A CLUSTERING ALGORITHM FOR PARTITIONING OF PROGRAMMED LOGIC ARRAYS

by

G. De Micheli and M. Santomauro

Memorandum No. UCB/ERL M82/74

22 October 1982

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A CLUSTERING ALGORITHM FOR PARTITIONING OF PROGRAMMED LOGIC ARRAYS [1]

Giovanni De Micheli

Mauro Santomauro [2]

Department of Electrical Engineering and Computer Sciences

University of California, Berkeley

ABSTRACT

The optimal topological design of Programmed Logic Arrays allows to implement complex switching functions in a minimal silicon area. We address the problem of **array partitioning** and the implementation of partitioned arrays as **block folded** or **parallel connected** PLAs. We present a graph theoretical interpretation of the problem and an efficient heuristic algorithm. Experimental results are reported.

---

[1] This research has been partially sponsored by IBM , DARPA grant # 25697 and Consiglio Nazionale delle Ricerche, Italy.

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

1. INTRODUCTION

Array logic is widely used in the structured design of VLSI circuits and systems [1]. In particular Programmed Logic Arrays (PLA) are a convenient implementation of multiple output switching functions [2][3], because they show a regular structure and can be effectively designed and optimized with the support of computer aids [4].

Though heuristic minimizers [5] [6] allow to express large switching functions as minimal sets of logical implicants , their physical implementation may still be too expensive in terms of silicon area. Large logic arrays are in general 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 optimal topological design aims to reduce the wasted area. A well known technique is PLA **folding** [7] [8] [9]. 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 physical logic array.

An alternative approach is **block folding** [10] which has been referred to also as bipartite folding in [11] and as array segmentation in [12] and [13]. 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.

PLA decomposition into **parallel connected** arrays has been investigated in

[12]. 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** . We present an algorithm for partitioning a logic array (or a plane of a logic array) into n subarrays . The algorithm takes advantage of **array transformations** based on logical operations to ease partitioning. Since partitioned arrays can be implemented as multiple block folded or parallel connected arrays , our approach embodies the previous proposed implementations as special cases.

The implementation of a partitioned array as a block folded array can be obtained in two different ways: **column block folding** and **row block folding** . Note that a multiple column block folded array requires a physical layout where inputs and/or outputs are connected from the side of the array. We refer to [14] for technological details.

In the parallel connected implementation each subarray can have different size and be placed in arbitrary position. Though this implementation is more general , the total area depends heavily on the interconnections among the subarrays.

2. BASIC CONCEPTS AND DEFINITIONS

We consider here Programmed Logic Arrays 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 (fig. 2.1a). 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 topological description of a PLA is contained in the *Topological Personality Matrix* (TPM) whose entries are 1 if the corresponding element in the PLA is a care, and is 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 (fig. 2.1b).

We define *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 *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 (input product row) and p_j^B (output product row), where p_j^A contains the first N entries of p_j and p_j^B the last ones.

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

$$x \vee y \quad (x \wedge y) \tag{2.1}$$

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

$$\vee_{i=1}^{n} x_i \quad (\wedge_{i=1}^{n} x_i) \tag{2.2}$$

throughout the paper.

Two vectors x, y are *independent (orthogonal)* if $x \wedge y = 0$, where 0 is the null vector. We denote by $x \perp y$ two independent vectors.

Two vector sets X, Y are *independent* if

$$x \perp y \quad \forall x \in X \text{ and } \forall y \in Y \tag{2.3}$$

Logic array partitioning relies on determining independent sets of vectors in the Topological Personality Matrix. 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 $p_j^A(p_j^B)$. A logic array is said to be *parallel partitionable* (or simply *partitionable*) if there exist product row independent sets.

Remark 2.1: 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. ■

3. 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 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 have 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 which gives an equivalent logic array. We present now 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:

i) 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

$$\vee_{k=1}^{s} i_{jk} = i_j \tag{3.1}$$

ii) 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

i) 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:

$$\vee_{k=1}^{s} o_{jk} = o_j \tag{3.2}$$

ii) The output signal from column o_j corresponds to the logic conjunction of the output signals from the column in O_j.

An output partitionable array can be obtained by a sequence of output column augmentations, and a partitionable logic array by a sequence of input and output augmentations.

Rule 3: *product row augmentation*

The logic array defined by A,B and A',B' are equivalent if:

i) [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

$$\vee_{k=1}^{s} p_{jk}^{B} = p_j^{B} \tag{3.3a}$$

$$p_{jk}^{A} = p_j^{A} \quad \forall k = 1,2,\ldots,s \tag{3.3b}$$

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.

4. 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. We refer the reader to [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, i.e. the tripartite graph $G(I,P,O,E)$, where $E=E^A \cup E^B$ (fig. 4.1).

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_0 on G as follows:

$$F_0 = (a|N|+b|M|)|P| + c|N| + d|M| + e|P| \qquad (4.1)$$

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 the order.

4.1 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, $j=1,2,...,n$. Because of independence, input columns are also partitioned into n disjoint sets I_j. As a consequence graph G^A is disconnected into subgraphs $G_j^A = (I_j, P_j, E_j^A)$ $j=1,2,...,n$.

Each subgraph G_j^A represents a block of an input partitioned PLA.

It is straightforward that in this case an input partitioned array takes 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 fig. 4.2 on a simple example.

