are also utilized.

```
Program FOLD2;

begin

    A ← φ;

    Build P, the set of all possible folding pairs;

    Select the first folding pair p1 ∈ P and add it to A
    with its direction fixed;

    Find all the independent folding pairs and add them to A
    with their direction unfixed;

    For i := 1 to N1 DELETE(Ai,P);

    Find N2;

    while (N2 > 0) do begin

        For i := 1 to N2 if CYCLE(pi) REMOVE(pi, P);

        Find the merits of the remaining pairs in P;

        Select a pair pi ∈ Pi, with a minimum merit Mi;

        If (Mi > 0)

            Fix the direction of the pairs in A which would
            create alternating cycles with pi;

        Add pi to A with its direction unfixed.

        DELETE(pi, P);

        Find N2;

    end;

    Fix the directions of the pairs in A whose directions
    are not already fixed;

    Write(A);

end;
```

Fig. 4.5 - A new PLA folding algorithm

Fixing the direction of a column folding pair means that a decision is made regarding the positions of the columns in the folding pair. The row relation which will enable us to implement this column fold is entered immediately in ROW-MATRIX. If the direction of a column folding pair is *unfixed*, it means that the potential column folding pair is added to the folding set A but the positions of the columns in the folding pair are not yet determined. Hence, the row relation induced by this pair will not be entered in ROW-MATRIX at that moment. In terms of the procedures described in Chapter 3, adding a column folding pair (c_i, c_j) to A with its direction *fixed* is equivalent to

$A \leftarrow A \cup \{(c_i, c_j)\}$;

ROW-MATRIX ← **TRANS**(ROW-MATRIX, c_i, c_j);

N1 = N1 + 1;

Adding this pair with its direction *unfixed* is equivalent to

$A \leftarrow A \cup \{(c_i, c_j)\}$;

N1 = N1 + 1;

At any given instance the set A may contain folding pairs whose directions are fixed as well as folding pairs whose directions are yet to be established. The *merit* of a folding pair $p_i \in P$, is given by the number of folding pairs in A whose directions must be fixed before adding p_i to A.

If $A_i = (c_m, c_n)$, then DELETE(A_i,P) removes all the folding pairs in P, which have either c_m or c_n as one of their members. Procedure CYCLE checks if a folding pair would create alternating cycles with the existing relations.

Procedure REMOVE removes a particular folding pair from the set P.

Implementation:

The set of all possible column folding pairs, P is maintained in a matrix-like data structure called MATC of size c x c, where c is the sum of the number of inputs and outputs in the PLA. A '0' (FALSE) in position (i,j) in MATC indicates that columns c_i and c_j are not disjoint and therefore unfoldable. A '1' (TRUE) in that position indicates that columns c_i and c_j are disjoint and therefore can be a folding pair. This data is found by checking each column of the PLA for disjointness with all other columns. Then the first folding pair is selected by using the heuristics discussed in Chapter 3 and added to A with its direction *fixed*. This means the row relations induced by the this pair are preserved in a similar way in ROW-MATRIX as in algorithm FOLD1.

In the next step, the set of independent folding pairs is found and added to the folding set A with their direction *unfixed*. The independence of any folding pair can be determined in constant time using the data structure MATC. Therefore the set of independent folding pairs can be found in $O(c^2)$ time in the worst case. If (c_i, c_j) is a folding pair in A and if MATC[i,m], MATC[i,n], MATC[j,m] and MATC[j,n] are all "TRUE" then (c_m, c_n) is an independent folding pair. Note that MATC[i,m] = TRUE indicates that columns c_i and c_m are disjoint and therefore $(c_i, c_m) \notin E$.

All such independent column folding pairs are found and added to A with their direction *unfixed*.

The cardinality N1 of the set A is incremented every time a folding pair is added to A. The cardinality N2 of the set can be found in $O(c^2)$ time using the procedure illustrated in Figure 4.6

All the folding pairs in P which would create alternating cycles with the existing relations already entered in ROW-MATRIX are then removed from P. Note that before the first pass through the *while* loop in the algorithm, we have determined the direction of only one folding pair in A, namely A_1. Hence, in the first pass through the *while* loop only those folding pairs which would create alternating cycles when added with A_1 are removed.

