MULTI-LEVEL PROGRAMMABLE ARRAYS FOR SUB-MICRON TECHNOLOGY BASED ON SYMMETRIES

MAREK PERKOWSKI, MALGORZATA CHRZANOWSKA-JESKE and YANG XU

Portland State University, Dept. of Electr. Engn., Portland, Oregon 97207, Tel: 503-725-5411, Fax: 503-725-4882, mperkows@ee.pdx.edu

Regular layout is a fundamental concept in VLSI design which can have applications in custom design for submicron technologies, designing new architectures for fine-grain Field Programmable Gate Arrays (FPGAs) and Electrically Programmable Logic Devices (EPLDs), and minimization of logic functions for existing FPGAs. PLAs are well-known examples of regular layouts. Lattice diagrams are another type of regular layouts that have been recently introduced for layout-driven logic synthesis [14]. In this paper we extend and combine these two ideas, by introducing the multi-level PLA-like structures, composed from multi-output (pseudo)symmetrical lattice planes and other planes(multi-input, multi-output regular blocks). The main idea is to decompose a non-symmetric general function to planes, in order to realize as much as possible of the function with totally symmetric and regularly connected planes.

1 Introduction.

The concept of a cellular array to realize logic is an old one, but so far, only the PLA-like structures (including EPLDs and Complex PLDs - CPLDs) have achieved successful status commercially. In the past, **three types of regular structures** have been proposed: **(1)** PLA-like based on a rectangular grid [6,11,12,13,18,21]. **(2)** Based on binary trees [7,20,22,23]. **(3)** Lattice diagrams [1,3,4,9,16,14].

In **PLA-like structures**, every cell has four neighbors in a rectangular grid. One can observe that in these structures there are two types of functions (planes): (1) the subfunction generators (such as the AND plane in PLAs). (2) the collectors of subfunctions (such as the OR plane in PLAs). The investigations of improved PLA-like cellular structures have gone into the following directions: **(1)** Replacing the OR gates in the collecting plane with other kind of gates (for instance an EXOR instead of an OR in [6,19]). **(2)** Replacing the AND gates in the subfunction-generating plane with more powerful columns realizing sequences of arbitrary two-input gates (such as for instance the generalized Maitra terms [18,21] or orthogonal functions [10,11]). **(3)** Adding function generators on the inputs to the generating plane [19]. **(4)** Adding more complex generating planes, or more levels of generating planes with the simplest possible gates, such as in TANT networks. **(5)** Combining 3 and 4, such as in Multi-Valued TANT networks [12]. **(6)** Adding vertical and horizontal buses for cells, as in Concurrent Logic's (Atmel's) Architecture [2,8]. **(7)** Creating more powerful cells as in Motorola, Atmel or Xilinx 6000 architectures [2,8]. **(8)** Adding more connections to neighbors [2,8]. All these ideas proved to be practically useful and contributed to the concept of the new designs.

Binary tree based regular structures were discussed in [7,20,22,23]). The disadvantage of these structures is fast growth of tree width for some functions and irregularity of trees in practical problems. Such structures were investigated for Binary Decision Diagrams, Permuted Reed-Muller Trees, and Kronecker Decision Diagrams.

A well-known switch realization of symmetric function, presented in the classical textbook by Kohavi (Fig.1a) can be a starting point to derive three structures: **Universal Akers Array(UAA)** [1], **Lattices** [14], and the new **Multi-Output Pseudo-Symmetric Array (MOPS)** and hierarchical MOPS PLAs introduced below.

The approach of Akers from 1972 was the first in the literature to propose lattices. It can be treated as an attempt to combine the properties of PLA-like and tree-like structures. Although based on rectangular grid similar to PLAs, the UAAs used multiplexer cell, allowing to use Shannon expansions, and thus UAAs were similar to tree expansions. To maintain the regularity of arrays for non-totally symmetric functions the variables were repeated many times. The universal construction of Akers basically created trees inside a square array, by repeating consecutively the same variables. Thus, cancelling all path $a\ \bar{a}$ transformed a directed acyclic graph to a tree. Aker's method, however, was very wasteful, leading to large arrays for **all** functions. No efficient procedures for finding the order of (repeated) variables were given, and it is easy to show simple functions that have very large UAAs. Besides, Akers in [1] presented also many other interesting symmetry-based properties of non-universal realizations of single and multi-output functions, that can be utilized to develop new design methods and efficient computer algorithms.

