Smile: a computer program for partitioning of programmed logic arrays

Giovanni De Micheli and Mauro Santomauro*

This paper presents a new approach to optimal topological design of PLAs (programmed logic arrays). In particular we address the array partitioning problem and the implementation of partitioned arrays as block folded or parallel connected PLAs. We present a graph theoretic interpretation of the problem and an efficient heuristic algorithm. A computer program, Smile, is described and experimental results are reported.

design, interpretation, algorithm

PLAs (programmed logic arrays) are used extensively in the structured design of VLSI circuits[1]. Multiple output switching functions are conveniently implemented by PLAs[2,3], because they show a regular structure and can be effectively designed and optimized with the support of computer aids.

We consider here PLAs implementing sum-of-products switching functions with the following structure. The PLA consists of two adjacent arrays: the input array or AND plane and the output array or OR plane (Figure 1). Input signals and their complements run vertically in the AND plane, product terms run horizontally in both planes and outputs run vertically in the OR plane. Both arrays are personalized by the presence of active devices in positions corresponding to the 'cares' of the switching function. Note that in general PLAs are implemented by two NOR subarrays in nMOS and in cMOS technology, but this does not affect our analysis.

The design of PLAs involves several steps as shown in Figure 2[4]. Boolean equations are translated first into a set of two-level sum-of-products logical implicants. In general, this is followed by a logic minimization, in order to reduce the number of implicants and literals. Logic minimizers are effective tools for this task[5,6]. However most arrays are still very sparse: the number of 'cares' is much smaller than the number of 'don't cares'[7]. A straightforward physical implementation results in a significant waste of the silicon area not directly contributing to the implementation of the logic function. The wasted area reduces circuit yield and degrades the time performance of the PLA by introducing unnecessary parasitics.

The topological design aims to reduce the wasted area. This design step has been recently investigated by several authors. PLA folding is a powerful technique to accomplish this task[7-10]. The objective of folding is to determine a permutation of the rows (and/or columns) of the array which permits a maximal set of column pairs (and/or row pairs) to be implemented in the same column (row) of the logic array.

An alternative approach is block folding[11] which has been referred to also as bipartite folding[12] and as array segmentation[13,14]. Block folding aims to determine a permutation of the rows (and/or columns) of the array such that the columns (rows) can be partitioned into two sets and any pair of columns (rows) in different sets can be implemented in the same column (row) of the physical logic array (see Figures 3(*b*) and 4(*b*)).

PLA decomposition into parallel connected arrays has

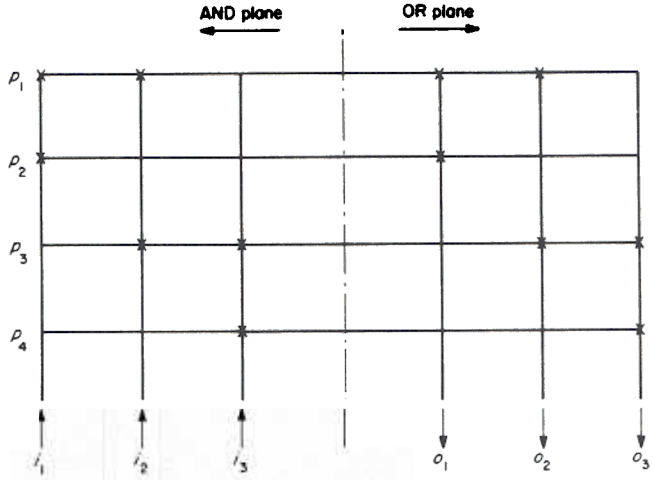

Figure 1. Example of a PLA structure

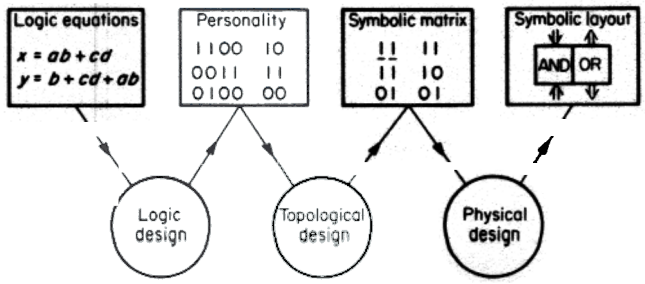

Figure 2. Computer-aided PLA design

Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, USA

This paper is based on research partially sponsored by IBM, DARPA grant # 25697 and Consiglio Nazionale delle Ricerche, Italy.

*On leave from Dipartimento di Elettronica, Politecnico di Milano, Italy. Partially supported by a NATO fellowship granted by CNR, Italy.

```
1          0000 0100100110110010
2          0001 0000100100010000
3          0010 1000000010010000
4          0011 0001011000001101
5          0100 0010100000110000
6          0101 0000010000001000
7          0110 0000100000110010
8          0111 0000000110100010
9          1000 1001011001000100
10         1001 1000011000000001
11         1010 1000001000100001
12         1011 1001011000100101
13         1100 1001001000001001
14         1101 1001001000000100
15         1110 1000010000001100
16         1111 0110000000110010

                              1111111
           1234 1234567890123456
```

Figure 3. TPM of a benchmark PLA (PLA 2) before partitioning

been investigated by Suwa[14]. A logic array is broken into several subarrays and the outputs of the subarrays are merged together.

We investigate in this paper a general framework for PLA optimal topological design based on array partitioning. Kang[11] proposed for the first time a heuristic algorithm for PLA partitioning. We present in this paper a partitioning algorithm based on a graph representation of the PLA structure. The algorithm takes advantage of array transformations based on logical operations to ease partitioning. Since partitioned arrays can be implemented as multiple block folded

```
BLOCK FOLDED OR PLANE

                    111112
                   58014780

*  3      0010     10000000
   4      0011     01110111
   6      0101     00100100
   9      1000     11111010
  10      1001     10110001
*  11     1010     10010001
*  12     1011     11110011
  13      1100     11010101
  14      1101     11010010
  15      1110     10100110
                   --------
   1      0000     10111111
   2      0001     00110010
   3      0010     00001010
   5      0100     01100110
   7      0110     00100111
   8      0111     00011101
  11      1010     00000100
  12      1011     00000100
  16      1111     11000111

                      11111
           1234     67923569

Partitioned PLA takes 71% of the original area
```

Figure 4. TPM of a benchmark PLA (PLA 2) after output-partitioning

or parallel connected arrays, our approach embodies the previous proposed implementations as special cases.

BASIC CONCEPTS AND DEFINITIONS

The topological description of a PLA is contained in the TPM (topological personality matrix) whose entries are 1 if the corresponding element in the PLA is a 'care', and 0 otherwise[8].

