View metadata, citation and similar papers at core.ac.uk brought to you by CORE

provided by Bulgarian Digital Mathematics Library at IMI-BAS

AN ALGORITHM FOR OPTIMAL BIPARTITE PLA FOLDING

Liudmila Cheremisinova

Abstract. This paper presents some results of PLA area optimizing by means of its column and row folding. A more restricted type of PLA simple folding is considered. It is introduced by Egan and Liu and called as bipartite folding. An efficient approach is presented which allows finding an optimal bipartite folding without exhaustive computational efforts.

Key Words: Programmable Logic Array, area optimization, PLA folding, bipartite folding

Introduction

Programmable Logic Array (PLA) is widespread used hardware form for the structured design of digital VLSI systems due to the regularity of its structure. PLA layout design is easily automated because of its direct correspondence with PLA personality matrix. The price paid for the structural regularity is that much PLA area is unused because a large percentage of the row-column intersections are not personalized. Several techniques have been proposed for reducing the area required.

Two approaches are usually used to reduce the area occupied by the PLA: logic minimization that provides logic expressions with minimal number of products and topological minimization reclaiming unused space. The proposed paper deals with the problem of topological optimizing PLA area by means of its folding [1 – 5]. PLA folding allows reducing needed area without loss of regular structure of the PLA. There exist different types of PLA folding. They are based on merging several columns (and/or rows) of a PLA into a single column (row). The paper focuses on simple column (and/or row) folding that involves merging pairs of columns (and/or rows) into single columns (rows).

Folding a column of a PLA supposes to split that column into two segments so that two inputs or outputs may share the same column of the folded PLA. This means vertical lines are broken into an upper and a lower parts and two variables (pair of inputs or pair of outputs) don't need two lines but only two segments of the only line of the PLA. The task is to find such a permutation of PLA rows, which allows a maximum set of column pairs to be implemented on segments of single lines of the folded PLA.

In [3, 4, 5] a special type of simple column and row PLA folding is considered, in which all of the breaks of the columns (or rows) occur at the same level (Fig. 1, 2). Such a case is referred to as bipartite folding. The single break level of bipartite folding allows speaking of an upper and lower folding regions, which contain the segments of those folded columns that are correspondingly above and below the breaks.

While bipartite folding may theoretically be only 25 percent as effective as regular simple folding for PLA's with column type constraints, this class of folding approaches the effectiveness of column simple folding for sparse PLA's. Some justifications for this approach are offered [3, 4], the most important of them are 1) the folded columns (rows) entering from the top (left) of PLA can be ordered independently of the folded columns entering from the bottom (right) of the PLA, that simplifies the routing signals; 2) the same algorithm can be applied for the row folding a previously column folded PLA, that simplifies subsequent PLA row folding; 3) a bipartite folded PLA allows to use much less additional area required for inclusion of testability features. The bipartite folding can be used to partition a large PLA into smaller PLA's, i.e. bipartite folding can be considered as a special type of PLA decomposition too.

In this paper a new bipartite PLA folding technique is presented. It is based on transformations of a Boolean column disjoint matrix that specifies the relation to be disjoint on the column set. That allows the PLA bipartite folding problem to be treated as a maximum unit minor problem. Before searching for a desired maximal unit minor some procedures of the column disjoint matrix reduction are made that allow pruning some rows and columns. Some of suggested results could reduce the search space for algorithm from [4], the other allow yielding optimal solutions.

Definitions

The overall combinational PLA is a standard two level NOR-NOR structure. Vertical lines of a standard combinational PLA are assigned with the input variables (and their complements) and output variables. Inputs run vertically through the first part of a PLA matrix, called as the PLA AND plane. It generates signals on its rows, which are used as inputs to the second part of a PLA matrix called as the PLA OR plane.

An example of the PLA AND plane (that is the example PLA from [4]) is shown in Fig.1. In this figure, columns are associated with complemented and uncomplemented inputs. Each horizontal line of the PLA carries a product term. A dot means placing a transistor on crosspoint of vertical and horizontal lines. This PLA AND plane will be used further throughout the paper. Here the OR plane is not shown but either AND plane or both AND and OR planes together can be described in symbolic form by a Boolean matrix. And the only difference is that inputs can share the columns with inputs only and outputs share the columns with outputs.

