TOPOLOGICAL PARTITIONING OF PROGRAMMABLE LOGIC ARRAYS

Giovanni De Micheli and Mauro Santomauro
Department of Electrical Engineering and Computer Science
University of California at Berkeley

Abstract: We present a new approach to optimal topological design of Programmable Logic Arrays. In particular we address the **array partitioning** problem and the implementation of partitioned arrays as **block folded** or PLAs connected in **parallel.** We present an efficient heuristic algorithm based on a graph theoretical interpretation of the PLA partitioning problem. A computer program, **SMILE** , is described and experimental results are reported.

1. INTRODUCTION

Programmable Logic Arrays (PLA) are used extensively in the structured design of Very Large Scale Circuits (VLSI) [1]. Multiple-output switching functions are effectively implemented by PLAs, because they show a regular structure and can be designed and optimized with the support of computer aids.

We consider Programmable 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. 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 and are represented by **Topological Personality Matrices,** i.e. 1-0 matrices whose $(i,j)^{th}$ entry is "0" if the $(i,j)^{th}$ location of the physical array is occupied by interconnect only.

We address here the optimal topological design of logic arrays: i.e. the problem of optimizing the silicon area taken by large PLAs by rearranging the array structure [2],[3],[4]. In particular we describe a general method for compacting a logic array based on **partitioning** .

There are two motivations in exploiting partitioning techniques for PLA topological design. The first is to explore and compare a topological-compaction technique alternative to folding [2],[3]. As an example, partitioned arrays can be implemented by block folded structures [5]. The second is to provide means of designing PLAs partitioned in subarrays of bounded size in order to satisfy some technological constraints. In particular, since timing delay through an array grows with the array size, it is often necessary to set an upper bound on the physical array size and partition the original array into sub-units, which satisfy the array-size bound.

PLA partitioning does not exclude folding. In particular, partitioned arrays can be folded to achieve a further array compaction. For example, Suwa [6] presented a technique for partitioning a PLA into segments and implement it as a row-segmented column-folded array.

Kang [7] proposed for the first time a general approach to PLA partitioning and a related heuristic algorithm. We proposed in [4] a graph representation of the PLA partitioning problem, which allows to state formally the optimal PLA partitioning problem. We present here a partitioning algorithm based on a **node cluster** search in the PLA graph representation and on **array transformations** and we show how partitioned arrays can be implemented.

2. EQUIVALENT ARRAYS AND PARTITIONING

Array partitioning techniques attempt to achieve a PLA implementation in a minimal area, by using array connectivity information in exploiting the array sparsity.

A logical array (or a plane of a logical array) is **connected** if no subset of the rows of the topological personality matrix (input/output plane topological personality matrices) have non-zero entries in a column subset only. It is obvious that disconnected arrays can be implemented by an appropriate connection of smaller arrays. It is also easy to show [7] that such an implementation is always convenient in terms of silicon area. However a good logical design of a switching function seldom leads to a disconnected array. On the other hand, several PLAs show weakly-connected structures. In this case, our PLA partitioning technique allows to transform the array into an equivalent disconnected one, which has a convenient implementation in terms of silicon area.

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 and/or columns [7], or by rearranging the personality matrix of the array by a reshape of the logic function.

A general equivalence transformation based on row (column) **augmentation** is described in [4]. The augmentation of an input, output or product-term, is the substitution of the input, output column or product-term row with a set of input, output columns or product-term rows which gives an equivalent logic array. Three rules for augmenting the rows and/or columns of a PLA are reported in [4]. An array (input/output array) can be transformed into an equivalent disconnected one by partitioning the row set into disjoint subsets and by replacing each column having "1"s in different subsets by a set of columns, each having "1"s in a row subset only and whose component-wise conjunction is the original column. Similarly an array (output array) can be transformed into an equivalent disconnected one by augmenting appropriate product-term rows, as described in [4].

It is clear that there are many different possible augmentation strategies. Array partitioning requires the determination of an optimal sequence of augmentations, in order to obtain area-effective partitioned array implementations.

