An Efficient Algorithm for Bipartite PLA Folding

Chun-Yeh Liu and Kewal K. Saluja, *Senior Member IEEE*

***Abstract*—Programmable Logic Arrays (PLA's) provide a flexible and efficient way of synthesizing arbitrary combinational functions as well as sequential logic circuits. They are used in both LSI and VLSI technologies. The disadvantage of using PLA's is that most PLA's are very sparse. The high sparsity of the PLA results in a significant waste of silicon area.**

PLA folding is a technique which reclaims unused area in the original PLA. In this paper, we propose a column bipartite folding algorithm based on matrix representation. Heuristics are used to reduce the search space and to speed up the search processes. The algorithm has been implemented in C programming language on a SUN-4 workstation. The program was used to study several large PLA's of varying sizes. The experimental results show that in most cases the proposed algorithm finds optimal solution in a reasonable CPU time.

I. Introduction

INTERACTIONS between the structured design techniques and the design of complex VLSI circuits have wide implications for overall design cost and efficiency of digital circuits and systems. Use of a regular structures facilitates the design process and eliminates tedious manual operation. Due to the regularity of the structure and the simplicity of the design, Programmable Logic Arrays (PLA) have found widespread acceptance in the design of digital systems.

The PLA is a hardware form used for implementing two-level multiple-output combinational logic circuit. PLA design is easily automated because of a direct correspondence between physical PLA layout and the personality matrix. The major disadvantage of the PLA is that most practical logic problems leave much PLA area unused. A straightforward physical design results into a significant waste of silicon area, which may be unacceptable. Also, speed and power become critical parameters as the size of the PLA increases [7]. The gate capacitances of the input signals carried by long polysilicon lines become the key factor in determining the timing (speed) performance. In moderate to large PLA's, the polysilicon resistance becomes as important a factor as the capacitance. The signal can be seriously degraded with the large resistance added to the line, no matter how large the drivers are. Further, if the PLA becomes large, the width of the power and the ground lines should also be increased to avoid possible metal migration. Most PLA generators [15] automatically increase the width of the power lines and the ground lines in the PLA, depending on the total current demand.

Manuscript received June 22, 1992; revised May 25, 1993. This work was supported in part by the National Science Foundation under Grant MIP-91111886 and a grant from the AT&T Foundation. This paper was recommended by Associate Editor R. Otten.

The authors are with the Department of Electrical and Computer Engineering, University of Wisconsin–Madison, Madison WI 53706.

IEEE Log Number 9212365.

PLA optimization aims at minimizing the area occupied by the PLA and as a result addresses almost all disadvantages listed above. Two minimization techniques are commonly used to reduce the PLA areas,

1) *Logic minimization*: Logic minimization seeks a logic representation with a minimal number of implicants. Reduction of the number of implicants allows a PLA to be implemented in a small area.
2) *Topological minimization*: PLA folding is a technique which reclaims unused space without destroying the regular structure of the PLA. According to Egan and Liu [2] arbitrary boolean functions produce sparse PLA's, in which typically 90% of the crosspoints are unused. Folding achieves size reduction by compaction and removal of areas of unused crosspoints.

In this paper, we study the problems of the PLA folding. There are many types of PLA folding, depending on the technology employed to implement a PLA. All PLA folding methods involve the merging of two or more columns (rows) of a PLA into a single column (row). The simplest form of folding, called Simple Column Folding [7], involves merging pairs of columns into single columns.

The object of PLA folding is to find the maximum number of pairs of columns/rows that can be folded simultaneously. The PLA folding has a complex functional dependence on the ordering of the rows. The optimal simple PLA column folding problem can be defined as:

Determine a permutation of the rows which allows a maximum set of column pairs to be implemented in such a way that each column of the folded PLA contains a pair of columns from the set.

The optimal folding problem has been shown to be NP-complete [6], [14]. Many algorithms and heuristics have been developed to solve this problem. The simplest one is the branch and bound algorithm [4], [13]. Although it is simple and able to find an optimal solution in theory, its practicality for large PLA's is questionable because it carries out an exhaustive search for an optimal solution. Therefore, many heuristics have been developed to find good, but nonoptimal solutions. Hwang *et al.* [10] used a best-first search algorithm to find a near-optimal result. Ullman [20] used a graph algorithm to find a feasible solution in a time complexity no worse than $O(wc^2)$, where w is proportional to the number of rows and c is the number of columns. Hachtel *et al.* [7], [8] proposed algorithms for both row and column foldings, which find the folding pairs one by one. The PLA folding results thus obtained are only locally optimal and depend on the selection of order of the folding pairs. For example, for column folding, they try to fold as many columns as possible and then determine the row permutation according to the folding set so found. In fact, each folding set corresponds implicitly to some row permutation order. Thus, after a folding set is selected, the next folding set is constrained by the row permutation orders. Wong [22] applied the simulated annealing technique to the folding problem. Lecky *et al.* [12] transformed the PLA into a graph where cliques in the graphs correspond to the PLA folding set, and Greedy algorithm [5], [11] is used to identify the maximal cliques. Hsu *et al.* [9] modeled the PLA personality matrix as network and the bipartite PLA folding as a partitioning problem of the network.

In this paper a new bipartite PLA folding algorithm based on matrix representation is presented. Before searching a bipartite folding, the columns which do not satisfy certain constrains, and hence nonfoldable, are pruned. This reduces the search space. During search, heuristics are used to find an alternative folding. This speeds up the search processes.