Example 4.1: Input node 2 is split into two nodes 2' and 2'' (column augmentation on the PLA) and the edges incident to 2 are now incident ei-

ther 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 $|I| \leq \sum_{j=1}^{n} |I_j|$ 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:

$$F^A = \sum_{j=1}^{n} |P_j|(a|I_j| + b|O|) + c\sum_{j=1}^{n} |I_j| + d|O| + e|P|$$

$$+ f\left(\sum_{j=1}^{n} |I_j| - |I|\right) \qquad (4.2)$$

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 $\Pi_n^*(E^A)$ such that:

$$F^A(\Pi_n^*(E^A)) \leq F^A(\Pi_n(E^A)) \quad \forall \Pi_n(E^A) \text{ and } \forall n \qquad (4.3a)$$

$$P_j \cap P_k = \phi \quad \forall j,k = 1,2,\ldots,n;\ j \neq k \qquad (4.3b)$$

Note that the optimal solution can be not unique.

4.2 Output partitioning

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

As stated in Section 3, output partitioning can be achieved by output column and /or product row augmentation. The procedure is shown in fig. 4.3 on a simple example.

Example 4.2: 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 $|O| \leq \sum_{j=1}^{m} |O_j|$ and $|P| \leq \sum_{j=1}^{m} |P_j|$. Subgraphs G_j^B $j=1,2,..,m$ correspond to the blocks of the output partitioned array.

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

$$F^B = \sum_{j=1}^{m} |P_j|(a|I|+b|O_j|) + c|I| + d\sum_{j=1}^{m}|O_j| + e\sum_{j=1}^{m}|P_j|$$

$$+ g[(\sum_{j=1}^{m}|O_j|)-|O|] + h[(\sum_{j=1}^{m}|P_j|)-|P|] \qquad (4.4)$$

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 $\Pi_m^*(E^B)$ such that:

$$F^B(\Pi_m^*(E^B)) \leq F^B(\Pi_m(E^B)) \quad \forall \Pi_m(E^B) \text{ and } \forall m \qquad (4.5)$$

Note that the optimal solution can be not unique.

Remark 4.1: If only output column augmentations are allowed, the last term in (4.4) 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 (4.3b) to equation (4.5) ■

4.3 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 i.e. by means of input ,product and/or output augmentations. The procedure is shown in fig. 4.4 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 $|O| \leq \sum_{j=1}^{l} |O_j|$ and $|P| \leq \sum_{j=1}^{l} |P_j|$.

Moreover also $|I| \leq \sum_{j=1}^{l} |I_j|$ because of the input augmentation required by eqn. (3.3). Any subgraph $G_j(I_j, P_j, O_j, E_j^A \cup E_j^B)$ corresponds to the $j-th$ 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:

$$F = \sum_{j=1}^{l} |P_j|(a|I_j|+b|O_j|) + c\sum_{j=1}^{l}|I_j| + d\sum_{j=1}^{l}|O_j| + e\sum_{j=1}^{l}|P_j|$$

$$+ f[(\sum_{j=1}^{l}|I_j|)-|I|] + g[(\sum_{j=1}^{l}|O_j|)-|O|] + h[(\sum_{j=1}^{l}|P_j|)-|P|] \qquad (4.6)$$

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

"Problem OP3"

Find a partition $\Pi_l^*(E^B)$ such that:

$$F(\Pi_l^*(E_j^B)) \leq F(\Pi_l(E_j^B)) \quad \forall \Pi_l(E_j^B) \text{ and } \forall l \qquad (4.7)$$

Note that the optimal solution can be not unique.

Remark 4.2: The unconstrained partitioning of the edge set E_j^B may lead to several product augmentations and consequently input augmentations as required by eqn. (3.3b). 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 add to eqn. (4.7) the following additional constraint:

$$P_j \cap P_k = \phi \qquad \forall j,k=1,2,\ldots,l; \quad j \neq k \qquad (4.8)$$

The procedure is shown in fig. (4.5) on the example. ■

5. A HEURISTIC CLUSTERING ALGORITHM FOR PLA PARTITIONING

The optimization problems arising from PLA partitioning require to minimize 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 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 in the sequel 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],[19],[20]. We base our algorithm on the **contour tableau** approach described in [21] and in [22]. The contour tableau is an array of three columns. The first one is called *iterating set* (IS) and its entries are nodes of the graph. The second one is the *adjacency set* (AS) and its entries are sets of nodes of the graph. The third

column is the *objective function* vector (OF) 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 $\bigcup_{j=1}^{i} IS(j)$. An efficient way to evaluate the procedure is described in [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 $\bigcup_{j=1}^{i} IS(j)$. 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, namely:

$$X = \cup_{j=1}^{i} IS(j) \qquad Y = AS(i) \qquad Z = V - X - Y. \tag{5.1}$$

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 $W \subseteq X$ 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 (i.e. $X_I = X \cap I$).

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 Z 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:

$$G_1(V_1, E_1) = G_1(X \cup Y_I, E_1) \qquad G_2(V_2, E_2) = G_2(Y \cup Z \cup W_I, E_2) \tag{5.2}$$

Example 5.1 : Consider the AND plane of PLA shown in fig 2.1. Suppose that at one step of the internal *while* loop the cluster set contains the following nodes : $X = \{I_1, P_1, P_2\}$. The adjacency set is: $Y = \{I_2\}$. The other two sets defined by the partitioning algorithm are : $W = \{P_1\}$ and $Z = \{I_3, P_3, P_4\}$ (Fig. 5.1). According to (5.2) $V_1 = \{I_1, I_2, P_1, P_2\}$ 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 : [1]

$$G_1(V_1, E_1) = G_1(X \cup Y_P, E_1) \qquad G_2(V_2, E_2) = G_2(Y \cup Z \cup W_P, E_2) \quad (5.3a)$$

$$(\quad G_1(V_1, E_1) = G_1(X \cup Y_O, E_1) \qquad G_2(V_2, E_2) = G_2(Y \cup Z \cup W_O, E_2) \quad) \quad (5.3b)$$

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 Z 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: [1]

$$G_1(V_1, E_1) = G_1(X \cup Y_I \cup Y_O, E_1) \qquad G_2(V_2, E_2) = G_2(Y \cup Z \cup W_I \cup W_O, E_2) \quad (5.4)$$

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

$$|AS(i)| = 0 \quad (5.5a)$$

$$\gamma(|X_I|, |X_P|, |X_O|, |Y_I|, |Y_P|, |Y_O|, |W_I|, |W_P|, |W_O|) > \gamma_{max} \quad (5.5b)$$

$$OF(i) \text{ is a local minimum} \quad (5.5c)$$

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

[1] 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.