The TPM can be divided into two submatrices A and B related to AND plane (input subarray) and OR plane (output subarray) respectively. If the PLA has N inputs, M outputs and P products, the TPM has P rows and N+M columns (Figure 5).

We define the input (output) column set $I = \{i_1, i_2, \ldots, i_N\}$ ($O = \{o_1, o_2, \ldots, o_M\}$) the set of the first N (last M) columns of the TPM.

We define the product row set $P = \{p_1, p_2, \ldots, p_P\}$ the set of rows of the TPM. A product row p_j is split into two parts: p_j^A contains the first N entries of p_j and p_j^B the last ones.

We define the logical conjunction (disjunction) of two vectors x, y:

$$x \vee y \quad (x \wedge y) \tag{1}$$

the vector obtained by the component-wise conjunction (disjunction) of x and y. Logical conjunction (disjunction) of n vectors will be indicated as:

$$\vee_{i=1}^{n} x_i \quad (\wedge_i^n = \hat{1}\, x_i) \tag{2}$$

throughout the paper.

Two vectors x,y are independent (orthogonal) if $x \wedge y = O$, where O is the null vector. We denote by $x \perp y$ two independent vectors.

Two vector sets X, Y are independent if

$$x \perp y \qquad \forall x \in X \text{ and } \forall y \in Y \tag{3}$$

Logic array partitioning relies on determining independent sets of vectors in the TPM. A logic array is said to be input (output) partitionable if there exist input (output) column independent sets. An input (output) partitionable array has also independent sets of input (output) product row p_i^A (p_i^B). A logic array is said to be parallel partitionable (or simply partitionable) if there exist product row independent sets.

A parallel partitionable array is input and output partitionable, but the inverse is not true because input independent product row sets and output independent product rows sets can belong to different product row sets.

EQUIVALENT ARRAYS AND PARTITIONING

In general the TPMs of logic arrays do not have input and/or output independent sets of products rows and cannot be partitioned as they are. It is then necessary to

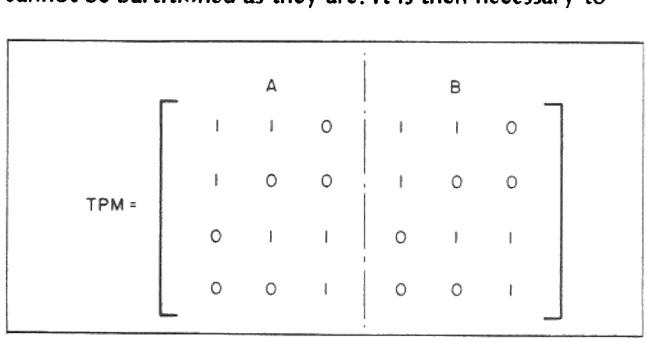

Figure 5. Topological personality matrix

transform an array into an equivalent one before partitioning it.

Two logic arrays are equivalent if they implement the same switching function. Equivalent arrays can be of different size and can be obtained by introducing redundant rows[15] and/or columns[10,13] or by rearranging the TPM of the array by a reshape[5] of the logic function.

We consider in this paper a general equivalence transformation based on row (column) augmentation. We define augmentation of an input, output or product, the substitution of the input, output column or product row with a set of input, output columns or product rows that gives an equivalent logic array. We now present rules to obtain equivalent arrays by augmentation:

- Rule 1: input column augmentation. The logic arrays defined by A,B and A′,B are equivalent if:
  - A′ is obtained from A by replacing an input column i_j with a column set $I_j = \{i_{j1}, i_{j2}, \ldots, i_{js}\}$ such that
    $$\vee_{k=1}^{s} i_{jk} = i_j \quad (4)$$
  - Input signals to columns in I_j correspond to input signal to column i_j.

  An input partitionable array can be obtained by a sequence of input column augmentations.
- Rule 2: output column augmentation. The logic array defined by A,B and A,B′ are equivalent if
  - B′ is obtained from B by replacing an output column o_j with a column set $O_j = \{o_{j1}, o_{j2}, \ldots, o_{js}\}$ such that:
    $$\vee_{k=1}^{s} o_{jk} = o_j \quad (5)$$
  - The output signal from column o_j corresponds to the logic conjunction of the output signals from the column in O_j.
- Rule 3: product row augmentation. The logic array defined by A,B and A′,B′ are equivalent if:
  - [A′|B′] is obtained from [A|B] by replacing a product row p_j with a row set $P_j = \{p_{j1}, p_{j2}, \ldots, p_{js}\}$ such that
    $$\vee_{k=1}^{s} p_{jk}^{B'} = p_j^{B} \quad (6)$$
    $$p_{jk}^{A'} = p_j^{A} \quad \forall k = 1, 2, \ldots, s \quad (7)$$

An output partitionable array can be obtained by a sequence of product row augmentations and a partitionable array by a sequence of product augmentations followed by a sequence of input augmentations.

It is clear that there are many different possible augmentations for a row or a column according to rules 1, 2 and 3. For optimal topological design it is convenient that augmented rows and columns keep the array as sparse as possible. Hence we require the augmented columns and the output part of the augmented product rows to be independent. Moreover optimal topological design based on array partitioning requires the determination of an optimal sequence of augmentations.

GRAPH INTERPRETATION OF THE PARTITIONING PROBLEM

A graph interpretation of the partitioning problem gives a pictorial representation of the connectivity of the array and is useful in understanding the underlying structure.

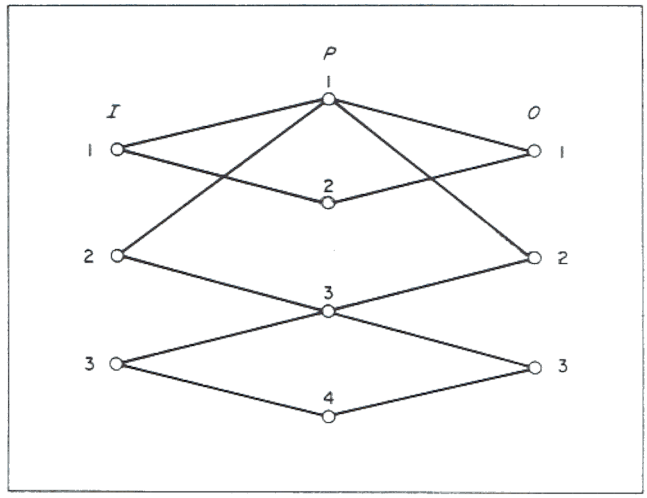

Figure 6. Graph G of the original PLA

We refer the reader to Lawler[16] for definitions of graph theory.

The AND plane (OR plane) of a PLA can be represented by a bipartite graph $G^A(I,P,E^A)(G^B(P,O,E^B))$ whose adjacency matrix is $\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$ ($\begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix}$). The whole logic array is therefore represented by the union of such graphs, ie the tripartite graph $G(I,P,O,E)$, where $E = E^A \cup E^B$ (Figure 6). The node sets I,P and O are in one-to-one correspondence with the PLA input column, product row and output column sets respectively.