This paper is organized as follows: The advantages and constraints of PLA folding are given in Section II. The PLA bipartite folding is also introduced in this section. Terms which are used in the proposed algorithm are defined in Section III. In Section IV, a bipartite folding algorithm is described. Results on the benchmark examples are presented in Section V. Section VI concludes the paper.

II. Advantages and Limitations of PLA Folding

Table I summarizes the Simple Column Folding results of 48 PLA's from the lists of 56 PLA's given in [1]. The results are found with the aid of a folding program "pleasure" [17]. For each PLA, the table shows the number of inputs N_i, the number of outputs N_0, the number of product terms N_p, number of folding pairs in AND plane f_a, number of folding pairs in OR plane f_0, and the relative area RA. The relative area in the last column is defined as

$$RA = \frac{A_f}{A} \times 100$$

where A_f is the area after folding, and A is the original area.

Layout follows the design rules for CMOS technology proposed in [21]. Since the "pleasure" limits the size of the PLA which can be folded, only 48 PLA's out of 56 PLA's are chosen.

For most of the large PLA's, the relative areas are less than 100%. For example the *x2dn* has the relative area 57.0%, which is very close to the optimal lower bound 50% [2]. This table shows that the folding technique is effective for area saving in PLA. Note that the RA's of some small PLA's are greater than 100%, as shown in Table I. This is because the area of placing the extra input decoders and output buffers in a folded PLA is greater than the area saved by folding.

Although PLA folding can reduce the area effectively, there exist constrains, such as routing, on folding a PLA or a folded PLA. In a VLSI system design, these constrains should be taken into account. Two of the important constraints and their impacts are discussed below.

1) *Routing:* In a folded PLA, one of the folded input (output) signals must come from the top of the PLA and the other from the bottom. Since the inputs may be required anywhere and the output may go to anywhere, it often increases the complexity of routing. Furthermore, it is well known that the routing of signals often takes more silicon area than the logic blocks. Typically, 30% of total design time and about 60% of the chips are expended merely to interconnect the circuit elements [18]. Therefore, any calculation made to estimate the overhead without considering the routing area is often too optimistic. Thus, folded PLA design must consider the routing area of interconnects and time to complete the design.
2) *Testing:* The input decoders and output buffers of a testable design of a PLA are generally augmented such that they can control the columns of the PLA easily. For a folded PLA, transistors can be placed between the cuts of the columns, such that the PLA can be controlled by the augmented input decoder only from one side during testing. Otherwise, input decoders at the both sides would need to be augmented to test the PLA. To place a transistor on the cut in a folded PLA can be costly for a CMOS technology design. This is because the layout of a PLA is very compact. For a simple column folding PLA, the cuts are in different levels, the area increased to place the transistors are dependent on the number of cuts in a folded PLA, which results in significant waste of silicon area.

A bipartite folding is a folding in which all of the breaks (cuts) occur at the same level. The single break level of a bipartite folding splits a PLA into two regions [2], an *upper folding region* (U) which contains those folded input and output lines that are above the break, and a *lower folding region* (L) which contains the folded input and output lines that are below the break. A column bipartite folding exists if every line in the upper folding region is disjoint from every line in the lower folding region.

There are several advantages for using bipartite folding:

- Our experimental results show that the size of a bipartite folded PLA compares favorably to the size of the PLA obtained after single column folding.

TABLE I
LIST OF PLA FOLDING AND RELATIVE AREA

Name	N_i	N_o	N_p	f_a	f_o	RA(%)	Name	N_i	N_o	N_p	f_a	f_o	RA(%)
z4	7	4	128	0	0	100.0	Z5xp1	7	10	128	0	1	104.4
adr4	8	5	256	0	0	100.0	alu1	12	8	19	5	4	85.2
alu2	10	8	91	0	4	97.9	alu3	10	8	72	0	4	99.9
apla	10	12	134	0	6	90.8	bcb	26	39	299	10	14	67.5
bcc	26	45	245	10	17	67.0	bcd	26	38	243	10	16	65.4
chkn	29	7	153	6	3	84.3	co14	14	1	47	0	0	100.0
dc1	4	7	15	0	3	119.1	dc2	8	7	58	1	2	99.9
dist	8	5	256	0	1	100.4	dk17	10	11	93	0	5	95.8
dk27	9	9	52	0	4	103.0	dk48	15	17	148	0	8	90.9
exep	30	63	175	3	31	75.7	f51m	8	8	256	0	0	100.0
gary	15	11	214	1	3	93.9	in0	15	11	138	2	1	96.1
in1	16	17	110	2	0	99.0	in2	19	10	137	4	2	87.7
in3	35	29	75	11	11	77.4	in4	32	20	234	11	9	68.7
in5	24	14	62	8	4	81.2	in6	33	23	54	16	9	66.5
in7	26	10	84	8	4	78.9	misg	56	23	75	28	11	59.3
mish	94	43	91	47	21	57.1	mlp4	8	8	256	0	0	100.0
radd	8	5	120	0	2	100.5	rckl	32	7	96	0	3	105.9
rd53	5	3	32	0	1	117.9	rd73	7	3	147	0	1	102.4
risc	8	31	74	1	15	76.9	root	8	5	256	0	1	100.4
sqn	7	3	96	0	0	100.0	sqr6	6	12	64	0	2	106.9
ti	47	72	241	19	35	60.2	vg2	25	8	110	0	4	102.3
wim	4	7	10	0	0	100.0	x1dn	27	6	112	1	3	100.7
x2dn	82	56	112	40	28	57.0	x6dn	39	5	121	14	0	72.9
x9dn	27	7	120	1	3	100.4	z4	7	4	128	0	0	100.0