The area of a PLA is proportional to the total number of its columns times the number of rows. The area occupied by the PLA AND plane in Fig.1 is 273.

Before we formulate a mathematically tractable definition of PLA folding problem and its solution we have to give some definitions.

Each PLA column c_i implies the set $R(c_i)$ of rows, which are populated on it. Any two columns c_i and c_j are *disjoint* if $R(c_i) \cap R(c_j) = \varnothing$. A pair of disjoint columns is defined to be a column *folding pair*, and only those two columns of a PLA can be folded together.

For example PLA, columns c_1 and c_6 are disjoint, since $R(c_1) \cap R(c_6) = \{r_{12}, r_{21}\} \cap \{r_{14}, r_{16}, r_{17}\} = \varnothing$. But $R(c_5) \cap R(c_6) = \{r_{14}, r_{17}\}$, hence columns c_5 and c_6 do not generate folding pair.

The square Boolean matrix that depicts when the PLA columns are disjoint will be called a *column disjoint matrix D*. This matrix has as many rows and columns as the number of the PLA columns. The element $d^j_i \in D$ is 1 if columns c_i and c_j are disjoint, otherwise $d^j_i = 0$.

The following column disjoint matrix corresponds to the example PLA AND plane of Fig. 1:

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

The relation to be disjoint on the column set is symmetric and irreflexive, so the column disjoint matrix D is symmetric too and has all 0's on the leading diagonal.

Bipartite folding is a column or row folding in which all of the breaks of the columns (or rows) occur at the same level (Fig. 2). The single break level of bipartite folding allows speaking of an upper and lower folding regions, which contain the segments of those folded columns that are correspondingly above and below the breaks.

In bipartite folding all column breaks used to facilitate folding are made between the same two rows. Thus all columns, which are folded and placed at the top of the PLA must be disjoint from all columns folded and placed to the bottom of the PLA. Let the PLA columns of the set C^u belong to the upper folding region and the columns of the set C^l belong to the lower one.

Thus the *necessary and sufficient condition*, the pair C^u, C^l of the column sets involves bipartite folding, is the columns of C^u to be disjoint from each column of C^l (and symmetrically vice versa). So, the rows of the column disjoint matrix D corresponding to the PLA columns of the set C^l have 1's in all columns of the set C^u.

Such a pair of equinumerous sets is called a *bipartite folding pair of sets*. The cardinality of these sets defines the size of the bipartite folding pair of sets and the size of induced PLA bipartite folding.

It is clear that a pair C^u, C^l of column sets contains all information needed to fold a PLA i.e. it specifies the pairs of columns to be folded and their relative position (top or bottom).