In order to give an estimate of the silicon area taken by the PLA we define a function F_o on G as follows:

$$F_o = (a|I| + b|O|)|P| + c|I| + d|O| + e|P| \quad (8)$$

where coefficients a—e are parameters depending on the physical layout of the PLA. The first term takes into account the area of the array and the last three terms the area taken by the drivers, the output inverters and the loads.

We will consider now the input, output and parallel partitioning problem in that order.

Input partitioning

In this case we restrict our attention only on graph $G^A(I,P,E^A)$ because input partitioning does not affect the OR plane.

Let us consider first the trivial case in which set P is the disjoint union on n input independent sets $P_j, j = 1, 2, \ldots, n$. Because of independence, input columns are also partitioned into n disjoint sets I_j. As a consequence graph G^A is disconnected into subgraphs $G_j^A = (I_j, P_j, E_j^A)$, $j = 1, 2, \ldots, n$.

Each subgraph G_j^A represents a block of an input partitioned PLA. It is straightforward that in this case an input partitioned array has an area smaller than the original one.

However, in general, graph G is connected and the input array is not partitionable. A transformation of the input array into an equivalent input partitionable one is then required: this corresponds to transform graph G^A into an equivalent disconnected one. This goal can be achieved by an input node splitting which is the counterpart of the input augmentation. The procedure is shown in Figure 7 on a simple example.

- Input node 2 is split into two nodes 2′ and 2″ (column

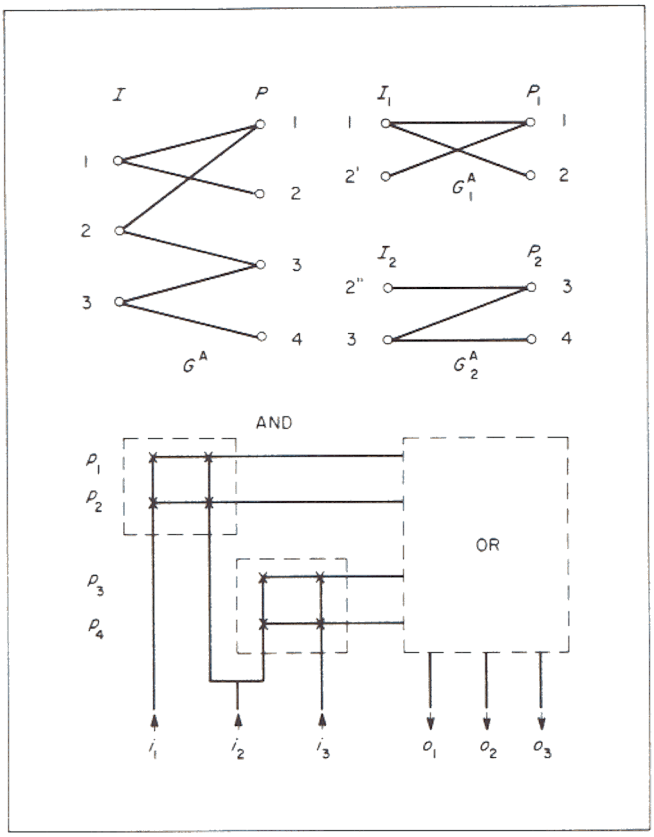

Figure 7. (Above left) graph G^A of the original PLA. (Above right) graphs G_1^A and G_2^A of the input-partitioned PLA. and (below) input-partitioned PLA

augmentation on the PLA) and the edges incident to 2 are now incident either to 2′ or to 2″. The equivalent augmented PLA is shown with its input partitioned implementation.

In general let us denote by $\Pi_n(E^A)$ a partition of the edge set E^A into n subsets $E_1^A, E_2^A, \ldots, E_n^A$. Let $G_j^A(I_j, P_j, E_j^A)$ be any subgraph induced by the partition where I_j and P_j are the sets of input and product nodes which are adjacent to edges in E_j^A. Because of input node splitting in general $|I| \leqslant \sum_{j=1}^{n} |I_j|$ while $|P| = \sum_{j=1}^{n} |P_j|$ (no product augmentation is allowed). Subgraphs G_j^A $j = 1, 2, \ldots, n$ correspond to the blocks of the input partitioned array. An estimate of the input partitioned array area is given by:

$$F^A = \sum_{j=1}^{n} |P_j|(a|I_j| + b|O|) + c \sum_{j=1}^{n} |I_j| + d|O| + e|P| + f\left(\sum_{j=1}^{n} |I_j| - |I|\right) \quad (9)$$

where the last term takes into account the overhead due to the routing of the augmented input columns.

We can now state the input partitioning optimization problem OP1 as follows:

- Problem OP1. Find a partition $\hat{\Pi}_n(E^A)$ such that:

$$F^A(\hat{\Pi}_n(E^A)) \leqslant F^A(\Pi_n(E^A)) \quad \forall \Pi_n(E^A) \text{ and } \forall n \quad (10)$$

$$P_j \cap P_k = \phi \quad \forall j, k = 1, 2, \ldots, n; \quad j \neq k \quad (11)$$

Note that the optimal solution may not be unique.

Output partitioning

In this case we restrict our attention to graph $G^B(P, O, E^B)$ since the input node set is not affected by output partitioning.

As stated above, output partitioning can be achieved by output column and/or product row augmentation. The procedure is shown in Figure 8 on a simple example.

- Product node 1 is split into two nodes 1′ and 1″ (product row augmentation) and the edges incident to 1 are now incident either to 1′ or to 1″. The equivalent augmented PLA is shown with its output partitioned implementation.

In general let us denote by $\Pi_m(E^B)$ a partition of the edge set E^B into m subsets $E_1^B, E_2^B, \ldots, E_m^B$. Let $G_j^B(P_j, O_j, E_j^B)$ be any subgraph induced by the partition where P_j and O_j are the sets of product and output nodes which are adjacent to edges in E_j^B. Because of output node splitting in general $|O| \leqslant \sum_{j=1}^{m} |O_j|$ and $|P| \leqslant \sum_{j=1}^{m} |P_j|$. Subgraphs G_j^B $j = 1, 2, \ldots, m$ correspond to the blocks of the output partitioned array.