- Routing the nets to or from the PLA is simplified, since the folded lines entering from the top of the PLA can be ordered independently of the folded lines entering from the bottom of the PLA.
- The area of a PLA can be further reduced by folding the upper folding region and lower folding region. The same algorithm can be applied recursively to the bipartite folded PLA.
- The area required for inclusion of testability features in a bipartite folded PLA is much less than that of a single column folded PLA. Since all of the cuts in a bipartite folded PLA are at the same level, therefore as argued earlier, this PLA can be tested from one side alone with very little area overhead. The idea proposed in [19] can be used to obtain Built-in Self-Test folded PLA by placing pass transistors between the cuts of a bipartite folded PLA.

III. PRELIMINARIES

An example of the AND plane of a PLA is shown in Fig. 1. In this figure, columns represent uncomplement–complement pairs of literals. A dot means the placing of a transistors on uncomplemented or complemented input. Each horizontal line of the PLA carries a product term. There are 13 inputs and 21 product terms in the example PLA. We will use this example AND plane of a PLA throughout this section.

Definition 1: Two columns are *disjoint* (*compatible*) if they do not share a common product line.

More explicitly, two input lines are disjoint if the corresponding inputs do not occur together in any product term for any of the output function. An input line is disjoint from an output line, if the literal represented by this input line is not present in the function represented by the output line. Finally, two output lines are disjoint, if they do not share a product term. For the example PLA, inputs 1 and 5 do not share any common product line, hence they are disjoint. Inputs 9 and 13 appear on product line 1, therefore they are not disjoint.

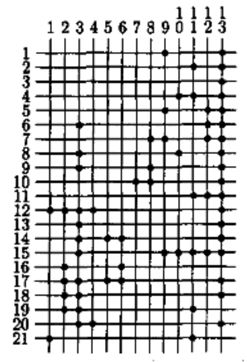
Fig. 1. An example PLA.

Definition 2: A square matrix which depicts the compatibility relation among all pairs of columns of a PLA will be called a *compatibility matrix* (*CM*) of the PLA.

The compatibility matrix contains a row and a column corresponding to each column of the PLA. Therefore, the size of *CM* is the sum of number of inputs and number of outputs. The $CM[i, j]$ element of the compatibility matrix is 1 if columns i and j of the PLA are compatible. *CM* satisfies the following two properties,

- *CM* has all 0's on the leading diagonal, since a column is not disjoint from itself,
- *CM* is symmetric, since if column i is disjoint from j, then j is disjoint from i.

Construction of the matrix *CM* from a given PLA is straightforward. The compatibility matrix of the PLA of Fig. 1 is shown in Fig. 2. The last column, labeled weight, will be explained later.

A *column bipartite folding* is a folding in which all the breaks (cuts) in the columns occur at the same level. Fig. 3 shows a bipartite folding of the example PLA. In Fig. 3, U contains inputs 7, 8, 9, 10, and 12, and L contains inputs 1, 2, 4, 5, and 6. The size of a bipartite folding PLA is the cardinality of either folding region. Since the breaks of a bipartite folding PLA are at the same level, the following lemma is evident and is given without proof.

Lemma 1: A column bipartite folding exists if and only if every line in the upper folding region is disjoint from every line in the lower folding region.

Definition 3: A submatrix of *CM* called a *foldability matrix* (*FM*) provided it satisfies the following properties:

- It is an $m \times m$ matrix where $2m \leq n$, n being the total number of columns of the PLA and m being the number of folding pairs.
- Every element in the upper folding region is the column of *FM*, and it has the lower folding region as its row.
- It has all-1.

An *FM* of the PLA in Fig. 1 is shown in Fig. 4. This *FM* corresponds to the bipartite folded PLA shown in

	1	2	3	4	5	6	7	8	9	10	11	12	13	weight
1	0	0	0	0	1	1	1	1	1	1	0	1	0	7
2	0	0	0	0	0	0	1	1	1	1	0	1	0	5
3	0	0	0	0	0	0	1	0	0	0	0	0	0	1
4	0	0	0	0	1	1	1	1	1	1	1	1	0	8
5	1	0	0	1	0	0	1	1	1	1	1	1	0	8
6	1	0	0	1	0	0	1	1	1	1	1	1	0	8
7	1	1	1	1	1	1	0	0	1	1	1	1	0	10
8	1	1	0	1	1	1	0	0	1	1	1	1	0	9
9	1	1	0	1	1	1	1	1	0	0	0	0	0	7
10	1	1	0	1	1	1	1	1	0	0	0	0	0	7
11	0	0	0	1	1	1	1	1	0	0	0	0	0	5
12	1	1	0	1	1	1	1	1	0	0	0	0	0	7
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 2. Compatibility matrix of the example PLA.

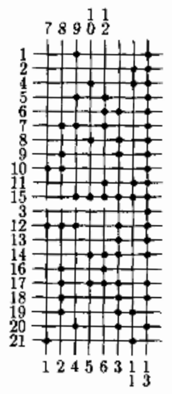

Fig. 3. A bipartite folding result of the example PLA.