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 (5.2) , (5.3) or (5.4) 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 [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 , though the total number of nodes may increase at each iteration. Let $n = |V|$.

Theorem 4.1 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 $O(n^3)$. ■

6. EXPERIMENTAL AND CONCLUDING REMARKS

The Partitioning Algorithm has been coded in *ratfor* and tested on several industrial Programmed Logic Arrays. Some results are reported in table 1. Fig. 6.1a shows the personality of a benchmark array (PLA 2) and Fig. 6.1b the output partitioned personality. Note that for this particular PLA , having no "don't cares" in the input plane TPM , no input partition is found by the algorithm.
The partitioned personality can be used as input to a silicon assembler program to obtain a layout of the block folded PLA masks. We used program *plaid* [23] to assemble the block folded PLA of fig. 6.1. The checkplot of the original array is shown in fig. 6.2a and the block folded implementation in fig. 6.2b.

7. ACKNOWLEDGEMENT

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

8. REFERENCES

[1] J.Fleisher and L.I.Maissel "An Introduction to Array Logic" *IBM Jour. on Res. and Devel.* vol 19 pp 98-109 Mar 75

[2] M.S.Schmookler "Design of Large ALUs Using Multiple PLA Macros" *IBM Jour. on Res. and Devel.* vol 24 pp 2-14 Jan 80

[3] C.Mead and L.Conway "*Introduction to VLSI systems*" Addison Wesley 1980

[4] A.R.Newton, D.O.Pederson, A.L.Sangiovanni Vincentelli and C.H.Sequin "Design Aids for VLSI: the Berkeley Perspective" *IEEE Tran. on Circ. and Sys.* vol CAS 28 pp 618-633 Jul 81

[5] S.J.Hong,R.G.Cain and D.L.Ostapko "MINI:a Heuristic Approach for Logic Minimization" *IBM Jour. on Res. and Devel.* vol 18 pp 443-458 Sep 74

[6] R.Brayton,G.D.Hachtel,L.Hemachanandra,A.R.Newton and A.L.Sangiovanni Vincentelli "A Comparison of Logic Minimization Strategies Using Espresso. An APL Program Package for Partitioned Logic Minimalization" *Proc. 1982 Inter. Symp. on Circ. and Syst.* Rome May 82

[7] R.A.Wood "A High Density Programmable Logic Array Chip", *IEEE Trans. on Comput.* vol C-28 pp 602-608 Sep 79

[8] G.D.Hachtel,A.R.Newton and A.L.Sangiovanni Vincentelli "An Algorithm for Optimal PLA Folding" *IEEE Trans on CAD of Int. Circ. and Sys.* vol 1 no 2 pp 63-76 Apr 82

[9] G.D.Hachtel,A.R.Newton and A.L.Sangiovanni Vincentelli "Techniques for Programmable Logic Arrays Folding" *Proc. 19th Design Automation Conf.* Las Vegas Jun 82

[10] Sungho Kang "Automated Synthesis of PLA Based Systems" *Ph.D. Dissertation Stanford University 1981*

[11] J.R.Egan and C.L.Liu " Optimal Bipartite Folding of PLA" *Proc. 19th Design Automation Conf.* Las Vegas Jun 82

[12] D.L.Greer " An Associative Logic Matrix " *IEEE Jour. of Solid State Circuits* vol SC-11 no.5 pp 679-691 Oct 76

[13] I.Suwa and W.J.Kubitz " A Computer Aided Design System for Segment-Folded PLA Macro Cells" *Proc. 18th Design Automation Conf.* Nashville Jun 81

[14] G.De Micheli and A.L.Sangiovanni Vincentelli "PLEASURE: a computer program for simple and multiple constrained folding of Programmable Logic Arrays" submitted for publication.

[15] S.Chuquillanqui and T. Perez Segovia " PAOLA: A Tool for Topological Optimization of Large PLAs" *Proc. 19th Design Automation Conference* Las Vegas Jun 82

[16] E.Lawler " *Combinatorial Optimization : Networks and Matroids*" Holt Rinehart and Winston 1976

[17] H.Spath "*Cluster Analysis Algorithms*" Ellis Horwood 1980

[18] F.Luccio and M.Sami "On the Decomposition of Networks in Minimally Interconnected Subnetworks" *IEEE Trans. on Circuit Theory* vol CT-16 pp 184-188 May 69

[19] E.L.Lawler "Cutset and Partitions of Hypergraphs" *Networks* no 3 pp 275-285 Jul 73

[20] B.W.Kernigham and S.Lin "An Efficient Heuristic Procedure for Partitioning Graphs" *Bell Sys. Tech. Jour.* vol 49 no 2 pp 291-307 Feb 70

[21] E.C.Ogbuobiri,W.F.Tinney and J.W.Walker "Sparsity-directed Decomposition for Gaussian Elimination on Matrices" *IEEE Trans. on Power App. and Sys.* vol PAS-89 no 1 pp 141-150 Jan 70

[22] A.Sangiovanni Vincentelli,Li-Kuan Chen and L.O.Chua "An Efficient Cluster Algorithm for Tearing Large-Scale Networks" *IEEE Trans. on Circ. and Sys.* vol CAS-24 no 12 pp 709-717 Dec 77

[23] M. Hoffman "A Method for Topological Compaction of Programmed Logic Arrays" *Master Report ERL University of California Berkeley* 1981.