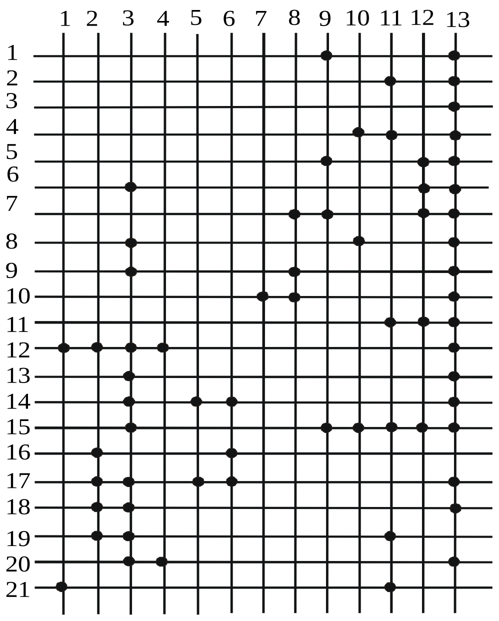
Fig.1. An example PLA

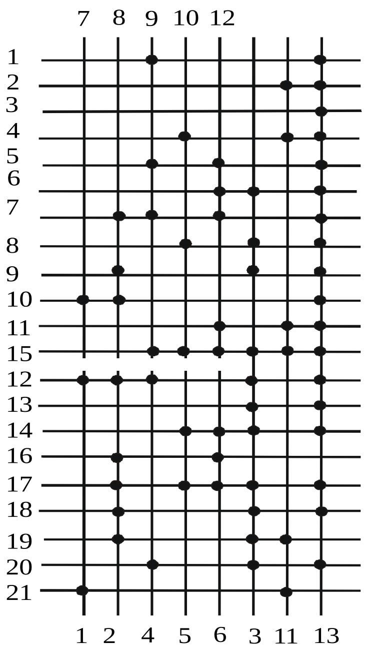
Fig.2. A bipartite folded example PLA

Transforming bipartite PLA folding into unit minor identification

The column disjoint submatrix, their columns and rows correspond to the PLA columns of the set C^u and the rows correspond to the PLA columns of the set C^l, is called a folding matrix ***F*** if the pair C^u, C^l involves bipartite folding. Such a matrix satisfies the following properties:

– it is square, $m \times m$ matrix, where m is the number of folding pairs of PLA columns;

– columns of the matrix correspond to the elements of the upper folding region, and rows of the matrix correspond to the elements of the lower folding region;

– $C^u \cap C^l = \varnothing$;

– all matrix elements are 1's.

Thus a folding matrix is a unit square minor of the column disjoint matrix of a PLA. It implies from this that bipartite folding of size m exists if and only if there is a folding matrix of size m.

A folding matrix ***F*** of the bipartite folded PLA in Fig.2 is specified by the pair $C^u = \{c_7, c_8, c_9, c_{10}, c_{12}\}$ and $C^l = \{c_1, c_2, c_4, c_5, c_6\}$ and is highlighted above in the column disjoint matrix ***D*** in thick print.

Identifying the optimal column bipartite folding set is to find a bipartite folding pair of sets of the greatest size, in other words the task is to find unit square minor of the PLA column disjoint matrix ***D*** with the greatest number of rows (or columns). That minor will be folding matrix required.

It is evident that the greatest size m of the folding matrix does not exceed $n/2$, where n is the number of PLA columns. The example PLA (Fig. 1) has 13 columns, so it allows for at most 6 pairs of folded columns.

Let the *weight* of the column c_i (row r_i) of the column disjoint matrix ***D*** be the number of 1's in it. The last row of the matrix ***D*** given above shows weights of its columns.

So, if a PLA permits bipartite folding pair of sets of size m then there is at least $2m$ columns having weights greater than or equal to m in the column disjoint matrix [4]. This statement follows from folding matrix definition. In other words, if looking for a bipartite folding of size m all candidate columns and rows of the column disjoint matrix should have weights greater than or equal to m.

The example PLA (Fig. 1) has only 9 columns with weights at least 6, so there is no folding matrix of size 6. But the example PLA has 11 columns with weights greater than or equal to 5. Thus a folding matrix is upper-bounded by 5.

Reducing the search space for bipartite folding

The optimal bipartite folding problem was shown to be NP-complete [3]. The classification of a problem as NP-complete is not sufficient reason to develop only heuristic optimizing methods. In many cases it is interesting to get just an optimal solution. Some results of reduction of exhaustive computational efforts on the search for the optimal PLA bipartite folding have been described.

The proposed PLA bipartite folding algorithm is organized such a manner to give an optimal solution during an exhaustive search but to allow finding “good” solution at the first its iteration. This algorithm starts from the pruned column disjoint matrix *D*. And then after each algorithm step pruning techniques are made.

The evident way to find a bipartite folding pair of sets of the greatest size is to find a unit minor of the column disjoint matrix *D* of the greatest size. PLA columns that correspond to the unit minor columns can be referred to the upper folding region, and the PLA columns that correspond to the unit minor rows can be referred to the lower folding region. These two PLA column sets C^u and C^l are independent. Before the algorithm functioning the set C^u contains all the PLA columns and the set C^l is empty. It should be noted that the same PLA column is never placed into the unit minor row and column sets simultaneously (both in C^u and in C^l) owing to the column disjoint matrix *D* has 0’s on the leading diagonal. In the process of the algorithm functioning, some PLA columns will be referred to C^l and eliminated from C^u. Thus corresponding rows (or columns) of the PLA column disjoint matrix *D* should be eliminated too.