	7	8	9	10	12
1	1	1	1	1	1
2	1	1	1	1	1
4	1	1	1	1	1
5	1	1	1	1	1
6	1	1	1	1	1

Fig. 4. Foldable matrix of the folding result.

Fig. 3. In the *FM* the row labels denote the elements in U, and the column labels denote the elements in L.

The optimal column bipartite folding problem can be formulated as the following decision problem:

Instance 1: Positive M, and a compatible matrix *CM*.

Question 1: Is there a foldable matrix *FM* of size M for the PLA with compatibility matrix *CM* and M is maximum?

Theorem 1: This decision problem is NP-complete.

Proof: The reduction is from the following problem which was shown to be NP-complete in [3], p. 196. ■

Instance 2: Bipartite graph $G = (V, E)$, positive integer $M \leq |V|$.

Question 2: Are there two disjoint subsets V_1 and V_2 such that $|V_1| = |V_2| = M$, and such that $u \in V_1$, $v \in V_2$, implies that $\{u, v\} \in E$?

This problem is known as the Balanced Complete Bipartite Subgraph Problem, which has shown to be NP-complete in [16]. G and E are constructed from the given PLA. Each node in the G denotes the column of the PLA, and if $CM[i, j] = 1$ then an edge is placed between the corresponding nodes i and j in G. Thus, two columns are disjoint if and only if there is an edge between the two corresponding nodes in G. V_1 represents the upper folding region (U), and V_2 represents the lower folding region (L). The condition that there exists an edge for each node in V_1 and each node in V_2 implies that each column in U is disjoint from each column in L. The requirement that every line in the upper folding region of a bipartite folding is disjoint from every line in the lower folding region implies that a bipartite folding of a size M exist if and only if there is a foldable matrix (FM) of size M.

Definition 4: In a *CM*, the number of 1's in a column (row) C_i is the *weight* of C_i.

Definition 5: If the weight of a column (row) is zero in a *CM*, then the corresponding column (row) is *expendable* with respect to the corresponding *CM*.

The last column in Fig. 2 shows the weight of each column. Input 13 is expendable. An expendable column (row) can be deleted from the *CM*, since it is nondisjoint from all other columns. In other words, the expendable columns (rows) can not appear in either upper folding region or lower folding region.

Theorem 2: If a PLA contains M bipartite folding pairs then at least $2M$ columns in the *CM* have weights greater than or equal to M.

Proof: In an M pairs bipartite folding PLA, each column in U can be folded with every column in L. Since there are M columns in U, each of these columns must have weight at least M. Likewise, each of the columns in L must also have weight at least M. Hence, the total number of columns with weight M or more in a bipartite folded PLA is at least $2M$. ■

This theorem can be used to reduce the search space for bipartite folding. In other words, if we are looking for a bipartite folding with size M, all candidate columns should have weights greater than or equal to M. For example, there are 12 columns in Fig. 2 after removing the expendable column 13. In the *CM*, the value of M cannot be 6 since there are only 9 columns with weights greater than or equal to 6. Hence, M is upper-bounded by 5 in the given example.

Definition 6: A pair of columns C_i and C_j is called a *companion pair* of order M if weight(C_i) + weight(C_j) $\geq 2M$, where M is the size of bipartite folding.

Theorem 3: Given $2M$ columns with M bipartite folded pairs, the number of companion pairs of these $2M$ columns is greater than or equal to M^2.

Proof: A column in U is disjoint from every column in the L. Hence, the weight of each column in U is at least M. Likewise, every column in L has weight greater than or equal to M. Further, every column in U is a companion pair with every column in L, with weight(C_i) + weight(C_j) $\geq 2M$, where C_i is a column in U and C_j is a column in L. There are M^2 combinations of the columns. Hence, the total number of companion pairs in a bipartite folding is M^2. ■

Theorem 3 gives the basic constrain for bipartite folding for a given set of $2M$ columns. If Theorem 2 is used to find $2M$ columns as the cardinality of a bipartite folding, Theorem 3 must be satisfied for these columns for bipartite folding to exist.

The next section gives a column bipartite folding algorithm based on these two theorems.

IV. The Folding Algorithm and Heuristics

The optimal bipartite folding problem was shown to be NP-complete [2]. Therefore, we propose a heuristic-based algorithm to obtain a "good" solution. To explain the basic ideas behind the algorithm, let us first introduce a frame of the algorithm. The algorithm frame is described in the following structured code[1]

```
    Algorithm BiFolding
begin
    $CM \leftarrow$ FindCompatibilityMatrix(PLA);
    $CM \leftarrow$ RemoveRedundant(CM);
    $M \leftarrow$ MaximumFoldingNumber($CM$);
    $found \leftarrow$ false;
    while ($M \geq 0$ and not $found$) do
    begin
        $V \leftarrow$ WeightSelect($CM$, $M$); /* select columns with weights $\geq M$ */
        if ($\|V\| > 2M$) then
        begin
            while (there exists any column in $V$ which has not been selected
                  and not $found$) do
            begin
                  $V \leftarrow$ Select($V$); /* selection of candidates */
                  InitialFolding($\hat{V}$, $U$, $L$); /* initial arrangement of columns */
                  if CheckFolding ($\hat{V}$, $U$, $L$) then /* test for folding */
                        $found \leftarrow$ true;
            end;
        end;
        $M = M - 1$;
    end;
    write($U$, $L$)
end;
```

BiFolding proceeds as follows. Initially, it selects $2M$ columns from the compatible matrix (CM), where each column has weight greater than or equal to M. If these $2M$ columns cannot be placed into L and U, BiFolding discards one of the columns and selects another one from CM. This process continues until a solution is found or all possible choices have failed. If the search fails, BiFolding reduces the M and tries to find another solution.