An estimate of the output partitioned array area is given by:

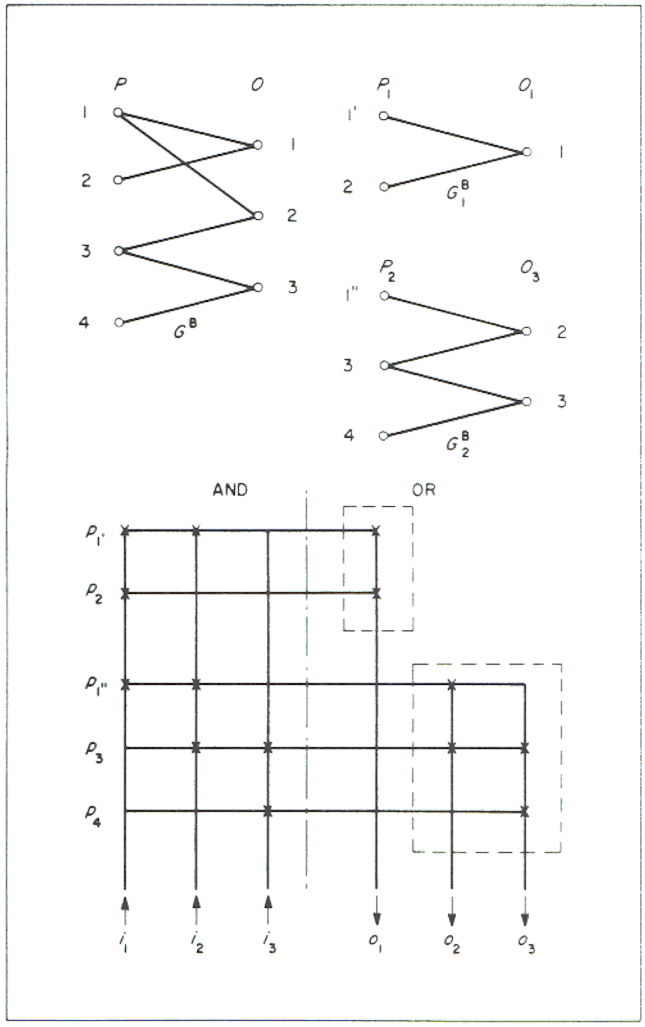

Figure 8. (Above left) graph G^B of the original PLA. (Above right) graphs G_1^B and G_2^B of the output-partitioned PLA. and (below) output-partitioned PLA

$$F^B = \sum_{j=1}^{m} |P_j|(a|I| + b|O_j|) + c|I| + d \sum_{j=1}^{m} |O_j| + e \sum_{j=1}^{m} |P_j|$$
$$+ g[(\sum_{j=1}^{m} |O_j|) - |O|] + h[(\sum_{j=1}^{m} |P_j|) - |P|] \quad (12)$$

where the last terms take into account the overhead due to the routing of the augmented output columns and product rows. We can now state the output partitioning optimization problem OP2 as follows:

- Problem OP2. Find a partition $\hat{\Pi}_m(E^B)$ such that:

$$F^B(\hat{\Pi}_m(E^B)) \leqslant F^B(\Pi_m(E^B))$$
$$\forall \Pi_m(E^B) \text{ and } \forall m \quad (13)$$

Note that the optimal solution may not be unique.

If only output column augmentations are allowed, the last term in equation (12) is equal to zero ($|P| = \sum_{j=1}^{m} |P_j|$) and then F^B can be obtained from F^A by interchanging I with O. In this case the output partitioning is exactly the 'dual' of the input partitioning. The problem OP2 is then obtained from the problem OP1 by adding the constraint equation (11) to equation (13).

Parallel partitioning

For this problem we require a graph representation of the whole logic array by means of $G(I,P,O,E)$. Parallel partitioning of a PLA can be obtained if we transform the original PLA into an equivalent one whose graph G is disconnected.

This goal can be achieved by node splittings, ie by means of input, product and/or output augmentations. The procedure is shown in Figure 9 on the same simple example. The equivalent augmented PLA is also shown with its parallel partitioned implementation.

In general let us denote by $\Pi_l(E^B)$ a partition of the edge set E^B into l subsets $E^{B_1}, E^{B_2}, \ldots, E^{B_l}$. Let $G^B(P_j,O_j,E_j^B)$ the subgraph induced by the partition where P_j and O_j are the node sets of product and output nodes which are adjacent to edges in E_j^B. Let E_j^A be the set of edges incident to nodes in P_j and I_j be the set of nodes adjacent to P_j.

Because of output node splitting in general $|O| \leqslant \sum_{j=1}^{l} |O_j|$ and $|P| \leqslant \sum_{j=1}^{l} |P_j|$. Moreover also $|I| \leqslant \sum_{j=1}^{l} |I_j|$ because of the input augmentation required by equations (6) and (7). Any subgraph $G_j(I_j, P_j, O_j, E_j^A \cup E_j^B)$ corresponds to the jth PLA of the parallel partition.

An estimate of the area taken by the l logic subarrays and by the interconnect to route them is given by:

$$F = \sum_{j=1}^{l} |P_j|(a|I_j| + b|O_j|) + c \sum_{j=1}^{l} |I_j| + d \sum_{j=1}^{l} |O_j|$$
$$+ e \sum_{j=1}^{l} |P_j| + f[(\sum_{j=1}^{l} |I_j|) - |I|] + g[(\sum_{j=1}^{l} |O_j|) - |O|]$$
$$+ h[(\sum_{j=1}^{l} |P_j|) - |P|] \quad (14)$$

We can now state the parallel partitioning optimization problem OP3 as follows:

- Problem OP3. Find a partition $\hat{\Pi}_l(E^B)$ such that:

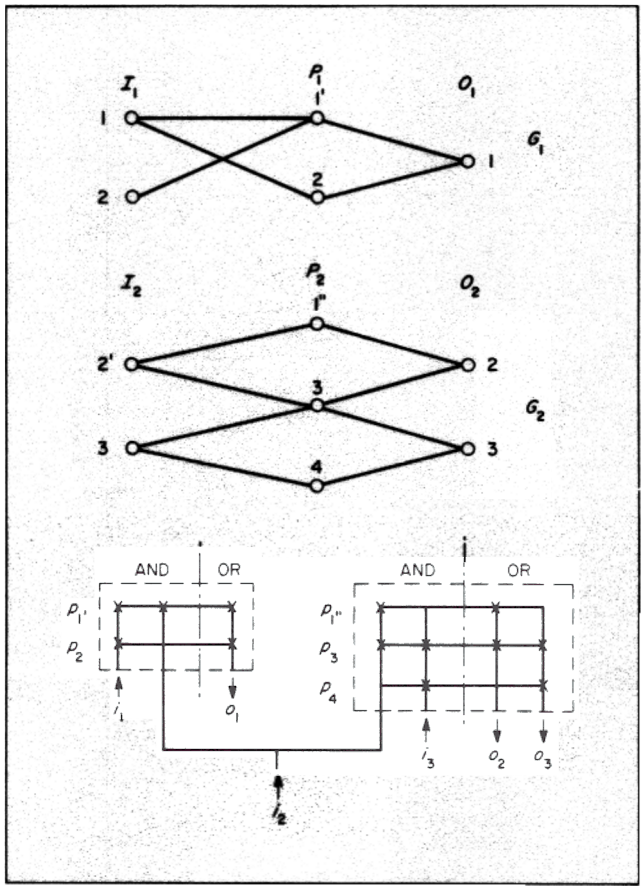

Figure 9. (Above) graphs G_1 and G_2 of the parallel-partitioned PLA (with product augmentation). (Below) parallel-partitioned PLA (with product augmentation)

$$F(\hat{\Pi}_l(E_j^B)) \leqslant F(\Pi_l(E_j^B)) \quad \forall \Pi_l(E_j^B) \text{ and } \forall l \quad (15)$$

Note that the optimal solution may not be unique.