Before seeking for the minors we can reduce search space, i.e. reduce the matrix *D*. First, for a dense PLA some columns have small weights.

1. It is trivial that columns (and rows) with weights equalled to 0 should be eliminated [4]. They are useless for folding. As well the columns (and rows) with weights equalled to 1 are superfluous: the bipartite folding pair of sets of size 1 can be found trivially.

For example, there exist two superfluous columns (and rows) in the example column disjoint matrix *D*: 3 and 13, having weights 1 and 0.

When the column disjoint matrix *D* has at least 2*m* columns having weights greater than or equal to *m* there can exist a folding matrix of size *m*. It can be found after reducing search space by means of elimination of all columns with weights less than *m*. But if the search fails we will have to decrease the value of *m* and repeat the search as it is proposed in [4]. So, we should choose such a value of *m* that there would be substantially greater than 2*m* columns having weights greater than or equal to *m* to refrain from repetition of the search for folding matrix.

2. The identical rows of matrix *D* should be united. These rows are not disjoint. Otherwise they would not be equal. And they are in the same relation with the rest PLA columns. So they will be in the same folding region in any permissible solution of the folding problem.

After removing superfluous columns (and rows) from the example column disjoint matrix *D* there are three groups of identical rows (and columns): 5 and 6; 7 and 8; 9, 10 and 12. Thus they can be united as it is shown below.

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

Here weights take into account repetition factor of the corresponding rows.

3. Further, the PLA column (and row) that is disjoint from every other can be added both to the set C^u, identifying the upper folding region, and to the set C^l, identifying the lower folding region. The column (and row) of the matrix *D*, corresponding to such a PLA column, has all 1’s but the only 0 (on the leading diagonal). The column weight is equal to n–1. So the column and the row can participate in no process of minor seeking and can be eliminated from the matrix *D*. After desired minor has been found the column can be included either in C^u or in C^l, depending on what of them has the less cardinality. Thus we should seek not necessarily a square unit minor but a unit minor of the greatest area.

For column disjoint matrix *D*, depicted above, the group of rows (columns) 7 and 8 is disjoint from all others. So that group can be eliminated from *D*. Having 1, 2, 3 in mind we get the following reduced column disjoint matrix *D*:

	1	2	4	5	6	9	10	11	12
1	0	0	0	1	1	1	1	0	1
2	0	0	0	0	0	1	1	0	1
4	0	0	0	1	1	1	1	1	1
5,6	1	0	1	0	0	1	1	1	1
9,10,12	1	1	1	1	1	0	0	0	0
11	0	0	1	1	1	0	0	0	0
weight	5	3	6	6	6	5	5	3	5

For dense PLAs, most columns have small weights, resulting in a small size of the column disjoint matrix. For sparse PLAs, most columns have large weights, resulting in a large number of identical rows (and columns). Thus in both cases it is expected that reduced column disjoint matrix *D* to be solved will have moderate size.

The search algorithm

The maximal unit minor is sought in the process of traversing the search tree in which each node represents a pair of two sets C^u and C^l of unit minor columns and rows. The rows for including to the set C^l are chosen from the set *R* of row-candidates corresponding to the node. C^u includes all columns that correspond to intersection of rows from the set C^l. At the first step the set C^u contains all the columns of the reduced column disjoint matrix *D*, the set C^l of minor rows is empty and the set of row-candidates for C^l contains all rows of the reduced column disjoint matrix *D*. The size of a folding matrix, associated with the unit minor described by the sets C^u and C^l, is the lesser of the cardinalities of these two sets.

The transition to a son of any node of the search tree consists in adding one more new row-candidate from *R* to the set C^l, thus decreasing, in general case the number of minor columns. The optimal solution is found during an exhaustive search of the tree reduced by means of some pruning techniques.