Since BiFolding only processes those columns whose weights are at least M, the columns with weights less than M are not considered. This reduces the search space. For a dense PLA, most columns have small weights. This results in a small size of CM, hence, the effort for searching is small. Also any column which has been places in $U(L)$ can easily block the change of placing next column in $L(U)$. For a sparse PLA, most columns have larger weights. Hence, BiFolding can easily find a bipartite solution. The experimental results in Section V show that BiFolding can find a solution efficiently.

The selection steps are important to produce a "good" folding. These steps have to be chosen so that U and L computed by the algorithm have cardinality as close as possible to the maximum.

If we select columns with large degrees (weights), the number of companion pairs is large. Because we have large freedom to arrange the folding pairs, a much easier task is left in the following folding steps. Therefore, it appears that a good heuristic is to select columns with maximum degree as candidates to be folded and place them in candidate FM.

However, selecting a column with large degree for inclusion in FM increases the difficulty of folding. In fact, many possible foldings are created by the choice of maximum degree columns. If we select columns with minimum degree the number of possible alternate foldings is kept small. This argument seems to suggest that we should select columns with minimum degree form candidates to be placed in FM.

Of course these two selection rules are contradictory, therefore a tradeoff must be made. We decided to use the following selection rule: *Select columns with maximum degree as the cardinalities to be folded and place them in candidate FM, then select columns with minimum degree from candidate FM and place it into FM.*

In BiFolding, there are several subprocedures. We now describe these subprocedures and the heuristics used in them.

[1]Algorithms are written in an informal notation call *pidgin algol*. The term *pidgin algol* appears to have been introduced in A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, pp. 33-39, (Reading, MA: Addison-Wesley, 1974).

The algorithm SELECT implements the idea of selecting $2M$ columns with maximum degree from CM. In CM, the columns are sorted into decreasing order according to weight. Hence, SELECT selects the first $2M$ columns which have not been justified. The sorting of CM saves the effort of searching $2M$ columns with weights at least M.

VSELECT is based on choosing a minimum degree column from the $2M$ selected column.

```
    **Algorithm** VSELECT ($V$)
begin
    min ← a large number
    for each $v$ in $V$ do
        if (weight($v$) < min) then
            begin
                min ← weight($v$)
            end
    return($u$)
end;
```

Algorithm INITIALFOLDING(V, U, L), given below, is used to arrange $2M$ columns into U and L. It proceeds a follows: In the beginning of the algorithm, a column v is selected with least freedom using VSELECT, and put into, say U. It then removes v from V and sets L to be an empty set. Then, for each element u in the V, if u is not disjoint from every element in $U(L)$, it is placed in $L(U)$ and removed from V. This step continues until every element in V has been justified.

INITIALFOLDING reduces the search space by fixing some columns in U or L. Since these columns have least freedom (weights), which limits the chance of the folding of other columns.

```
    Algorithm INITIALFOLDING($V$, $U$, $L$)
begin
    $v \leftarrow$ VSELECT($V$);
    $V \leftarrow V - \{v\}$;
    $U \leftarrow \{v\}$;
    $L \leftarrow 0$;
    while ($V \neq 0$) do
    begin
        if not COMPATIBLE($u$, $U$) and not COMPATIBLE
                ($u$, $L$) then
            return false
        if not COMPATIBLE($u$, $U$) then
            $U \leftarrow U \cup \{u\}$;
        if not COMPATIBLE($u$, $L$) then
            $L \leftarrow L \cup \{u\}$;
        $V \leftarrow V - \{u\}$;
    end;
end;
```

The algorithm COMPATIBLE(v, R) determines the compatibility of a column v with a folding region R, where R is upper folding or lower folding region. If v is disjoint from every element in R, COMPATIBLE returns true, otherwise it returns false. For a given folding region R, if v is compatible with R, v can be placed in the opposite folding region.

```
    Algorithm COMPATIBLE($v$, $R$)
begin
    if ($\|R\| = M$) then
        return false
    else
    begin
        for each $u$ in $R$ do
            if $CM(v, u) = 0$ then
                return false;
    end
    return true;
end;
```

CHECKFOLDING(V, U, L) tries to place the elements in V into U and L. It first selects the column with least freedom in V. If this element can be put in neither U nor L, the folding condition fails, and CHECKFOLDING returns false. Otherwise, this element is put into the corresponding folding region. Every element in V is tested using CHECKFOLDING.

```
    Algorithm CHECKFOLDING($V$, $U$, $L$)
begin
    if ($V = 0$) then
        return true;
    else
    begin
        $v \leftarrow$ VSELECT($V$);
        if not COMPATIBLE($v$, $U$) and not COMPATIBLE
                ($v$, $L$) then
            return false;
        else
        begin
            if COMPATIBLE($v$, $L$) then
                CHECKFOLDING($V - v$, $U \cup v$, $L$);
            if COMPATIBLE($v$, $U$) then
                CHECKFOLDING($V - v$, $U$, $L \cup v$);
        end;
    end;
end;
```

V. EXPERIMENT RESULTS

The algorithms have been implemented in C programming language on a SUN SPARC-station 2. We have used our algorithm to find bipartite folding for a number of PLA's of varying sizes. We have not analyzed the complexity of the algorithm but its efficiency is evident from the results presented in this section.