TABLE 1

Normalized partitioned array areas. Initial area = 100.				
PLA	size P*(N+M)	Input Partitioning	Output Partitioning	Parallel Partitioning
PLA 1	6*(6+4)	71	64	61
PLA 2	16*(4+16)	100	71	65
PLA 3	30*(19+10)	78	81	67
PLA 4	75*(35+29)	75	70	46
PLA 5	62*(24+14)	75	80	60
PLA 6	84*(27+10)	71	81	59
PLA 7	84*(27+10)	69	81	57

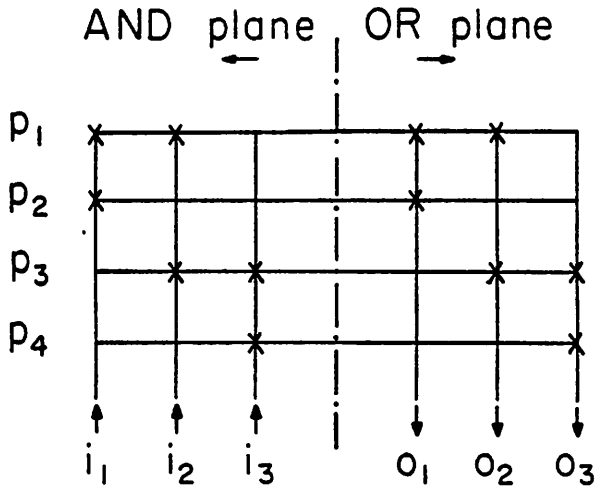

Fig. 2.1a Example of a PLA

$$TPM = \left[\begin{array}{ccc|ccc} & A & & & B & \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Fig. 2.1b Topological Personality Matrix