The merits of the remaining folding pairs in P are found next. The procedure for finding the merit of folding pairs in P and the data structure for the sets A and P are described in Figure 4.7. Along with the merits of the folding pairs in P, this procedure also determines the folding pairs in A whose direction must be fixed before adding p_i to A. Since this procedure involves finding the transitive closure of temporary matrix, the complexity of this procedure is $O(r^2)$ where r is the number of product terms in the PLA. This is the only time consuming procedure in our algorithm since the merits of each pair in P must be found for every pass through the while loop. In the actual implementation of this algorithm, we have reduced the time

```
begin
    N2 := 0;
    for i := 1 to c
        for j := 1 to c
            if (MATC[i,j] == TRUE) then N2 := N2 + 1;
end.
```

Fig 4.6 - Procedure for finding the cardinality of P

spent in this procedure by exiting from the procedure as soon as a folding pair with merit = 0 is found.

A folding pair p_i with minimum merit is then selected. If the merit of this pair is 0 then it is simply added to A with its direction unfixed. Otherwise the direction of some of the folding pairs in A (indicated by P[i].foes) are determined by the procedure illustrated in Figure 4.8

When the set P becomes empty, the direction of the folding pairs in A whose direction are not yet determined can be determined using the procedure similar to the one shown in Figure 4.8.

Complexity Analysis:

For the complexity analysis, let us consider column folding. Let c,r be the number of columns and the number of rows respectively of a PLA. Two columns can be checked for disjointness in O(r) time. Hence it requires $O(c^2r)$ time to

```
Procedure merit(i : integer);
int j,k;
struct  {
            int topcol;
            int botcol;
            int status; /* FIXED or UNFIXED */
            int merit;
            int foes[];
        }   P[], A[];

begin
    P[i].merit = 0;
    copy ROW-MATRIX into a temporary matrix TEMP;
    TEMP ← TRANS(TEMP,P[i].topcol,P[i].botcol);
    for j := 1 to N1
        if (A[j].status == UNFIXED) then
        begin
            TEMP ← TRANS(TEMP,A[i].topcol,A[i].botcol);
            TEMP ← TRANSCLOS(TEMP);
            if TEMP has alternating cycles
            begin
                P[i].merit := k := P[i].merit + 1;
                P[i].foes[k] = j;
            end;
        end;
end;
```

Fig 4.7 - Procedure for finding the merits of folding pairs

```
for k := 1 to P[i].merit
begin
    let (c_m, c_n) be P[i].foes[k];
    if not CYCLE(ROW-MATRIX,c_m,c_n)
        begin
            ROW-MATRIX ← TRANS(ROW-MATRIX,c_m,c_n);
            ROW-MATRIX ← TRANSCLOS(ROW-MATRIX);
        end;
        else if not CYCLE(ROW-MATRIX,c_n,c_m)
        begin
            ROW-MATRIX ← TRANS(ROW-MATRIX,c_n,c_m);
            ROW-MATRIX ← TRANSCLOS(ROW-MATRIX);
        end;
        else begin
```

```
            A ← A - {(cm,cn)};
            N1 = N1 - 1;
        end;

end;
```

Fig. 4.8 - Fixing the direction of folding pairs

build MATC. The set of all possible folding pairs P can be directly built from MATC in $O(c^2)$ time since there can be c^2 folding pairs in the worst case. The first folding pair A_1 is selected in $O(c)$ time using the procedure USELECT and VSELECT used in the algorithm FOLD1. Each folding pair in P can be checked for independence from another pair in A in constant time as described earlier. Procedure DELETE requires $O(c^2)$ time. A particular folding pair in P can be removed in constant time (Procedure REMOVE).