Table II summarizes the results of 30 large PLA's from the list of 56 PLA's given in [1]. The definitions of the column headings are as follows,

N_i, N_o, and N_p: number of inputs, outputs, and product terms, respectively.

f_{aM} and f_{oM}: the upper bound on folding pairs in the AND plane and in the OR plane, respectively. Note that f_{aM} and f_{oM} are not the optimum solution for folding. They indicate the maximum possible number of folding pairs.

TABLE II
COMPARISON OF FOLDED AREA

name	N_i	N_o	N_p	f_{aM}	f_a	f_{oM}	f_o	RA(%)	time	RRA(%)
apla	10	12	134	0	0	6	6	90.8	0.183	1.00
bc0	26	11	479	7	7	0	0	81.2	0.650	–
bca	26	46	301	10	10	18	10	73.8	26.650	–
bcb	26	39	299	10	10	13	8	73.8	60.750	1.09
bcc	26	45	245	10	10	16	10	74.1	60.267	1.11
bcd	23	38	243	10	10	15	8	72.1	20.217	1.10
chkn	29	7	153	5	5	3	3	87.3	0.183	1.04
cps	24	109	654	3	3	54	54	64.4	58.770	–
dk48	15	17	148	0	0	8	8	90.9	0.133	1.00
exep	30	63	175	3	3	31	31	75.7	7.583	1.00
gary	15	11	214	2	2	3	2	91.7	0.200	0.98
in0	15	11	138	2	2	1	1	96.1	0.133	1.00
in2	19	10	137	5	4	3	2	87.7	0.133	1.00
in3	35	29	75	11	11	12	11	77.4	9.617	1.00
in4	32	20	234	10	10	8	7	73.3	3.367	1.07
in5	24	14	62	7	7	4	4	84.6	0.150	1.04
in6	33	23	54	12	11	10	9	78.7	1.767	1.18
in7	26	10	84	7	7	4	4	82.2	0.100	1.04
jbp	36	57	166	16	15	28	28	60.8	55.783	–
misg	56	23	75	28	28	11	11	59.3	4.783	1.00
mish	94	43	91	47	47	21	21	57.1	32.917	1.00
opa	17	69	342	2	2	34	34	67.4	9.883	–
ti	47	72	241	20	18	34	28	65.7	46.733	1.09
vg2	25	8	110	4	4	4	4	88.8	0.117	1.19
x1dn	27	6	112	4	4	3	3	90.9	0.117	0.90
x2dn	82	56	112	41	40	28	28	57.0	22.650	1.00
x6dn	39	5	121	16	14	0	0	72.9	43.667	1.00
x7dn	66	15	622	28	27	7	7	61.3	4.683	–
x9dn	27	7	120	5	4	3	3	90.7	0.150	0.90
pla	23	19	52	9	9	9	9	71.8	0.867	–

f_a and f_o: number of folding pairs found by BIFOLDING in the AND plane and the OR plane, respectively.
RA: relative area for the folded PLA,
time: the CPU time in second.
RRA: the relative area with respect to arbitrary column folding.

Note that the RA's are computed from the "exact" area of the PLA, which includes the input decoders, pull-ups, and output buffers. The layout of a bipartite folded PLA is generated by a CMOS PLA generator developed by us. The choice of 30 PLA's out of 56 PLA's is based on the area reduction (RA) of folding. As pointed out in Section II, placing the input decoders and output buffers on top of small PLA may increase the area in spite of folding. The PLA's with increased area after folding are not listed in Table II. Besides, some PLA's among the 56 PLA's are too small such that the searching of the solution can be carried exhaustively. These small PLA's are also not considered in Table II.

We notice from Table II that for most of the PLA's $f_a = f_{aM}$ and $f_o = f_{oM}$. Thus, we conclude that BIFOLDING provides optimal solution in most cases.

From the results given in Tables I and II, it is evident that the areas of PLA's obtained by BIFOLDING are comparable to those obtained by simple column folding. For many of the large sparse PLA's, the folding pairs found by bipartite folding are the same as found by simple column folding. A dash (–) in the last column indicates that the simple column folding program was either unable to provide an answer or could not be run due to the large size of the PLA.

TABLE III
COMPARISON OF EXECUTION TIME WITH PLEASURE

name	PLA			Pleasure		BiFolding		Time	
	in	out	pr	in	out	in	out	Pleasure	BiFolding
apla	10	12	134	0	6	0	6	25.2	2.233
bc0	26	11	479	-	-	7	0	-	16.467
bca	26	46	301	-	-	10	10	-	26.650
bcb	26	39	299	10	14	10	8	91.3	60.750
bcc	26	45	245	10	17	10	10	79.7	60.267
bcd	23	38	243	10	16	10	8	67.9	45.217
chkn	29	7	153	6	3	5	3	31.6	4.133
cps	24	109	654	-	-	3	54	-	58.770
dk48	15	17	148	0	8	0	8	25.8	2.917
exep	30	63	175	3	31	3	31	54.4	7.583
gary	15	11	214	1	3	2	2	39.4	4.305
in0	15	11	138	2	1	2	1	27.2	2.850
in2	19	10	137	4	2	4	1	26.7	4.517
in3	35	29	75	11	11	11	11	21.0	8.133
in4	32	20	234	11	9	7	7	74.6	0.867
in5	24	14	62	8	4	6	4	15.3	15.250
in6	33	23	54	16	9	11	9	15.8	1.767
in7	26	10	84	8	4	7	4	17.3	31.767
jbp	36	57	166	12	28	15	28	44.0	55.783
misg	56	23	75	28	11	28	11	18.3	25.483
mish	94	43	91	47	21	47	21	27.0	32.917
opa	17	69	342	-	-	2	34	-	9.883
ti	47	72	241	19	35	18	28	98.8	46.733
vg2	25	8	110	0	4	4	4	22.4	19.250
x1dn	27	6	112	1	3	4	3	22.6	81.800
x2dn	82	56	112	40	28	40	28	34.5	53.067
x6dn	39	5	121	14	0	14	0	28.1	7.233
x7dn	66	15	622	-	-	27	7	-	4.683
x9dn	27	7	120	1	3	4	3	25.3	72.800
pla	23	19	52	9	8	9	9	10.5	0.867