3. A CLUSTERING ALGORITHM FOR PLA PARTITIONING

Three different PLA topological partitioning problems are dealt with here, namely: **input-plane partitioning, output-plane partitioning** and **partitioning** . A graph representation of the problem is useful to understand the underlying structure and to develop heuristics for the related algorithms. The input plane (output plane) of a PLA is represented by a bipartite graph $G^A(I,P,E^A)$ ($G^B(P,O,E^B)$) whose adjacency matrix is: $\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$ ($\begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix}$), where A and B are the topological personality matrices of the AND and OR plane respectively. The whole logic array is therefore represented by the union of such graphs, i.e. the tripartite graph $G(I,P,O,E)$, where $E=E^A \cup E^B$. 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. Since the search for loosely-coupled sets is performed in a similar fashion for the three problems, the graph will be referred to as $G(V,E)$ in the sequel for the sake of simplicity.

The optimization problems arising from PLA partitioning are stated formally in [4] and related to the graph representation. The three problems require to minimize a nonlinear function (partitioned array area) subject to integer constraints (equivalence of the partitioned array to the original one). Since this problem is hard to solve, a heuristic algorithm based on a cluster search and on array transformations is reported.

The rationale of the algorithm is the following. The algorithm attempts first to find a node cluster inside $G(V,E)$ and then partitions V into two subsets V_1 and V_2. The former contains the cluster nodes and the latter the remaining ones. If the node partition induces a graph partition into two disjoint subgraphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, the partition corresponds to the array partition. Else, the array is transformed by augmenting rows or columns according to the rules stated in [4] (i.e. adding appropriate nodes to the graph) until the edge set 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. 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 cluster search in $G(V,E)$ is based on the **contour tableau** approach described in [8]. 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 in this particular case its entries are the values of the partitioned array area estimates. An area estimate can be easily obtained from the knowledge of subgraphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ as shown in [4].

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 C.

PARTITIONING ALGORITHM

```
while ($V \neq \phi$) {

        $IS = \phi$ ; $AS = \phi$ ; $OF = \phi$ ;
        $i = 1$;
        $IS(i)$ = INSELECT [$V$];
        $AS(i)$ = ADJ [$IS(i)$];

        while ( cluster criterion not satisfied ) {

                $IS(i+1)$ = NEXTSELECT [$AS(i)$];
                $AS(i+1)$ = NEXTADJ [$IS,AS(i)$];
                $i = i+1$;
        }:

        $G(V,E)$ = UPDATE [$G(V,E)$];
        };
}.
```
■

The algorithm constructs step-by-step a cluster set $X = \bigcup_{j=1}^{i} IS(j)$. Procedure **INSELECT** $[V]$ selects the initial cluster node. The cluster set X is increased by adding adjacent nodes to it. Procedure **ADJ** $[i]$ returns the nodes adjacent to node i. In particular **ADJ** $[IS(1)]$ returns the nodes adjacent to the initial node. Procedure **NEXTADJ** $[IS, AS(i)]$ returns all the nodes adjacent to node $IS(i+1)$ not contained in the cluster set X. Procedure **NEXTSELECT** $[AS(i)]$ selects the next iterating node in $AS(i)$ according to a heuristic criterion described in the sequel. Procedure **UPDATE** $[G(V,E)]$ stores subgraph $G_1(V_1, E_1)$ and returns subgraph $G_2(V_2, E_2)$, where $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are defined according to the partitioning problem and the augmentation strategy required. The strategy follows the augmentation reported in [4].

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

$$|AS(i)| = 0$$

$$\text{block area} \geq \text{maximum block area}$$

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

The first condition guarantees that a cluster is found if $G(V,E)$ is not connected. The second condition allows the user to define the maximum size of each block according to the technological constraints on the implementation of the partitioned array. The third condition is a heuristic rule for determining a cluster. It can be also required that $OF(i)$ be 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.

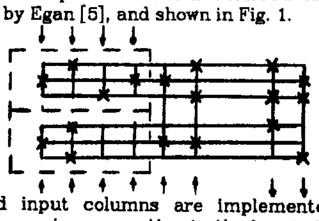
Procedure **NEXTSELECT** uses a greedy strategy to select the next iterating node among the nodes in $AS(i)$. When any node in $AS(i)$ is added to the cluster node set X, $G(V,E)$ can be partitioned according to the augmentation rules and the corresponding value of the objective function be computed. 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 [8], 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.

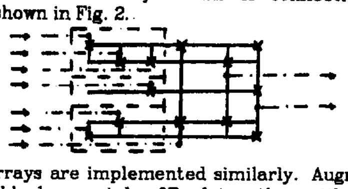
It is shown in [4] that the time computational complexity of the algorithm is polynomially bounded , though the total number of nodes may increase at each iteration.