In the main body of the algorithm (the *while* loop), the procedure to find the merit of a folding pair requires finding the transitive closure of ROW-MATRIX. Therefore, the complexity of finding the merit of one folding pair is $O(r^3)$. In the worst case, the merits of all c^2 folding pairs must be found in each pass through the *while* loop and the *while* loop will be repeated c^2 times. Hence, the overall complexity of the column folding algorithm is $O(c^4r^3)$ as compared to Hachtel's $O(c^3r^3)$ algorithm.

In the actual implementation of this algorithm, the time required to find the merits of folding pairs during each pass through the *while* loop is reduced by exiting from

the procedure MERIT as soon as a folding pair with merit = 0 is found. The number of passes through the *while* loop is reduced by eliminating all folding pairs in P which would create alternating cycles in each pass through the *while* loop. The procedures DELETE and REMOVE also considerably reduce the number of passes required through the *while* loop. Separating input column from output column folding further reduces the time spent in this algorithm.

The algorithm for row folding is identical to that of column folding. Therefore, the complexity of the row folding can be shown as $O(r^4c^3)$. The complexity of mixed folding is the sum of the complexities of column and row folding.

4.2 Results

We have tested this algorithm on the same 20 example PLAs used to test Hachtel's algorithm. The results of column folding are listed in Table 4.1; the results of row folding are listed in Table 4.2 and the results of mixed folding are listed in Table 4.3.

Table 4.1 - Results of column folding using algorithm FOLD2

No.	PLA SIZE (IxOxP)	# OF TRANS	# OF STATES	CFP	TIME† sec	% AREA SAVING
1	8x8x15	44	0	7	2.[illegible]	[illegible]
2	10x9x18	54	0	8	6.8	
3	12x20x24	81	0	13	14.6	41
4	16x16x31	92	0	15	24.4	47
5	8x10x12	100	0	5	3.0	28
6	10x20x30	122	0	8	24.1	27
7	26x20x43	131	0	21	91.2	46+
8	32x32x63	188	0	31	319.8	49
9	5x32x32	192	0	16	29.1	44
10	8x8x107	812	0	4	291.8	25
11	4x5x4	22	5	2	1.2	23
12	6x7x9	62	4	2	2.8	16
13	11x5x12	82	7	4	5.3	25
14	14x12x17	132	7	10	9.6	39+
15	14x14x19	167	10	9	11.3	33
16	11x9x24	197	14	5	23.2	25
17	11x10x25	226	12	5	20.2	24
18	15x13x26	234	16	10	21.8	36
19	12x20x26	275	14	12	24.8	38+
20	15x14x33	311	17	11	39.8	38+

† : includes time taken by PARSE-PLA and PRESTO
\+ : more area reduction than FOLD1
\- : less area reduction than FOLD1

Table 4.2 - Results of row folding using algorithm FOLD2

No.	PLA SIZE (IxOxP)	# OF TRANS	# OF STATES	RFP	TIME† sec	% AREA SAVING
1	8x8x15	44	0	7	2.9	47*
2	10x9x18	54	0	7	6.5	39*
3	12x20x24	81	0	10	19.1	42*
4	16x16x31	92	0	14	22.0	46
5	8x10x12	100	0	0	2.3	0
6	10x20x30	122	0	7	21.5	24*
7	26x20x43	131	0	13	86.3	31*
8	32x32x63	188	0	30	296.9	48
9	5x32x32	192	0	0	8.6	0
10	8x8x107	812	0	0	101.9	0
11	4x5x4	22	5	0	1.1	0
12	6x7x9	62	4	0	2.7	0
13	11x5x12	82	7	0	4.8	0
14	14x12x17	132	7	0	7.9	0
15	14x14x19	167	10	0	8.7	0
16	11x9x24	197	14	0	20.4	0
17	11x10x25	226	12	0	17.1	0
18	15x13x26	234	16	0	15.5	0
19	12x20x26	275	14	0	17.1	0
20	15x14x33	311	17	0	25.0	0

† : includes time taken by PARSE-PLA and PRESTO
\+ : more area reduction than FOLD1