The CPU time in seconds is also shown in Table II. For most of the PLA's, the CPU time is less than 1 s. The reasons that BIFOLDING is fast are,

1) The folding constrains are considered before searching. Hence, no time is spent on those columns which cannot be folded.
2) The heuristics in BIFOLDING select those columns with maximum freedom and then search the candidates with minimum freedom. This increases the possibility of finding a folding while keeping the search space alternatives as few as possible.

In the last row of Table II, "pla" is the PLA that appears in the paper by Hachtel *et al.* [7]. The PLA has 23 inputs, 19 outputs, 52 product terms, and 182 of the row-column intersections are personalized. The BIFOLDING was able to fold 18 columns, 9 in AND plane and 9 in OR plane. This implementation yields 71.8% area reduction and required only 0.867 s of CPU time. For physical consistency, the input columns have been folded with input columns and output columns with output columns. This has been easily achieved by placing additional constrains in *CM*. Hachtel's algorithm only folded 17 pairs, 9 and AND plane and 8 in OR plane,

To compare the speed between our algorithm and "pleasure," BIFOLDING was also ported onto a VAX 11/750 machine, where both "pleasure" and BIFOLDING could run. Table III shows the folding results and the CPU time of BIFOLDING and "pleasure." The last 2 columns show the CPU time in seconds for "pleasure" and BI-

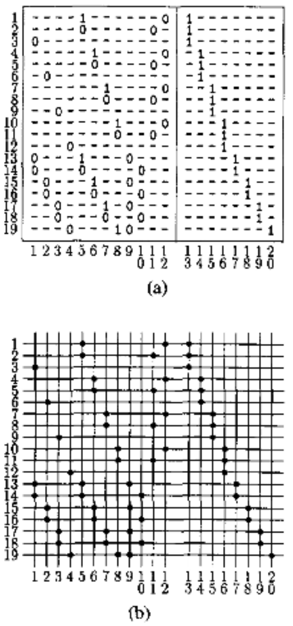

Fig. 5. A PLA "alu1." (a) Personality matrix. (b) PLA implementation.

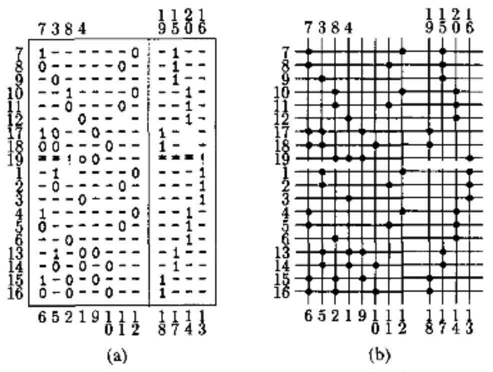

Fig. 6. The bipartite folding result of "alu1." (a) Personality matrix. (b) PLA implementation.

FOLDING, respectively. This table substantiates the claims made about the speed of BIFOLDING.

To show the folding result of BIFOLDING, a small PLA "alu1" which is listed in the 56 PLA's in Table I, is used to demonstrate.

Fig. 5(a) show the personality matrix and Fig. 5(b) shows the PLA implementation of "alu1." "alu1" has 12 inputs, 8 outputs, and 19 product terms. The size of "alu1" is 608. In Fig. 5(a), a "1" ("0") in the AND plane shows the presence of the corresponding uncomplemented (complemented) input on a product term. Similarly, a "1" in the OR plane shows the presence of the corresponding product term in the output. The reason to choose "alu1," instead of those PLA's in Table II, is that its size is small and there are folding pairs existed in it.

Fig. 6 shows the "alu1" after folding. There are 4 folding pairs in the AND plane and 4 folding pairs in the OR plane. In the AND plane, for physical consistency, the input columns have been folded with input columns and output columns with output columns. A "!" is a normal contact to uncomplemented input and split below, a "o" is a normal contact to complemented input and split below, and a "=" means no contact but split below. Similarly, a "!" in OR plane is a normal contact to output and split below, and a "=" means no contact but split below. The size of the folded PLA is 380. This implementation yields 37.5% area reduction.

VI. CONCLUSION

We proposed an efficient PLA folding algorithm applicable for column bipartite folding, based on matrix representation. The compatibility of the column pairs is found and stored in a compatibility matrix. This step discards "pairs" that cannot be in the final solution. Theorems are proved and invoked during this phase that help limit the search space. The BIFOLDING algorithm proposed in the paper makes use of heuristics to guide the search. The algorithm proposed in this paper yields nearly optimal results for almost all examples and in certain cases bipartite folded PLA's provide a better solution than arbitrary simple column folding as obtained by "pleasure." Generally, our algorithm provides PLA's that have areas comparable to single column folded PLA's but is much faster in providing the solution.