The unconstrained partitioning of the edge set E_j^B may lead to several product augmentations and consequently input augmentations as required by equation (7). The augmentation may induce a kind of chain reaction. It is therefore more convenient to consider a constrained partitioning of the set E_j^B which avoids product augmentations. This corresponds to adding to equation (15) the following additional constraint:

$$P_j \cap P_k = \phi \quad \forall j,k = 1, 2, \ldots, l; \quad j \neq k \quad (16)$$

The procedure is shown in Figure 10 on the example.

HEURISTIC CLUSTERING ALGORITHM FOR PLA PARTITIONING

The optimization problems arising from PLA partitioning require the minimization of a nonlinear function with integer constraints. The objective functions depend on the cardinality of the node subsets induced by an edge set partitioning.

We propose a heuristic algorithm based on a cluster search[17] and on array transformations. We use the same cluster search strategy for the three partitioning problems. For this reason we denote by $G(V,E)$ the graph related to a partitioning problem. The node set V is defined as $I \cup P$, $P \cup O$ and $I \cup P \cup O$ and the edge set E as E^A, E^B

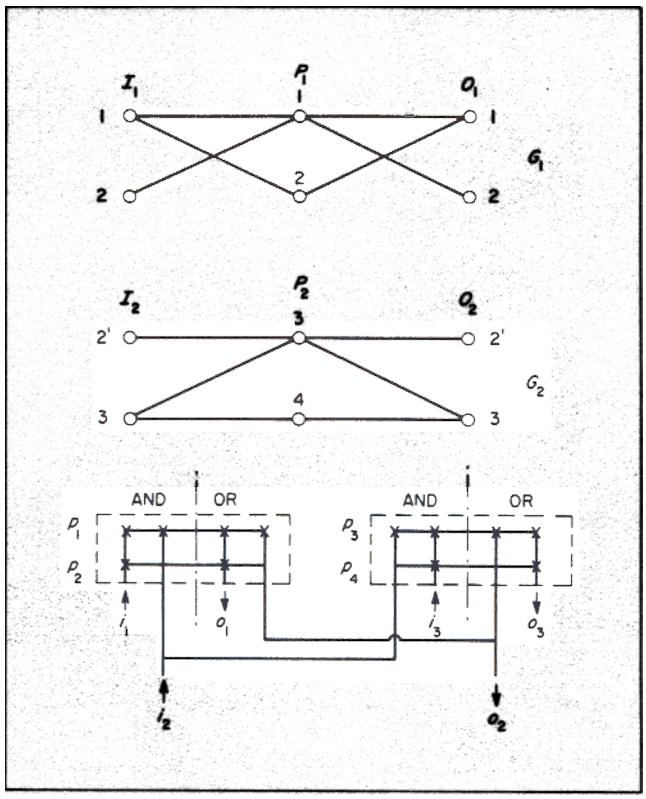

Figure 10. (Above) graphs G_1 and G_2 of the parallel-partitioned PLA (without product augmentation). (Below) parallel-partitioned PLA (without product augmentation)

and $E^A \cup E^B$ for input, output and parallel partitioning respectively.

The algorithm attempts first to find a node cluster inside graph $G(V,E)$ and then partitions V into two subsets V_1 and V_2. The former contains the cluster nodes and the latter the remaining ones. Let $\overline{E} \subset E$ be the set of edges joining nodes in V_1 to nodes in V_2. If $\overline{E}$ is empty, the node partition induces a graph partition into two disjoint subgraphs $G_1(V_1,E_1)$ and $G_2(V_2,E_2)$. If $\overline{E}$ is not empty, the algorithm modifies the graph by adding to V_1 and V_2 appropriate nodes incident to $\overline{E}$, so that E is partitionable into E_1 and E_2 and $G_1(V_1,E_1)$ and $G_2(V_2,E_2)$ are disjoint. This operation corresponds to node splitting (augmentation) and is described in detail later according to the different partitioning problems. Subgraph $G_1(V_1,E_1)$ is stored and the algorithm reattempts a cluster search on the updated graph $G(V,E) = G_2(V_2,E_2)$. The selection of cluster nodes is driven by the values taken by the objective function.

Different authors have dealt with clustering related problems[18–20]. We base our algorithm on the contour tableau approach[21,22]. The contour tableau is an array of three columns. The first one is called an IS (iterating set) and its entries are nodes of the graph. The second one is the AS (adjacency set) and its entries are sets of nodes of the graph. The third column is the OF (objective function) vector and for our purposes its entries are the values of the area estimates F^A, F^B and F.

The tableau is built iteratively until a cluster is found and convenient conditions are met to separate it from the rest of the graph. At this point the tableau is cleared and the algorithm restarts on the rest of the graph. The algorithm is described in pidgin Algol.

Partitioning algorithm

```
begin
  while (V≠φ) do
  begin
    IS = φ; AS = φ; OF = φ;
    i = 1;
    IS(i) = INSELECT [V];
    AS(i) = ADJ [IS(i)];
    while ({cluster criterion not satisfied}) do
    begin
      IS(i+1) = NEXTSELECT [AS(i)];
      AS(i+1) = NEXTADJ [IS,AS(i)];
      i = i+1;
    end
    G(V,E) = UPDATE [G(V,E)];
  end
end
```

Procedure ADJ $[i]$ returns the nodes adjacent to node i. Procedure NEXTADJ $[IS,AS(i)]$ returns all the nodes adjacent to node $IS(i+1)$ not contained in $\cup_{j=1}^{i} IS(j)$. An efficient way to evaluate the procedure is described in Sangiovanni Vincentelli et al[22]; the nodes returned by NEXTADJ are obtained from $AS(i)$ by deleting $IS(i+1)$ and adding the set of all the nodes which are adjacent to $IS(i+1)$ that are not already in $AS(i)$ or in $\cup_{j=1}^{i} IS(j)$. Procedure INSELECT $[V]$ selects an initial node of the graph $G(V,E)$ and procedure NEXTSELECT $[AS(i)]$ selects the next iterating node in $AS(i)$. Both selections follow an heuristic criterion described in the sequel. Procedure UPDATE $[G(V,E)]$ stores subgraph $G_1(V_1,E_1)$ and returns subgraph $G_2(V_2,E_2)$.