Table 4.3 - Results of mixed folding using algorithm FOLD2

No.	PLA SIZE (IxOxP)	# OF TRANS	# OF STATES	CFP	RFP	TIME† sec	% AREA SAVING
1	8x8x15	44	0	7	2	4.2	52-
2	10x9x18	54	0	8	4	12.5	55+
3	12x20x24	81	0	13	6	32.8	56+
4	16x16x31	92	0	15	6	52.3	58+
5	8x10x12	100	0	5	0	3.5	28
6	10x20x30	122	0	8	7	65.8	44+
7	26x20x43	131	0	21	11	488.0	60+
8	32x32x63	188	0	31	14	863.6	60-
9	5x32x32	192	0	16	0	29.1	44
10	8x8x107	812	0	4	0	294.6	25
11	4x5x4	22	5	2	0	1.3	23
12	6x7x9	62	4	2	0	3.0	16
13	11x5x12	82	7	4	0	5.4	25
14	14x12x17	132	7	10	0	10.3	39+
15	14x14x19	167	10	9	0	11.7	33
16	11x9x24	197	14	5	0	23.7	25
17	11x10x25	226	12	5	0	20.9	24
18	15x13x26	234	16	10	0	22.8	36
19	12x20x26	275	14	12	0	25.6	38+
20	15x14x33	311	17	11	0	40.6	38+

† : includes time taken by PARSE-PLA and PRESTO
+ : more area reduction than FOLD1
- : less area reduction than FOLD1

4.3 Analysis and Comparison of the Results

Tables listed here and in Chapter 3 illustrate that folding can considerably reduce the area of PLAs. The area reduction achieved by folding ranges from 16 - 63%. In the case of finite state machines, the area reductions achieved by folding were less than that of combinational networks due to the following reasons. The transistor densities in the columns representing the present and the next states of the finite state machines are very high and the input signal that initiates the reset condition is present in all the product terms and therefore, there are no disjoint product term rows. Hence row folding on finite state machines implemented as PLAs does not yield any area reduction. Also, the constraints induced by the feedback connections inhibit column folding and therefore the area reduction obtained by column folding finite state machines implemented as PLAs is reduced.

A comparison of the results of algorithms FOLD1 and FOLD2 shows that the area reductions obtained by FOLD2 are slightly better or equal to that of FOLD1 in almost all the examples in every type of folding. The additional area reduction obtained by FOLD2 over FOLD1 are 2 - 9% in five examples in column folding, 5 - 27% in five examples in row folding and 2 - 9% in nine examples in mixed folding. FOLD1 obtained 7% more area reduction in one example in column folding and upto 9% more area reduction in two example in mixed folding. The tables also show that time spent in

algorithm FOLD2 is almost equal to that of FOLD1 in most of the examples in every type of folding. Hence, we can conclude that the performance of our algorithm is better than that of Hachtel's. Even though the percentage of additional PLA area reduction achieved by our algorithm may not seem too great, it should be noted that the number of folding pairs selected by our algorithm is considerably higher than that selected by Hachtel's algorithm in most of these examples. It should also be noted that the results obtained by these algorithms are near optimum in most cases.

4.4 Future Research

In the following paragraphs, we offer some suggestions to improve the performance of the PLA folding algorithms and to expand the PLA design system.

1. In our implementation of the algorithms FOLD1 and FOLD2, an input variable and its complement are treated as a single column during the folding process. PLA area reduction obtained by folding may be improved if an input variable and its complement are handled as different columns but assigned to the same part (top or bottom) of neighbouring vertical columns [Gfas 84].
2. Efficient algorithms should be developed to perform input partitioning to produce the *decoded* PLAs discussed in Chapter 2.
3. Significant area reduction can be achieved by dividing a PLA with a large number of product terms into a number

of PLAs with a smaller number of product terms. This procedure is known as *synthesis* or *segmentation* [Kang 82].