We are going to build successively one by one the unit minors of the Boolean matrix *D*. After getting a new better solution we store it and its size *k*. At each next step we need to consider only those minors of greater size than that early-found ($i > k$). If we get a new better solution we can reduce the Boolean matrix *D* at the sacrifice of elimination of all its columns and rows with weights less than or equal *k*.

The search tree. The set of all possible unit minors is organized in the form of a search tree in which each node represents two sets: C^u and C^l, presenting columns and rows of the corresponding unit minor and the set *R* of row-candidates for including in C^l. The sons of a node represent those minors that contain one additional row in C^l. By traversing this search tree all possible unit minors of increasing size should be systematically examined. The process continues until a solution is found or all possible choices have exhausted.

Pruning the search tree. Traversal of the search tree is done in a depth-first manner, backtracking from a node whenever the sub-tree rooted by a node has been completely explored. The size of such search tree grows exponentially. However the suggested branch and bound algorithm restricts the exploration to within only a small subset of the nodes in the tree by means of pruning the search tree. A sub-tree can be pruned only if the algorithm can determine that this sub-tree contains no unit minor of size greater than that has been found so far.

A node of the search tree is called viable if the size of the corresponding unit minor is greater than the size of a minor found when traversing the tree before getting to this node. Only viable nodes need to be examined during an exhaustive search of the tree. When reaching a viable node, the rows with weights, less than or equal to the size of the unit minor corresponding to that node, should not be considered. This allows reducing the search space. If in any step a unit minor of the greatest possible size *m* is found then it is the desired maximal unit minor and the algorithm ends.

Backtracking is made if not viable node is reached. In that case a row, last included in C^l, is discarded and another one, not considered earlier, is selected.

Heuristic for choosing the traversing path of the tree. The algorithm suggested employs a simple heuristic to determine an order in which sub-trees, rooted in a node, are traversed so that a near optimal solution can be discovered quickly. Finding a near-optimal solution quickly is important since the sooner such a pair of sets C^u and C^l is found the sooner it can be used for pruning purposes.

The selection steps are important to produce a “good” solution the sooner the better, so to find a solution at the first branch of the search tree as close as possible to the maximum. It appears that justified strategy is greedy one. Therefore, a good heuristic is to select, at the first step, a row with maximum weight as candidate to be placed in the set C^l. Then a row of the matrix *D* is chosen that most intersects the set C^u: in other words it has the greatest number 1's in the columns of the set C^u.

Starting with reduced column disjoint matrix *D* depicted above, the algorithm finds maximal unit minor with C^u = {9, 10, 12} and C^l = {1, 2, 4, 5, 6}. It is highlighted in the matrix *D* in thick print. Having in mind that PLA columns 7 and 8 may be included either in C^u or C^l, we insert them in the set C^u with less cardinality. Thus the following folding matrix *F* is associated with found maximal unit minor:

	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

The resulting bipartite folded PLA is given in Fig. 2. The area occupied by the bipartite folded PLA is 168 instead of 273 (Fig. 1).

Additional chance to reduce the area of bipartite folded PLA

After completion of the algorithm for bipartite PLA folding we can get a unit minor that is defined by a pair of sets C^u and C^l of different cardinalities. In this case we have some extra (for the bipartite folding) PLA columns. That is another possibility to reduce the area of the PLA. Such extra members of C^u or C^l can be folded with any PLA columns that are disjoint from them. It is possible to find unit minors with row (or column) set containing only some of the extra PLA columns and with column (or row) set containing PLA columns not included in $C^u \cup C^l$.

An example of such a case of double folding is shown in Fig. 3 (that is example PLA from [4] with modified 13-th column) and Fig. 4. The area occupied by the double bipartite folded PLA is reduced to 147 instead of 168 (Fig. 2).