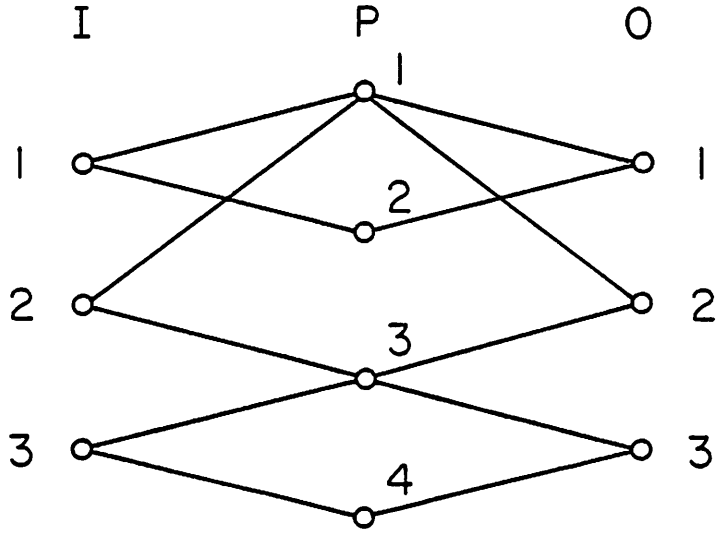

Fig. 4.1 G of the original PLA

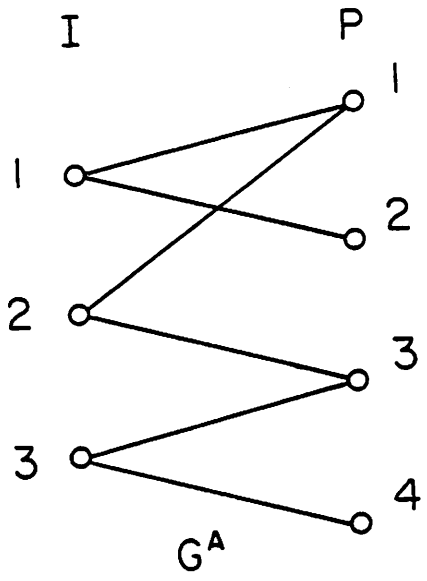

Fig. 4.2a G^A of the original PLA

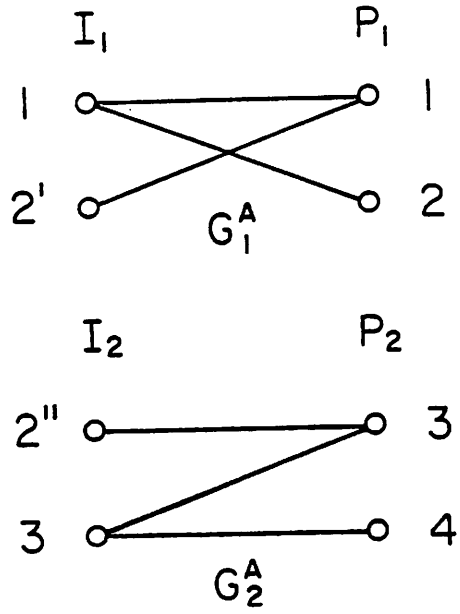

Fig. 4.2b G_1^A and G_2^A of the input-partitioned PLA

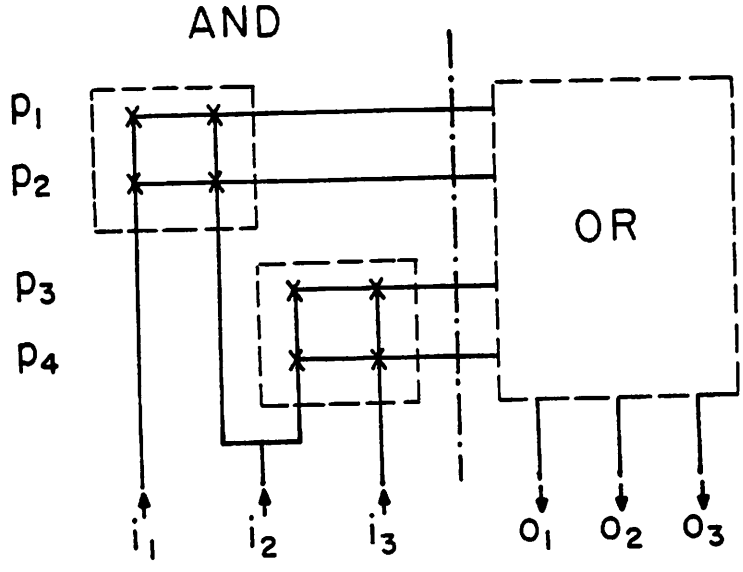

Fig. 4.2c Input-partitioned PLA

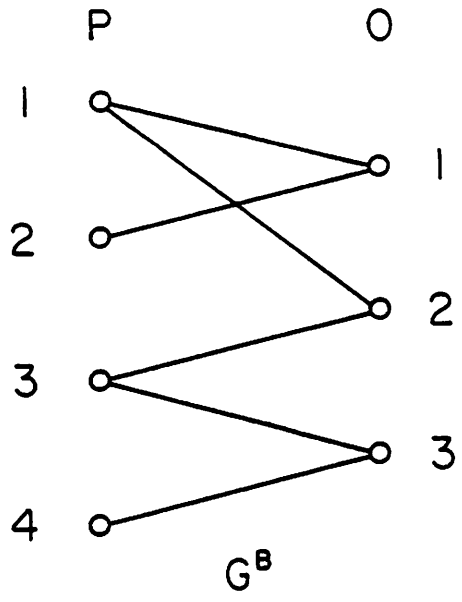

Fig. 4.3a G^B of the original PLA

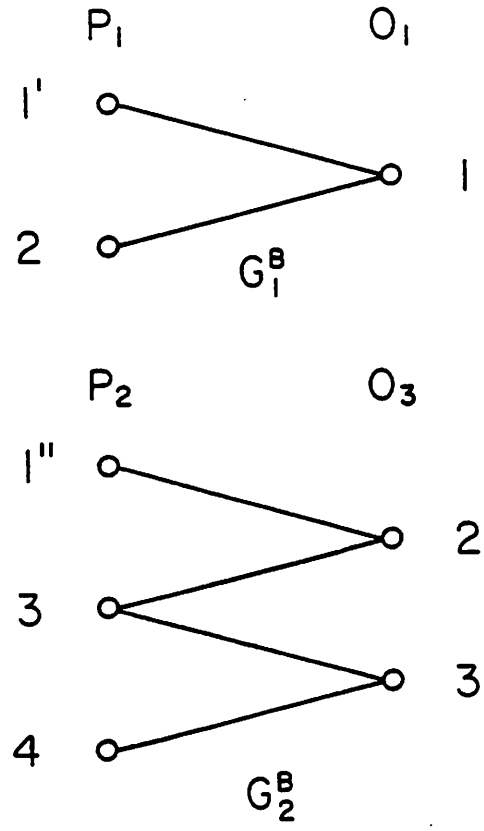