4. The storage required for ROW-MATRIX and COL-MATRIX which keep track of the relations induced by folding pairs and the time required to find the transitive closure of such relations can be reduced by utilizing sparse matrix techniques [Wirt 79].
5. Even though the state assignment presented in this thesis gives better results than random state assignment, a better state assignment method should be developed in the future. This state assignment algorithm should be specifically developed for finite state machines implemented as PLAs which will facilitate better performace by the folding algorithms [Giov 85].
6. All the PLA folding algorithms that have been reported in the literature are aimed only at the sequential mixed folding. Further research needs to be done in the area of simultaneous mixed folding.

Chapter 5

Conclusions

PLAs are extensively used in VLSI circuits to implement combinational and sequential logic. The main drawback of using PLAs in VLSI circuits is their poor silicon area utilization. In this thesis, we looked at the steps involved in the PLA design process and the area reduction methods that can be applied at each stage of the PLA design.

The main contribution of this thesis is the development of a new PLA folding algorithm. The basic principle of our folding algorithm is to delay the decision regarding the placements of the variables in a folding pair as long as possible until the selection of another folding pair forces us to decide their positions.

We also implemented the PLA folding algorithm by Hachtel, *et al.*, as a starting point in studying PLA folding and also for comparison purposes. These two algorithms have been tested with 20 example PLAs. Experimental results show that our folding algorithm performs slightly better than most of the examples in obtaining better PLA area reduction. The time required by our algorithm is measured to be about the same as that of the algorithm by Hachtel, *et al.*.

The minor contributions of this thesis are the implementation of an algorithm to make quick state assignments for finite state machines and the dynamic CMOS

PLA generator. These programs and the existing programs PARSE-PLA and PRESTO make a complete PLA design system which is capable of taking the designer from functional description of the logic to be performed to the layout description of the PLA.

We demonstrated that folding can reduce a considerable amount of area of the PLAs. This makes PLAs much more attractive for VLSI circuits. We have also shown that our algorithm performs better than the algorithm by Hachtel, *et al*. without taking any additional amount of computer time. We feel that the work reported in this thesis has made some progress in minimizing the area of PLAs.

REFERENCES

[Arev 78] Arevalo, Z. and Bredeson, J.G., "A Method to Simplify a Boolean Function into a Near Minimal Sum-of-Products for Programmable Logic Arrays", *IEEE Transaction on Computers*. (November 1978), pp. 1028-1039.

[Arms 62] Armstrong, D.B., "A Programmed Algorithm for Assigning Internal Codes to Sequential Machines", *IRE Transactions on Electronic Computers*. (August 1962), pp. 466-472.

[Arms 62] Armstrong, D.B., "On the Efficient Assignment of Internal Codes to Sequential Machines", *IRE Transactions on Electronic Computers*. (October 1962), pp. 611-622.

[Bray 82] Brayton, R.K., Hachtel, G.D., Hemachandra, L.A., Newton, A.R. and Sangiovanni Vincentelli, A.L.M., "A Comparison of Logic Minimization Strategies Using ESPRESSO: An APL Program Package for Partitioned Logic Minimization", *Proceedings International Symposium on Circuits and Systems*. (1982), pp. 42-48.

[Bric 78] Bricaud, P. and Campbell, "Multiple Output Minimization: EMIN", *Wescon 78, Paper 33/3*, (1978).

[Brow 81] Brown, D.W., "A State Machine Synthesizer - SMS", *18th Design Automation Conference*. (1981), pp. 301-305.

[Carr 72] Carr, W.N. and Mize, J.P., *MOS/LSI Design and Applications*. (McGraw-Hill, 1972).

[Chuq 82] Chuquillanqus, S. and Segovia, T.P., "PAOLA : A Tool for Topological Optimization of Large PLAs", *19th Design Automation Conference*. (1982), pp. 300-306.

[Cook 79] Cook, P.W., Ho, C.W. and Schuster, S.E., "A Study in the Use of PLA Based Macros", *IEEE Journal of Solid State Circuits*.

(Vol. SC-14, October 1979), pp.833-840.

[Curt 69] Curtis, H.A., "Systematic Procedures for Realizing Synchronous Sequential Machines Using Flip-flop Memory: Part I", *IEEE Transactions on Computers*. (Vol. C-18, December 1969), pp. 1121-1127.