The following column disjoint matrix corresponds to the example PLA AND plane of Fig. 3:

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

There exists the only superfluous column (and row) in the above column disjoint matrix *D* – 3, having the weight 1. After its removing there are three groups of identical rows (and columns): 5 and 6; 7 and 8; 9, 10 and 12, they are united. Thus we have the following reduced column disjoint matrix *D*:

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

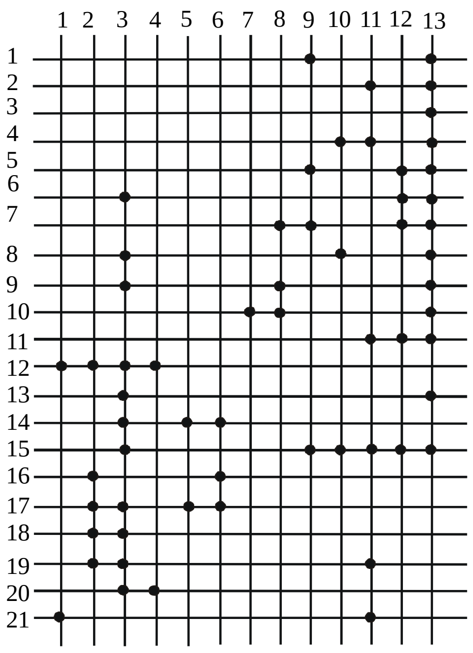
Fig. 3. The second example PLA

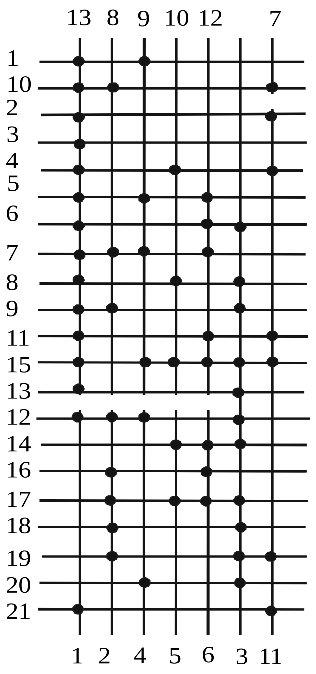
Fig. 4. A bipartite folded second example PLA

The bipartite folding (Fig. 4) is induced from the maximal unit minor with C^u = {7, 8, 9, 10, 12, 13} and C^l = {1, 2, 4, 5, 6}. It is highlighted in the above PLA column disjoint matrix *D* in thick print. Extra PLA columns 7, 8 of the set C^u are disjoint from the PLA column 11 constituting one of the possible unit minors of size 2.

Conclusion

In this paper a new bipartite folding technique is presented. Compared with the other bipartite folding methods the suggested method has the following new features. The problem of bipartite folding is reduced to a search for a maximal unit minor of a Boolean matrix. The method contains some new procedures of reduction of Boolean column disjoint matrix that allow reducing the search space before functioning the basic bipartite folding algorithm. The approach presented in the paper may lead to the optimal bipartite folding without much wasteful computation.

References

1. G.D. Hachtel, A.R. Newton and A.L. Sangiovanni-Vincentelli, “An Algorithm for optimal PLA Folding”, IEEE Trans. Computer-Aided Design of Integrated Circuit Syst., vol. CAD–1, no 2, pp. 63–77, 1982.
2. G. DeMicheli and A. Sangiovanni-Vincentelli, “Multiple Constrained Folding of Programmable Logic Arrays: Theory and Applications”, IEEE Trans. Computer-Aided Design, vol. CAD-2, no 3, pp. 151–167, 1983.
3. 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, 1984.
4. Chun-Yeh Liu and Kewal K. Saluja, “An efficient algorithm for bipartite PLA folding”, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 12, no 12, pp. 1839–1847, 1993.
5. J. E. Lecky, O.J. Murphy and R.G. Absher, “Graph theoretic algorithms for the PLA folding problem”, IEEE Trans. Computer-Aided Design, vol. 8, no 9, pp. 1014–1021, 1989.

Author information

Liudmila Cheremisinova, The United Institute of Informatics Problems of National Academy of Sciences of Belarus, Surganov str., 6, Minsk, 220012, Belarus, Tel.: (10-375-17) 284-20-76,
e-mail: cld@newman.bas-net.by