Graphs $G_1(V_1,E_1)$ and $G_2(V_2,E_2)$ are defined according to the partitioning problem and the augmentation strategy required. At each step of the internal **while** loop of the algorithm, the set V is partitioned into three disjoint subsets:

$$X = \cup_{j=1}^{i} IS(j) \quad Y = AS(i) \quad Z = V - X - Y \tag{17}$$

The nodes in X are inside the cluster and are adjacent only to nodes in $X \cup Y$. Nodes in set Y are 'border' nodes. By construction, the nodes in Z are not adjacent to any node in X. Let $W \subset X$ be the subset of nodes adjacent to Y at the current step of the algorithm. Let us define $X_I(X_P,X_O)$, $Y_I(Y_P,Y_O)$, $W_I(W_P,W_O)$, $Z_I(Z_P,Z_O)$ the subsets of input (product and output) nodes of X, Y, W and Z respectively (ie $X_I = X \cap I$).

In the case of input partitioning we augment only input columns. Hence the set V_1 is obtained by adding to cluster nodes X the input nodes Y_I adjacent to cluster nodes. Set V_2 is obtained by adding to the cluster complement set nodes $Y \cup Z$ the input cluster nodes W_I adjacent to them. Note that the product node set P is partitioned into two subsets X_P and $Z_P \cup Y_P$. The edge set E is partitioned accordingly: E_1 and E_2 are the subsets of E, whose elements are incident to nodes in X_P and $Z_P \cup Y_P$ respectively. Hence we define:

$$\begin{aligned} G_1(V_1,E_1) &= G_1(X \cup Y_I, E_1) \\ G_2(V_2,E_2) &= G_2(Y \cup Z \cup W_I, E_2) \end{aligned} \tag{18}$$

The following example illustrates this.

- Consider the AND plane of PLA shown in Figure 5. Suppose that at one step of the internal **while** loop the cluster set contains the following nodes:

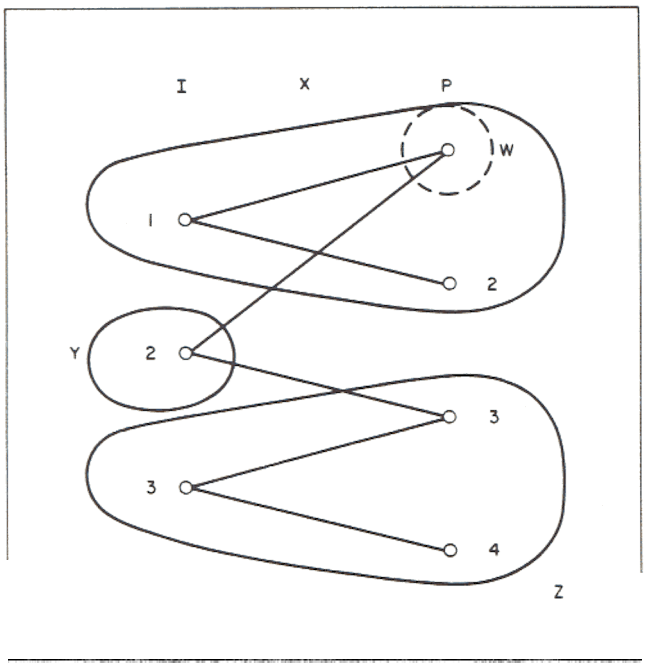

Figure 11. Node sets generated by cluster algorithm

$X = \{I_1, P_1, P_2\}$. The adjacency set is $Y = \{I_2\}$. The other two sets defined by the partitioning algorithm are: $W = \{P_1\}$ and $Z = \{I_3, P_3, P_4\}$ (Figure 11). According to equation (18) $V_1 = \{I_1, I_2, P_1, P_2\}$ and $V_2 = \{I_2, I_3, P_3, P_4\}$.

A similar definition applies, mutatis mutandis, to the output partitioning problem with product (output) augmentations only*.

$$G_1(V_1,E_1) = G_1(X \cup Y_P, E_1)$$
$$G_2(V_2,E_2) = G_2(Y \cup Z \cup W_P, E_2) \quad (19)$$

$$(G_1(V_1,E_1) = G_1(X \cup Y_O, E_1)$$
$$G_2(V_2,E_2) = G_2(Y \cup Z \cup W_O, E_2)) \quad (20)$$

In the case of parallel partitioning with input and output augmentations only, the set V_1 is obtained by adding to cluster nodes X the input nodes Y_I and the output nodes Y_O adjacent to cluster nodes. Set V_2 is obtained by adding to the cluster complement set nodes $Y \cup Z$ the input and the output cluster nodes $W_I \cup W_O$ adjacent to them. Note that the product node set P is partitioned into two subsets X_P and $Z_P \cup Y_P$ as in the input partitioning problem. The edge set E is partitioned accordingly: E_1 and E_2 are the subsets of E, whose elements are incident to nodes in X_P and $Z_P \cup Y_P$ respectively. We define:

$$G_1(V_1,E_1) = G_1(X \cup Y_I \cup Y_O, E_1)$$
$$G_2(V_2,E_2) = G_2(Y \cup Z \cup W_I \cup W_O, E_2) \quad (21)$$

The cluster criterion is satisfied when at least one of the following conditions is met:

$$|AS(i)| = 0 \quad (22)$$

$$\gamma(|X_I|, |X_P|, |X_O|, |Y_I|, |Y_P|, |Y_O|, |W_I|, |W_P|, |W_O|) > \gamma_{max} \quad (23)$$

$OF(i)$ is a local minimum (24)

*In the case of output partitioning with product and output duplications and parallel partitioning with input, output and product duplications, subgraphs $G_1(V_1,E_1)$ and $G_2(V_2,E_2)$ are defined differently. Since these definitions do not affect the analysis of the algorithm, they are not reported here for the sake of simplicity.

The first condition guarantees that a cluster is found if graph $G(V,E)$ is not connected. The second condition allows the user to define a scalar function γ of the cardinality of the subsets $X_I, X_P, X_O, Y_I, Y_P, Y_O, W_I, W_P$ and W_O in order to specify the maximum size of each block according to the technological constraints of the implementation of the partitioned array. The third condition is a heuristic rule to determine a cluster. It can be also required that $OF(i)$ is smaller than a proper fraction of the initial area $OF(0)$ to ensure that partitioning is performed only if it gives a considerable saving in the total area. Since the objective function vector may have several local minima close to each other, the cluster decision can be taken a few steps after the minimum is detected.