[Davi 67] Davis, W.A., "An Approach to the Assignment Input Codes", *IEEE Transactions Electronic Computers*. (Vol. EC-16, August 1967), pp. 435-442.

[Dolo 64] Dolotta, A. and McCluskey, E.J., "The Coding of Internal States of Sequential Circuits", *IEEE Transactions Electronic Computers*. (Vol. EC-13, October 1964), pp. 549-563.

[Egan 82] Egan, J.R. and Liu, C.L., "Optimal Bipartite Folding of PLA", *19th Design Automation Conference*. (1982), pp. 141-146.

[Egan 84] Egan, J.R. and Liu, C.L., "Bipartite Folding and Partitioning of a PLA",*IEEE Transactions on Computer-Aided Design*. (Vol. CAD-3, No. 3, July 1984), pp. 191-199.

[Flei 75] Fleisher, H. and Maissel, L.I., "An Introduction to Array Logic", *IBM Journal of Research and Development*. (March 1975), pp. 98-109.

[Gare 79] Garey, M.R. and Johnson, D.S., "Computer and Intractability A guide to the theory of NP-completeness",(W.H. Freeman and company, San Francisco, 1979).

[Giov 83] Giovanni, D.M. and Sangiovanni Vincentelli, A.L.M., "PLEASURE: A Computer Program for Simple/Multiple Constrained/Unconstrained Folding of Programmable Logic Arrays", *20th Design Automation Conference*. (1983), pp. 530-537.

[Giov 85] Giovanni, D.M., Brayton, R.K. and

Sangiovanni Vincentelli, A.L.M., "Optimal state assignments for finite state machines", *IEEE Transactions on CAD*, (Vol. CAD-4, No. 3, July 1985), pp. 269-284.

[Gras 82] Grass, W., "A Depth-first Branch-and-Bound Algorithm for Optimal PLA Folding", *19th Design Automation Conference*. (1982), pp. 133-140.

[Gree 76] Greer, D.L., "An Associative Logic Matrix", *IEEE Journal of Solid State Circuits*. (Vol. SC-11, October 1976).

[Hach 82] Hachtel, G.D., Sangiovanni Vincentelli, A.L.M. and Newton, A.R., "An Algorithm for Optimal PLA Folding", *IEEE Transactions on Computer-Aided-Design of Integrated Circuits and Systems*. (Vol. CAD-1, No. 2, April 1982), pp. 63-67.

[Hach 80] Hachtel, G.D., Sangiovanni Vincentelli, A.L.M. and Newton, A.R., "Some Results in Optimal PLA Folding", *IEEE International Conference on Circuits and Computers*. (October 1980), pp. 1023-1026.

[Hach 82] Hachtel, G.D., Sangiovanni Vincentelli, A.L.M. and Newton, A.R., "Techniques for Programmable Logic Folding", *19th Design Automation Conference*. (1982), pp. 147-155.

[Hong 74] Hong, S.J., Gain, R.G. and Ostapko, D.L., "MINI: a Heuristic Approach for Logic Minimization", *IBM Journal of Research and Development*. (September 1974), pp. 443-458.

[Jone 79] Jones, J.W., "Array Logic Macros", *IBM Journal of Research and Development*. (March 1979), pp. 120-125.

[Kang 81] Kang, S., "Synthesis and optimization of Programmable Logic Arrays",*P.hd Dissertation*, (Stanford University, 1981).

[Lebl 80] Leblond, A., Serrero, G., and Verdillon, A., "Automatic Layout of Symbolic MD-MOS Circuits", *IEEE International Conference on Circuits and Computers.* (October 1980), pp. 772-774.

[Logu 75] Logue, J.C., Brichkman, N.F., and Howley, F., "Hardware Implementation of a Small System in Programmable Logic Arrays", *IBM Journal of Research and Development.* (Vol. 19, 1975), pp. 110-119.

[Logu 81] Logue, J.C., Kleinfelder *et al*, "Technique for Improving Engineering Productivity of VLSI Designs", *IBM Journal of Research and Development.* (Vol. 25, 1981), pp. 107-115.