We have also outlined some of the advantages of bipartite folding, many of these ideas, especially those relating to testability have been incorporated in a program which generates testable PLA.

ACKNOWLEDGMENT

We are thankful to the anonymous referees for their constructive criticism. Their careful review and valuable comments of the manuscript of this paper greatly improved the quality of presentation.

REFERENCES

[1] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Boston, MA: Kluwer Academic, 1984.

[2] J. R. Egan and C. L. Liu, "Bipartite folding and partitioning of a PLA," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, no. 3, pp. 191–199, July 1984.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[4] W. Glass, "A depth-first branch and bound algorithm for optimal PLA folding," in *Proc. 19th Design Automat. Conf.*, June 1982, pp. 133–140.

[5] J. R. Griggs, "Lower bounds on the independence number in terms of degrees," *J. Combinatorial Theory*, vol. 8, no. 34, pp. 22–38, 1983.

[6] G. D. Hachtel, A. R. Newton, and A. Sangiovanni-Vincentelli, "Some results in optimal PLA folding," in *Proc. Int. Conf. Circuits and Computers*, 1980, pp. 1023–1027.

[7] ——, "An algorithm for optimal PLA folding," *IEEE Trans. Computer-Aided Design Integrated Circuits Syst.*, pp. 63–76, 1982.

[8] ——, "Techniques for programmable logic array folding," in *Proc. 19th Design Automat. Conf.*, June 1982, pp. 147–155.

[9] Y. C. Hsu, Y. L. Lin, H. C. Hsieh, and T. H. Chao, "Combining

logic minimization and folding for PLA's," *IEEE Trans. Comput.*, vol. 40, no. 6, pp. 706–713, 1991.

[10] S. Y. Hwang, R. W. Dutton, and T. Blank, "A best-first search algorithm for optimal PLA folding," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, no. 3, pp. 433–442, July 1986.

[11] D. S. Johnson, "Approximation algorithms for combinational problems," *J. Comput. Syst.*, pp. 256–278, 1974.

[12] J. E. Lecky, O. J. Murphy, and R. G. Absher, "Graph theoretic algorithms for the PLA folding problems," *IEEE Trans. Computer-Aided Design*, vol. CAD-8, pp. 1014–1021, 1990.

[13] J. L. Lewandowski and C. L. Liu, "A branch and bound algorithm for optimal PLA folding," in *Proc. 21st Design Automat. Conf.*, June 1984, pp. 426–433.

[14] M. Luby, U. Varzirni, V. Varzirni, and A. Sangiovanni-Vincentelli, "Some theoretical results on the optimal PLA folding problem," in *Proc. 1982 Int. Symp. Circuit Syst.*, Oct. 1982, pp. 185–170.

[15] B. Mayo and J. Ousterhout, "Pictures with parentheses: Combining graphics and procedures in a VLSI layout tool," in *Proc. 20th Design Automat. Conf.*, June 1983, pp. 270–276.

[16] R. Müller and D. Wagner, "α-vertex separator is np-hard even for 3-regular graphs," *Computing*, vol. 4, pp. 343–353, 1991.

[17] G. D. Micheli and A. Sangiovanni-Vincentelli, "PLEASURE: A computer program for simple/multiple constrained/unconstrained folding of programmable logic arrays," in *Proc. 20th Design Automat. Conf.*, June 1983, pp. 27–29.

[18] A. Mukherjee, *Introduction to NMOS and CMOS VLSI Systems Design*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[19] R. Treuer, H. Fujiwara, and V. K. Agrawal, "Implementing a built-in self-test PLA design," *IEEE Design Test Comput.* Apr. 1985, pp. 37–48.

[20] J. D. Ullman, *Computational Aspects of VLSI*. Rockville, MD: Computer Science Press: 1984.

[21] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design—A Systems Perspective*. Reading, MA: Addison-Wesley, 1984.

[22] D. F. Wong, H. W. Leong, and C. L. Liu, "Multiple PLA folding by the method of simulated annealing," in *Proc. Integrated Circuits Conf.*, May 1986, pp. 351–355.

Chun-Yeh Liu received the B.S. degree from the National Tsing-Hua University, Taiwan, and M.S. degree in electrical and computer engineering from the University of Wisconsin–Madison.

He is a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of Wisconsin–Madison where he is working on the design of testable finite state machines. His research interests are CAD tools development with emphasis on testing, VLSI design and computer architecture. He has worked at Intel Corporation where he was involved in circuit design and performance verification.

Kewal K. Saluja (S'70–M'73–SM'89) received the B.E. degree from the University of Roorkee, India, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Iowa.

He is a Professor in the Department of Electrical and Computer Engineering, University of Wisconsin–Madison where he teaches logic design, computer architecture, microprocessor based systems, VLSI design and testing. Previously he was at the University of Newcastle, Australia. He has also held visiting and consulting positions at such institutions as the University of Southern California, University of Iowa, Hiroshima University. His research interests include design for testability, fault tolerant computing, VLSI design and computer architecture. He is the Associate Editor for the letters sections of the *Journal of Electronic Testing: Theory and Applications (JETTA)*.