We can now describe procedure NEXTSELECT. Procedure NEXTSELECT uses a greedy strategy to select the next iterating node among the nodes in $AS(i)$. When any node in $AS(i)$ is added to the cluster node set X, graph $G(V,E)$ can be partitioned according to equations (18), (19), (20) or (21) and the corresponding value of the objective function be computed. The selected node is the one that minimizes the objective function at that step of the algorithm. This means that the selected node is the 'local best' node.

Procedure INSELECT returns the initial iterating node. As pointed out in Sangiovanni Vincentelli et al[22], a node connecting two clusters is a bad selection of initial node. Nodes with degree 1 cannot join two clusters and hopefully the lower the degree of the node, the lower is the probability of choosing a 'bad' node. Hence procedure INSELECT returns the min-degree node in the actual implementation of the algorithm.

It is interesting to show that the time computational complexity of the algorithm is polynomially bounded, although the total number of nodes may increase at each iteration. Let $n = |V|$.

Theorem: The time computational complexity of the partitioning algorithm is bounded by $O(n^3)$.

Proof: Every time the algorithm cycles through the external while loop, procedure UPDATE $[G(V,E)]$ returns $G_2(V_2,E_2)$. At least one node of the cluster set is not added to V_2, because otherwise $G(V,E) = G_2(V_2,E_2)$ and the cluster condition cannot be met. Hence $V_2 \subset V$ and $|V|$ is decreasing at every step of the external loop. The algorithm cycles at most n times through the external while loop. Moreover since $AS(i) \subset V$ and $|AS(i)| < n$, the algorithm will execute at most n inner inner while loops, because there is necessarily an integer m, $m < n$, such that $|AS(m)| = 0$ and a cluster condition is satisfied. Since procedures NEXTSELECT and NEXTADJ can perform at most n comparisons and objective function evaluations, the time complexity of the algorithm is bounded by $O(n^3)$.

SMILE DESCRIPTION

Smile is an interactive program for programmed logic arrays partitioning. The program is a module of an integrated system for the automated synthesis of programmed logic arrays and finite state machines developed at the University of California, Berkeley, USA.

The PLA description is given as input to the program in the form of personality matrix (Figure 3(*a*)). The output file of logic minimizer Presto[23] can be used as input to Smile. Partitioning instructions are entered into the

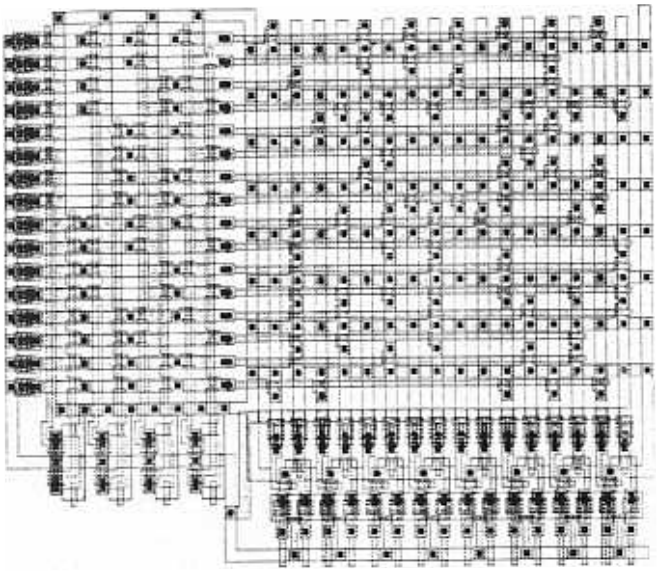

Figure 12. Check plot of a benchmark PLA (PLA 2) before partitioning

program along with the personality in the input file. Input, output or parallel partitioning can be requested. The program performs input and output augmentations by default. In the case of output partitioning, product augmentations can be allowed.

The user can require limitation of the number of clusters, ie the number of subarrays in which a plane (or both planes) is partitioned as well as the maximum size of the subarrays.

Smile generates an output file containing a symbolic matrix, representing the personality of the partitioned array (Figure 3(*b*)). As an example consider the PLA shown in Figure 3. Since the OR plane is very sparse, an output partitioning is attempted by the program. Smile partitions the output column set into two disjoint subsets: $\{o_5, o_8, o_{10}, o_{11}, o_{14}, o_{17}, o_{18}, o_{20}\}$ $\{o_6, o_7, o_9, o_{12}, o_{13}, o_{15}, o_{16}, o_{19},\}$. Three product terms, namely p_3, p_{11} and p_{12}, are augmented in order to transform the original array into an equivalent partitionable one. Figure 4 shows the output partitioned array. The OR plane has been implemented as a block folded array. Note that the array size has changed: eight columns are not needed for the PLA implementation at the expense of adding three extra rows. A global area saving of 29 per cent has been achieved.

The Smile output file can be processed by a silicon assembler program, which generates the mask layout of the array according to a given technology. Note that the symbolic array is technology independent. We used the program Plaid[24] to assemble the partitioned PLA as a column block folded array. The check plot of the original array is shown in Figure 12 and the block folded implementation of the partitioned array in Figure 13.

Table 1. Normalized partitioned array areas. Initial area = 100

PLA	size $P*(N+M)$	Input partitioning	Output partitioning	Parallel partitioning
PLA 1		71	64	61
PLA 2		100	71	65
PLA 3		78	81	67
PLA 4		75	70	46
PLA 5		75	80	60
PLA 6		71	81	59
PLA 7		69	81	57