Fig. 4.3b G_1^B and G_2^B of the output-partitioned PLA

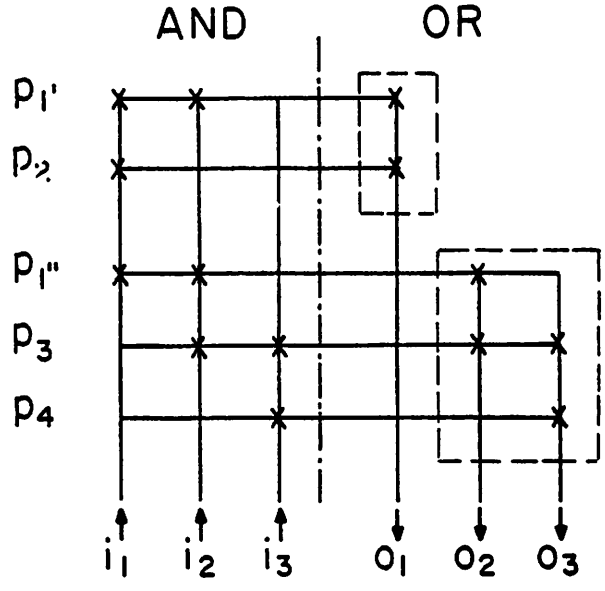

Fig. 4.3c Output-partitioned PLA

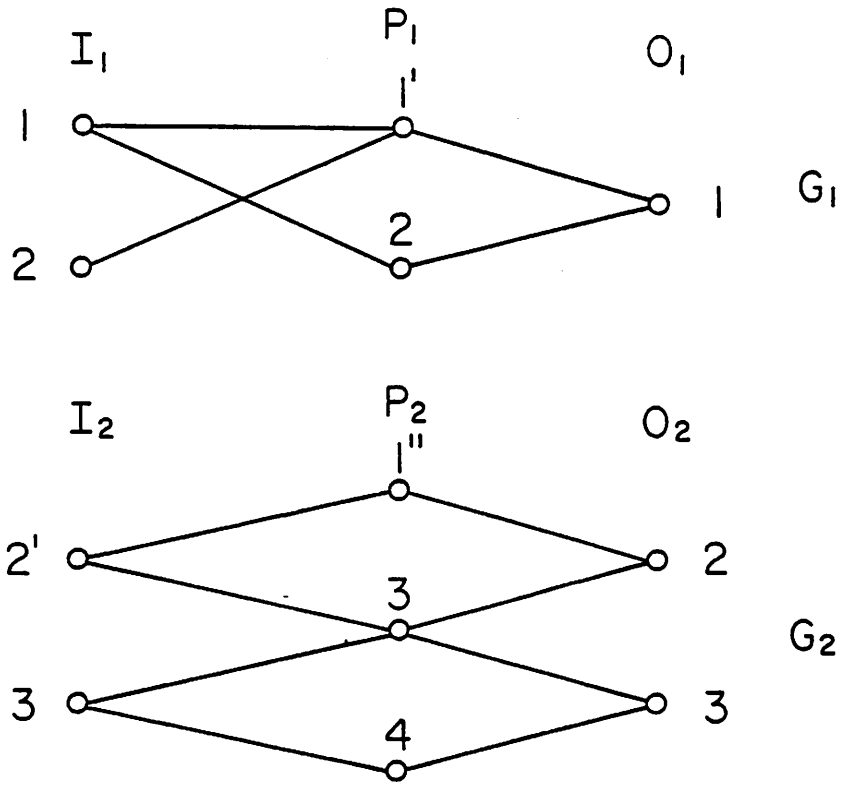

Fig. 4.4a G_1 and G_2 of parallel-partitioned PLA (with product augmentation)

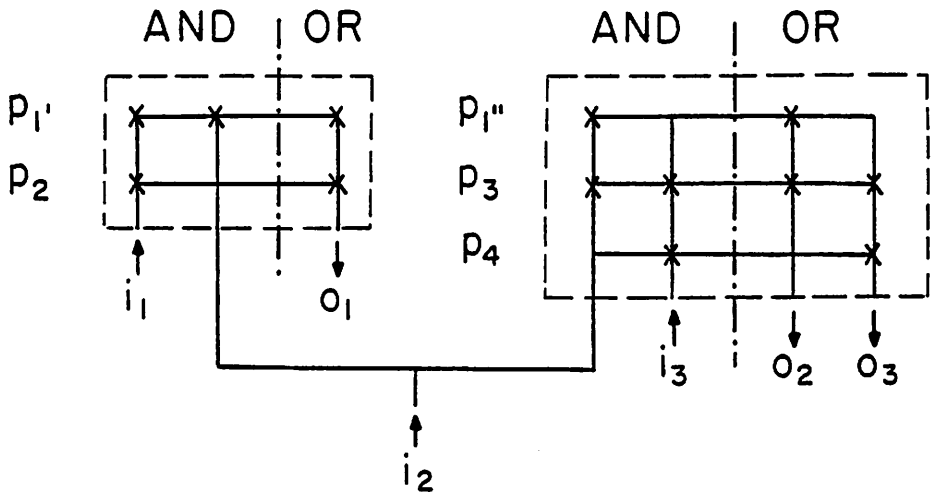

Fig. 4.4b Parallel-partitioned PLA (with product augmentation)

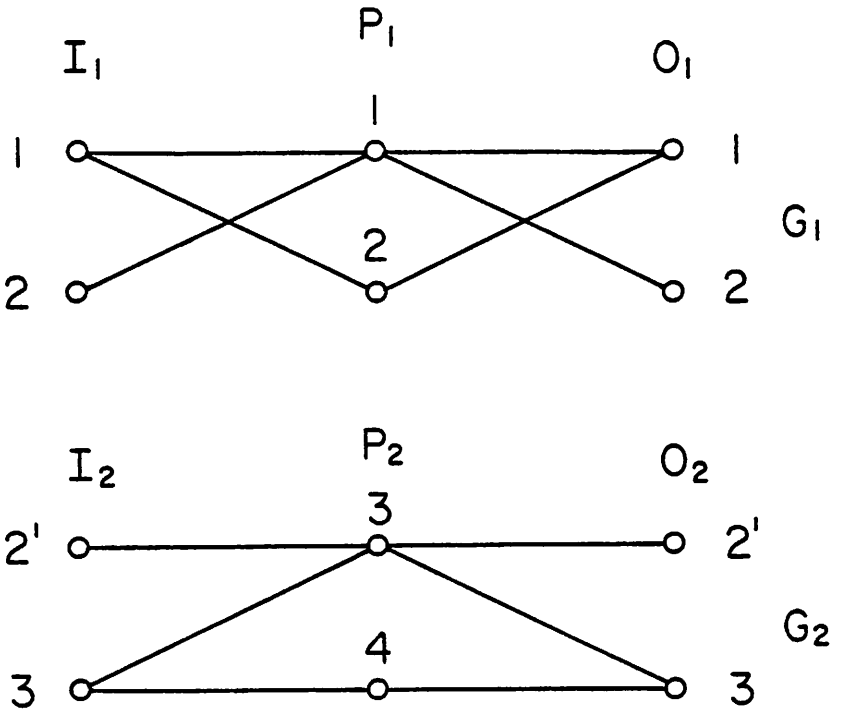

Fig. 4.5a G_1 and G_2 of parallel-partitioned PLA (without product augmentation)