4. PARTITIONED PLA IMPLEMENTATIONS

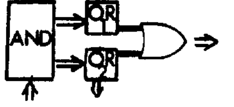
Different implementations of partitioned PLAs are possible. Two-fold partitioned input-arrays can be implemented as simply column block-folded arrays. This implementation is referred to as **bipartite folded** implementation by Egan [5], and shown in Fig. 1.

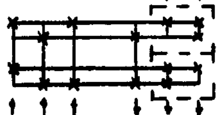


Note that augmented input columns are implemented as unsplit columns and therefore require connection to the input signal line from one side of the array only. The column positions in each block are arbitrary and can be assigned to route optimally the PLA-input signal lines from the other circuit blocks. Input arrays partitioned into more than two blocks can be implemented by a multiply column block-folded array, where inputs are routed by means of connection rows as described in [3] and shown in Fig. 2.

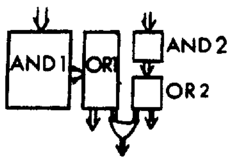


Partitioned output arrays are implemented similarly. Augmented outputs from different blocks must be OR-ed together with the proper phase (Fig. 3a). In MOS NOR implementation, the column segments corresponding to the augmented columns can be connected by a wired-NOR by simply stacking them on top of each other (Fig. 3b).

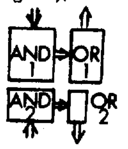


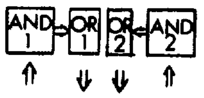


Partitioned arrays can be implemented as connected in parallel, as shown in Fig. 4. In this case the component-PLAs are placed to simplify the routing between them [9].



In particular, partitioned arrays can be stacked or arranged in a line. The former implementation is similar to a column block-folded implementation (Fig. 5a), the latter to a row block-folded one (Fig. 5b).





5. SMILE

Smile is an interactive program for Programmable Logic Arrays partitioning. The PLA description is given as input to the program in the form of personality matrix along with the partitioning instructions. The program performs input and output augmentations by default. In the case of output partitioning, product augmentations can be allowed. The user can require to limit the number of clusters, i.e. 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. This matrix is suitable to be processed by a *silicon assembler*, which generates the mask layout of the array according to a given technology. Examples are reported in [4]. Note that the partitioned PLA structure generated by *smile* is technologically independent.

6. EXPERIMENTAL AND CONCLUDING REMARKS

We tested program *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. Since execution time is small, circuit designers may want to use the program with different requirements in order to compare the different partitioned structures.

Table 2 compares the area taken by a partitioned array and implemented as a block folded array to the area taken by the same array when folded by *pleasure* [3]. The results show that PLA topological partitioning is a viable tool for optimal PLA design.

7. ACKNOWLEDGEMENTS

This research was partially supported by IBM corporation, NSF under subcontract #392741C-1 and Consiglio Nazionale delle Ricerche, Italy.

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] 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

[3] G.De Micheli and A.L.Sangiovanni Vincentelli "PLEASURE: a computer program for simple and multiple constrained folding of Programmable Logic Arrays" *Proc. Des Aut. Conf.* Miami Beach, Jun 83.

[4] G. De Micheli and M. Santomauro "Smile: A Computer Program for Partitioning of Programmed Logic Arrays" *Computer-Aided Design* vol 15 No. 2 Mar 1983

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

[6] 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

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

[8] 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

[9] G.D. Hachtel *Private communication*.

NORMALIZED PARTITIONED ARRAY AREAS. INITIAL AREA = 100.			
PLA SIZE P*(I+O)	INPUT PARTITIONING	OUTPUT PARTITIONING	PARALLEL PARTITIONING
6*(6+4)	71	64	61
16*(4+16)	100	71	66
30*(19+10)	78	81	67
75*(35+29)	75	70	46
62*(24+14)	75	80	60
84*(27+10)	71	81	59
84*(27+10)	69	81	57

TABLE 1

NORMALIZED FOLDED VERSUS PARTITIONED ARRAY AREAS. INITIAL AREA = 100.		
PLA SIZE P*(I+O)	FOLDING	PARTITIONING
6*(6+4)	60	61
16*(4+16)	60	65
30*(19+10)	68	67
75*(35+29)	53	46
62*(24+14)	58	60
84*(27+10)	54	59
84*(27+10)	51	57

TABLE 2