[Mart 83] Martinez-Carballido, J.F. and Powers, V.M., "PRONTO: Quick PLA Product Reduction", *20th Design Automation Conference.* (1983), pp. 545-552.

[Mavo 83] Mavor, J., Jack, M.A. and Deyer, P.B., *Introduction to MOS LSI Design.* (Addison-Wesley Publishing Company, 1983), pp. 86-109.

[Mccl 79] Mccluskey, E.J., "Designing with PLAs", *Center for Reliable Computing, Technical Note No. 168.* (Stanford University, November 1979).

[Mead 80] Mead, C. and Conway, L., *Introduction to VLSI Systems.* (Addison-Wesley Publishing Company, 1980), pp. 76-84.

[Newt 81] Newton, A.R., "Computer-Aided Design of VLSI Circuits", *Proceedings of the IEEE.* (Vol. 69, No. 10, October 1981), pp. 1189-1199.

[Pail 81] Paillotin, J.F., "Optimization of PLA Area", *18th Design Automation Conference.* (1981), pp. 406-410.

[Pati 79] Patil, S.S. and Welch, T.A., "A Programmable Logic Approach for VLSI", *IEEE Transactions of Computers*, (Vol. C-28, Sep 1979), pp. 594-601.

[Roth 78] Roth, J.P., "Programmed Logic Array Optimization", *IEEE Transactions on Computers*. (Vol. C-27, February 1978), pp. 174-176.

[Schm 80] Schmookler, M., "Design of Logic ALUs Using Multiple PLA Macros", *IBM Journal of Research and Development*. (Vol. 24, No. 1, 1980), pp. 2-14.

[Sego 80] Segovia, T. P., "Topological Optimization of PLAs"; *IMAG Research Report No. 216*. (Grenoble, France, October 1980).

[Stri 79] Stritter, E. and Gunter, T.,"Microprocessor Architecture for a Changing World: the Motorola 68000", *IEEE Computers*. (Vol. 12, No. 2, 1979), pp.43-52.

[Sign 79] *Signetics Bipolar and MOS Memory Data Manual* (1979), pp. 156-188.

[Stor 72] Story, J.R., Harrison, H.J. and Reinhard, E.A., "Optimum State Assignment for Synchronous Sequential Circuits", *IEEE Transactions on Computers*. (Vol. C-21, No. 12, December 1972), pp. 1365-1373.

[Suwa 81] Suwa, I. and Kubitz, W.J. "A Computer Aided Design System for Segmented-Folded PLA Macro Cells", *18th Design Automation Conference*. (1981), pp. 398-405.

[Svob 75] Svoboda, A., "Multiple Output Optimization with Mosaics of Boolean Functions", *IEEE Transactions on Computers*. (Vol. C-24, No. 8, August 1975).
The PRESTO logic minimization program was written by D. Brown and A. Svoboda of Tektronix, Inc.

[Torn 68] Torng, H.C., "An Algorithm for Finding Secondary Assignments of Synchronous Sequential Machines", *IEEE Transactions of Electronic Computers*. (Vol. C-17, May 1968), pp. 461-469.

[Ullm 84] Ullman, J.D., *Computational Aspects of VLSI*. (Computer Science Press, 1984), pp. 311-323.

[Wein 67] Weiner, P. and Smith, E.J., "Optimization of Reduced Dependencies for Synchronous Sequential Machines", *IEEE Transactions of Electronic Computers*. (Vol. EC-16, December 1967), pp. 835-857.

[Wirt 79] Wirth, N., "Algotiyhmrn and Datenstrukturen", (Teubner Vertag, Stuttgart, Auflage 1979).

[Wood 79] Wood, R.A., "A High Density Programmable Logic Array Chip", *IEEE Transactions on Computers*. (Vol. C-28, No. 9, September 1979) pp. 602-608.

[Yahi 79] Yahiko, K., "Logic Design of Programmable Logic Arrays", *IEEE Transactions on Computers*. (Vol. C-28, No. 9, September 1979), pp. 609-616.