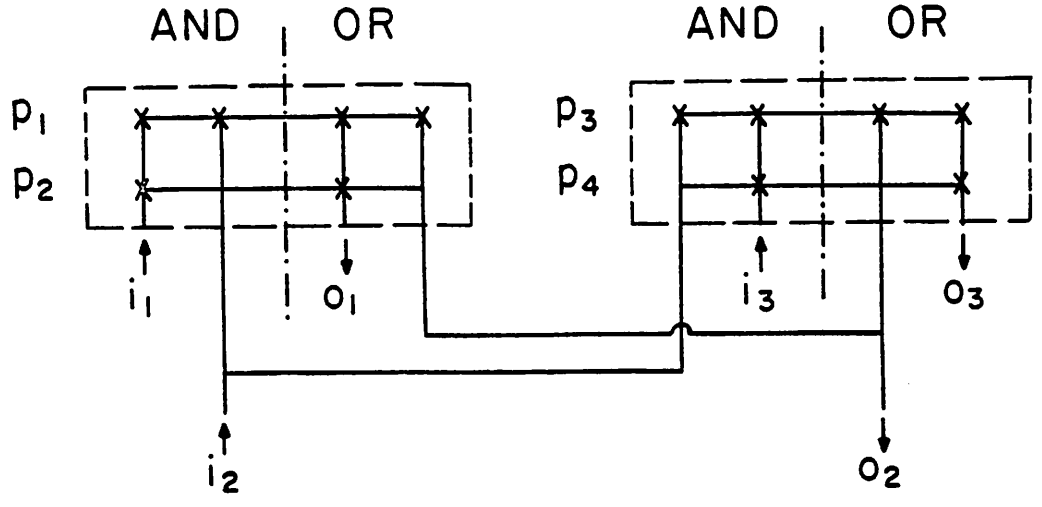

Fig. 4.5b Parallel-partitioned PLA (without product augmentation)

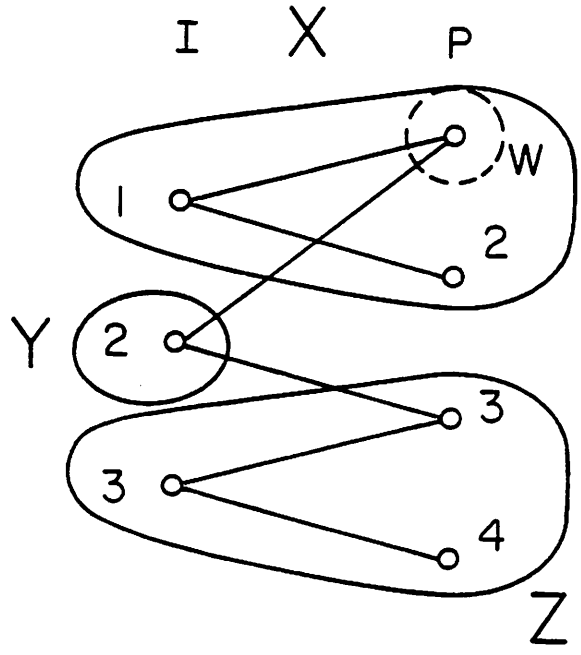

Fig. 5.1 Node sets generated by cluster algorithm

```
 1        0000 1000101010101010
 2        0001 0000101000001000
 3        0010 0100000010001000
 4        0011 0001010100010101
 5        0100 0010100000101000
 6        0101 0000010000010000
 7        0110 0000100000101010
 8        0111 0000001010100010
 9        1000 0101010101000100
10        1001 0100010100000001
11        1010 0100000100100001
12        1011 0101010100100101
13        1100 0101000100010001
14        1101 0101000100000100
15        1110 0100010000010100
16        1111 1010000000101010


                              1111111
                  1234 1234567890123456
```

Fig. 6.1a TPM of a benchmark PLA (PLA 2) before partitioning

BLOCK FOLDED OR PLANE

```
                          000002
                         68024680

  * 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  57913579
```