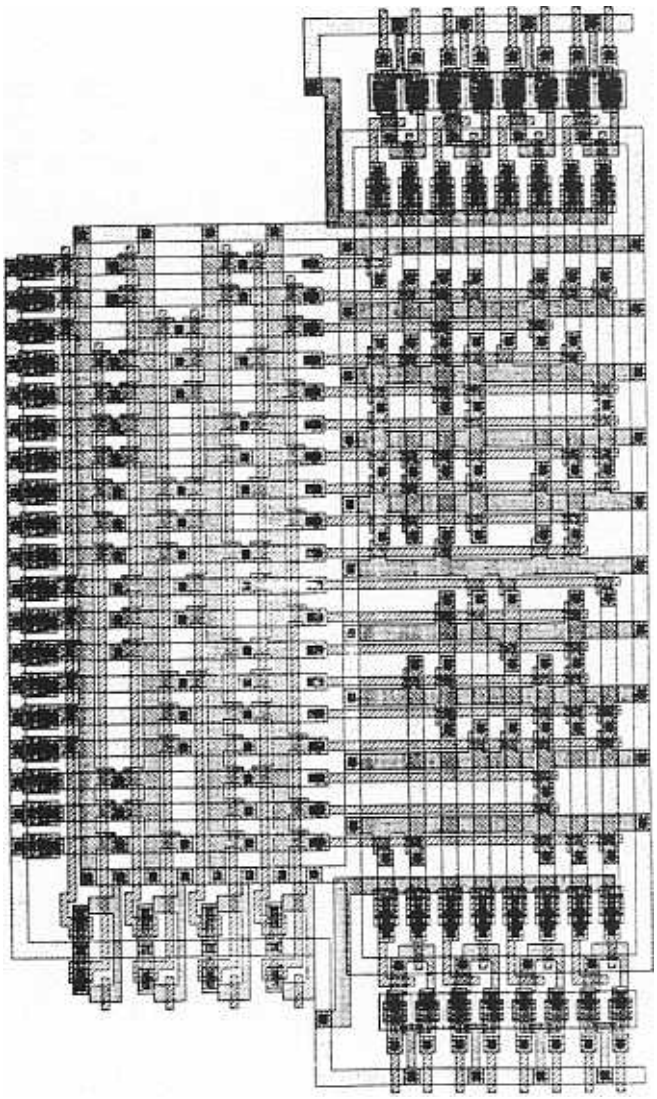

Figure 13. Check plot of a benchmark PLA (PLA 2) after output-partitioning

The program Smile is coded in Ratfor. Intermediate code in Fortran 77 is available. Smile runs in a VAX-UNIX environment, but is easily transportable to other machines.

EXPERIMENTAL REMARKS

We tested Smile on a large set of industrial PLAs. Some results are reported in Table 1. The time spent by the algorithm ranges from a few hundreds of milliseconds for PLA 1 to several seconds for larger arrays (PLA 7). Since execution time is small, circuit designers may want to use the program with different requirements in order to compare the different partitioned structures.

Note that it is not possible to achieve an area reduction of PLA 2 by means of input partitioning, because the AND plane has a full structure (no 'don't cares').

CONCLUSIONS

The method presented in this paper attempts to save the total silicon area by adding extra columns (and/or rows) to the array. Smile is a first implementation of this method and has been used to test its validity.

Future work in this direction includes the investigation of more general rules for array transformations to allow partitioning and the implementation of other partitioning algorithms based (or not) on cluster methods.

ACKNOWLEDGEMENTS

The authors wish to thank Professor Alberto Sangiovanni-Vincentelli for many helpful and stimulating discussions.

REFERENCES

1 **Fleisher, J and Maissel, L I** 'An introduction to array logic' *IBM J. Res. Devel.* Vol 19 (March 1975) pp 98–109

2 **Schmookler, M S** 'Design of large ALUs using multiple PLA macros' *IBM J. Res. Devel.* Vol 24 (January 1980) pp 2–14

3 **Mead, C and Conway, L** *Introduction to VLSI systems* Addison Wesley (1980)

4 **Newton, A R, Pederson, D O, Sangiovanni-Vincentelli, A L and Sequin, C H** 'Design aids for VLSI: the Berkeley perspective' *IEEE Trans. Circ. Sys.* Vol CAS 28 (July 1981) pp 618–633

5 **Hong, S J, Cain, R G and Ostapko, D L** 'MINI: a heuristic approach for logic minimization' *IBM J. Res. Devel.* Vol 18 (September 1974) pp 443–458

6 **Brayton, R, Hachtel, G D, Hemachanandra, L, Newton, A R and Sangiovanni-Vincentelli, A L** 'A comparison of logic minimization strategies using Espresso, an APL program package for partitioned logic minimalization' *Proc. 1982 Int. Symp. Circ. Syst.* Rome, Italy (May 1982)

7 **Wood, R A** 'A high density programmable logic array chip' *IEEE Trans. Comput.* Vol C-28 (September 1979) pp 602–608

8 **Hachtel, G D, Newton, A R and Sangiovanni-Vincentelli, A L** 'An algorithm for optimal PLA folding' *IEEE Trans. CAD Int. Circ. Sys.* Vol 1 No 2 (April 1982) pp 63–76

9 **Hachtel, G D, Newton, A R and Sangiovanni-Vincentelli, A L** 'Techniques for programmable logic arrays folding' *Proc. 19th Design Automation Conf.* Las Vegas, CA, USA (June 1982)

10 **De Micheli, G and Sangiovanni-Vincentelli, A L** 'PLEASURE: a computer program for simple and multiple constrained folding of Programmable Logic Arrays' unpublished paper

11 **Kang, S** *Automated synthesis of PLA based systems* Ph D thesis Stanford University, CT, USA (1981)

12 **Egan, J R and Liu, C L** 'Optimal bipartite folding of PLA' *Proc. 19th Design Automation Conf.* Las Vegas, CA, USA (June 1982)

13 **Greer, D L** 'An associative logic matrix' *IEEE J. Solid State Circuits* Vol SC-11 No 5 (October 1976) pp 679–691

14 **Suwa, I and Kubitz, W J** 'A computer aided design system for segment-folded PLA macro cells' *Proc. 18th Design Automation Conf.* Nashville, TN, USA (June 1981)

15 **Chuquillanqui, S and Perez Segovia, T** 'PAOLA: a tool for topological optimization of large PLAs' *Proc. 19th Design Automation Conf.* Las Vegas, CA, USA (June 1982)

16 **Lawler, E** *Combinatorial optimization: networks and matroids* Holt Rinehart and Winston (1976)

17 **Spath, H** *Cluster analysis algorithms* Ellis Horwood (1980)

18 **Luccio, F and Sami, M** On the decomposition of networks in minimally interconnected subnetworks' *IEEE Trans. Circuit Theory* Vol CT-16 (May 1969) pp 181–188

19 **Lawler, E L** 'Cutset and partitions of hypergraphs' *Networks* No 3 (July 1973) pp 275–285

20 **Kernigham, B W and Lin S** 'An efficient heuristic procedure for partitioning graphs' *Bell Syst. Tech. J.* Vol 49 No 2 (February 1970) pp 291–307

21 **Ogbuobiri, E C, Tinney, W F and Walker, J W** 'Sparsity-directed decomposition for Gaussian elimination on matrices' *IEEE Trans. Power Appl. Syst.* Vol PAS-89 No 1 (January 1970) pp 141–150

22 **Sangiovanni-Vincentelli, A, Chen, L and Chua, L O** 'An efficient cluster algorithm for tearing large-scale networks' *IEEE Trans. Circ. Syst.* Vol CAS-24 No 12 (December 1977) pp 709–717

23 **Brown, D W** 'A state-machine synthesizer – SMS' *Proc. 18th Design Automation Conf.* Nashville, TN, USA (June 1981)

24 **Hoffman, M** 'A method for topological compaction of programmed logic arrays' *Master Report ERL University of California* Berkeley, CA, USA (1981)