Partitioned PLA takes 71% of the original area

Fig. 6.1b TPM of a benchmark PLA (PLA 2) after output-partitioning

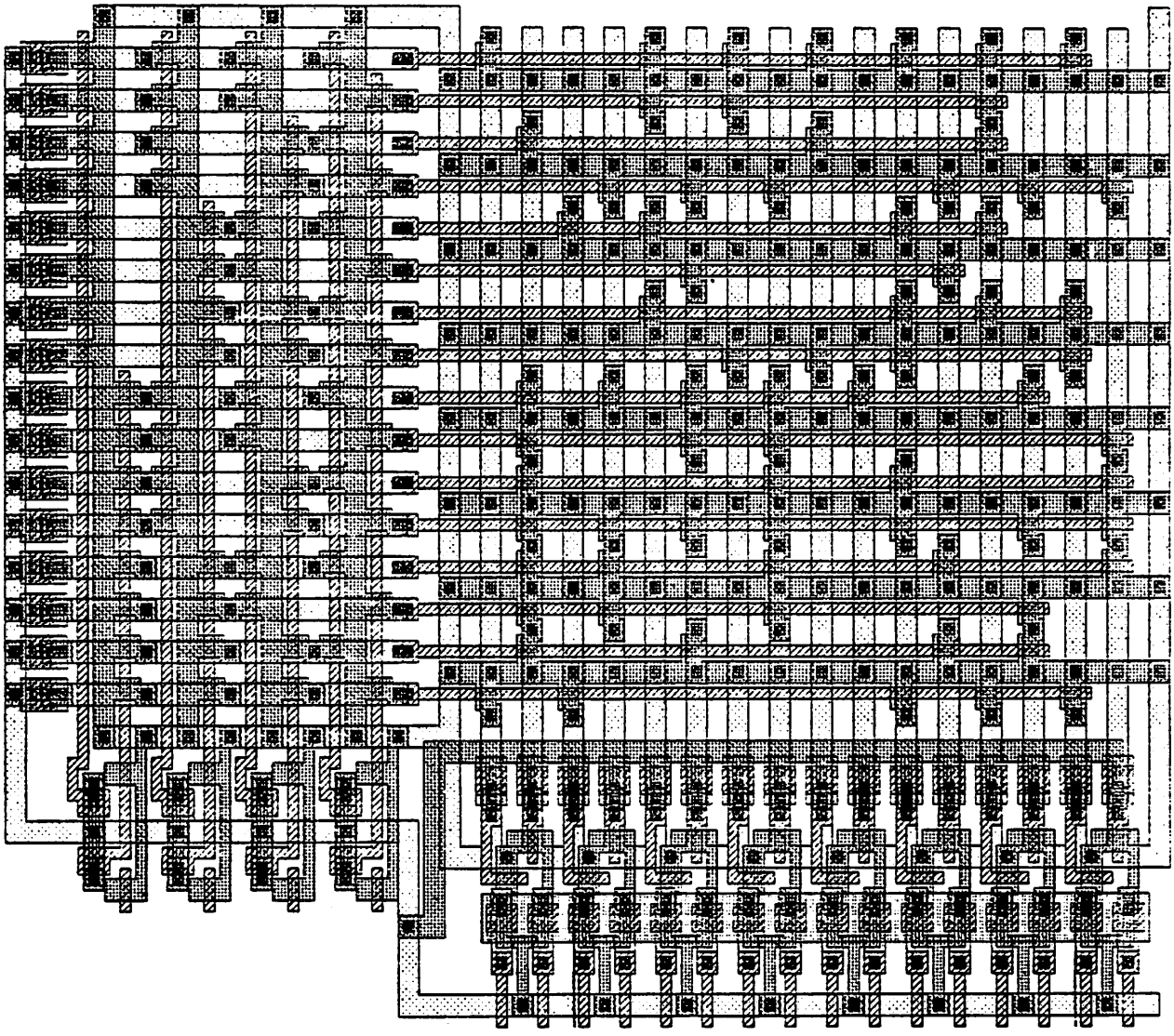

Fig. 6.2a Checkplot of a benchmark PLA (PLA 2) before partitioning

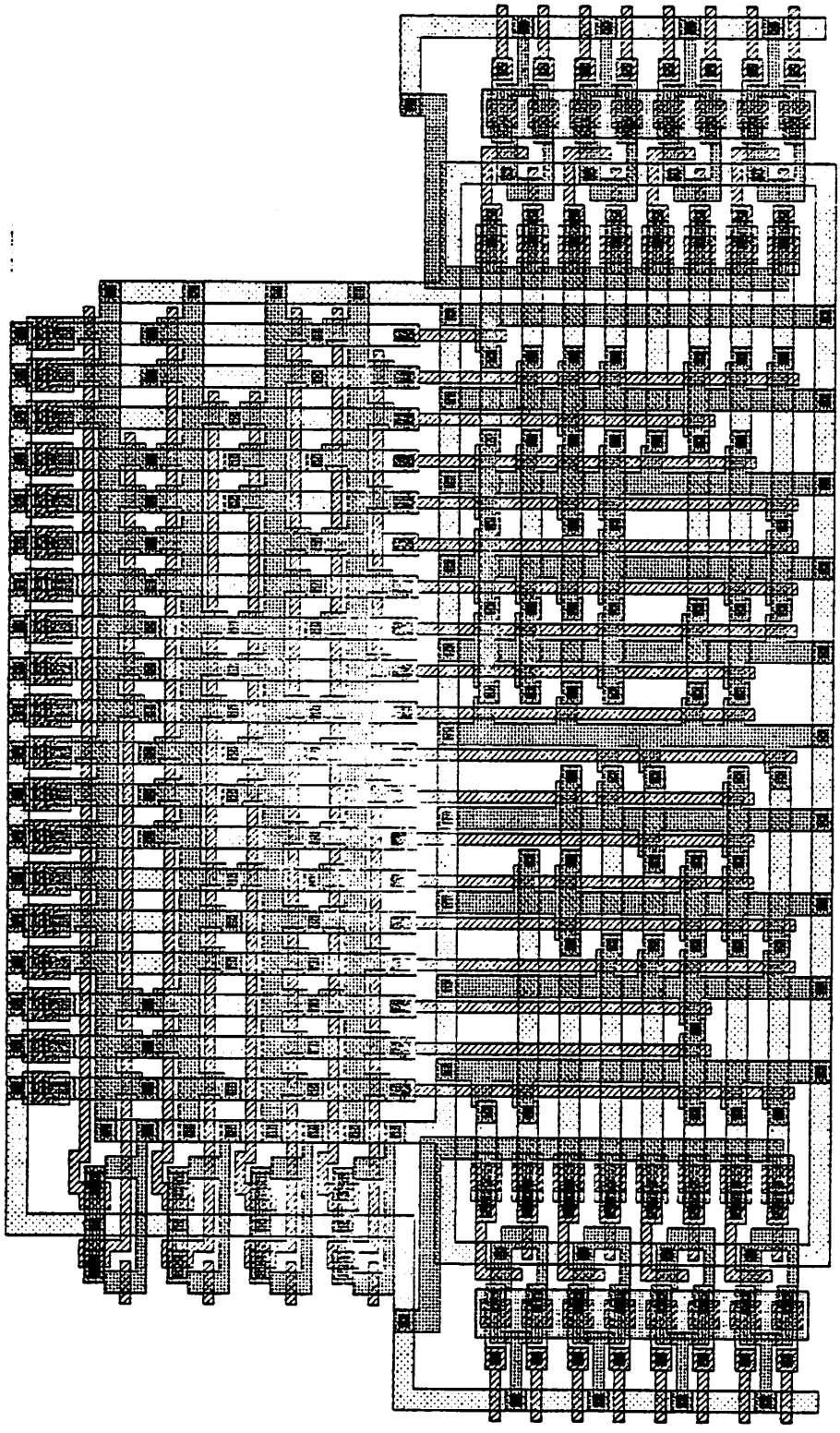

Fig. 6.2b Checkplot of a benchmark PLA (PLA 2) after output-partitioning