Figure 1 presents the derivation of both UAAs and Lattices from switch realization of a symmetric function. Figure 1a shows the multiplexer subcircuits (oval loops) drawn on the switch realization. Each multiplexer has an (input) *control variable*, and two *data inputs*, selected by a state 0 or 1 of the control variable. For variable b the data inputs 0 and 1 correspond to $\bar{b}$ (denoted as b') and b, respectively. This circuit leads directly to Universal Akers Array connection structure and layout, but not necessarily to Akers' way of repeating variables and synthesizing functions. The lattice from Figure 1b is derived from the circuit in Figure 1a. Circles with + are OR gates, circles with * are AND gates. Small circles are negations. Observe that every

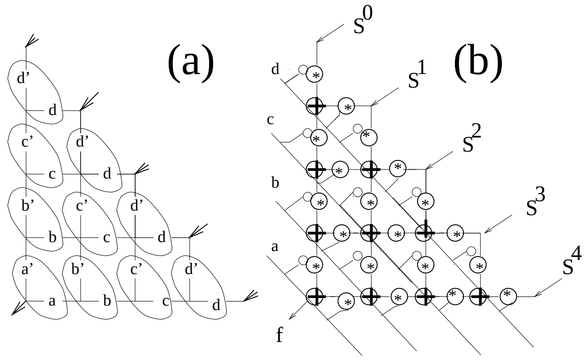

Figure 1: *Derivation of Universal Akers Arrays and Lattice Diagrams from switch realization of symmetric functions: (a) switch array with multiplexers grouped for UAAs, (b) structure of UAAs and Lattice Diagrams derived from the switch array in (a).*

variable from a diagonal bus goes to one negated AND and one non-negated AND of a multiplexer with OR gate as its output. Each signal S^i, corresponding to a *symmetry index* of the function, is set to a Boolean constant. Each multiplexer in the lattice obtains one data input from North and one from East, and directs its output to South and West. Diagonal lines are used for inputs (control variables of multiplexers). UAA has the same structure and gates, but has a square shape of area $(2^{n-2}+1) \cdot (2^{n-2}+1)$ cells, and a total of $(2 \cdot 2^{n-2}+1)$ repeated variables on the diagonal. Thus, for instance every function of 4 variables can be realized in a 5 * 5 square with variables d,d,c,a,b,$\bar{a}$,$\bar{c}$,$\bar{d}$,$\bar{d}$ on a diagonal. Akers proved that every single-output binary function can be realized with such a structure, but an exponential number of levels was necessary (which means, the control variables in diagonal buses are repeated so many times that the structure becomes completely impractical for functions with more than 6 variables).

The remaining of this paper is structured as follows. Section 2 introduces Lattice Diagrams and illustrates how to realize incomplete multi-output functions in them. Section 3 analyzes reasons of inefficiency of multi-output lattices. MOPS arrays and new MOPS-based PLAs for symmetric functions are introduced in section 4. Section 5 generalizes these architectures for arbitrary incompletely specified multi-output functions, and section 6 concludes the paper.

2 Lattice Diagrams, Lattice Arrays and Regular Layouts

Because of the progress in hardware and software technologies since 1972, our approach is quite different from that of Akers. We do not want to design a universal array for **all** functions, because such array would be very inefficient for **nearly all** functions. Instead we create **layout-driven logic function generators** which give efficient results for **many** real-life functions, not only symmetrical ones. We argue that there is no need to realize the "worst-case" functions, since it was shown in [17] that, in contrast to the randomly generated "worst-case" functions, 98% of functions from real-life are decomposable by Ashenhurst/Curtis type of decomposition. Therefore, the "other" functions are either decomposable to the easy realizable functions, or they do not exist in practice [15,17].

We assume layout to be a two-dimensional shape such as a trapezoid with horizontal parallel lines. Let us assume for a moment that outputs go vertically and input variables come in horizontal buses (as in Fig. 2b). Then the "vertical growth" of layout is increasing the depth of the circuit, and the "horizontal growth" increases the circuit area in the other dimension (breadth). We analysed vertical and horizontal growth in various realizations of arithmetic, symmetric, unate, "tough", and standard benchmark functions in lattices, PLAs and trees. Based on investigations of new commercial technologies [2,8] and discussions with their designers we established these and other bottlenecks in the regularly structured designs. Finally, we analysed the cases that Lattice Diagrams gave poor solutions. We found that the multi-output lattices [14] differ significantly from the single-output ones [3]. Based on all of above, we arrived at substantially generalized concepts of lattices and related PLAs. Although for simplicity we will use here mostly the completely specified multi-output functions, all our new methods are also for incompletely specified functions. Moreover, the more unspecified is the function, the better are the results. (We extended the new methods also to the input data specified as multi-output Boolean relations [15]. A function or relation is decomposed to relation blocks, and next each of the relation blocks is realized in hardware as the simplest function corresponding to this relation.)

Definition 1. The *Producted Cofactor* f_{prod} of function f is an intersection of product of literals $prod$ and function f. It is created as follows: $ON(f_{prod}) = ON(f) \cap prod$, $OFF(f_{prod}) = OFF(f) \cap prod$.

Observe, that while calculating f_{prod}, all minterms outside $prod$ become don't cares. Producted Cofactor $f_{a\bar{b}}$ is equivalent to standard cofactor $f_{\underline{a}\bar{b}} = f(a = 1, b = 0, c, d, ..)$ inside product $a\bar{b}$, but is unspecified outside $a\bar{b}$, in contrast to the standard cofactor. Note that although these two concepts are

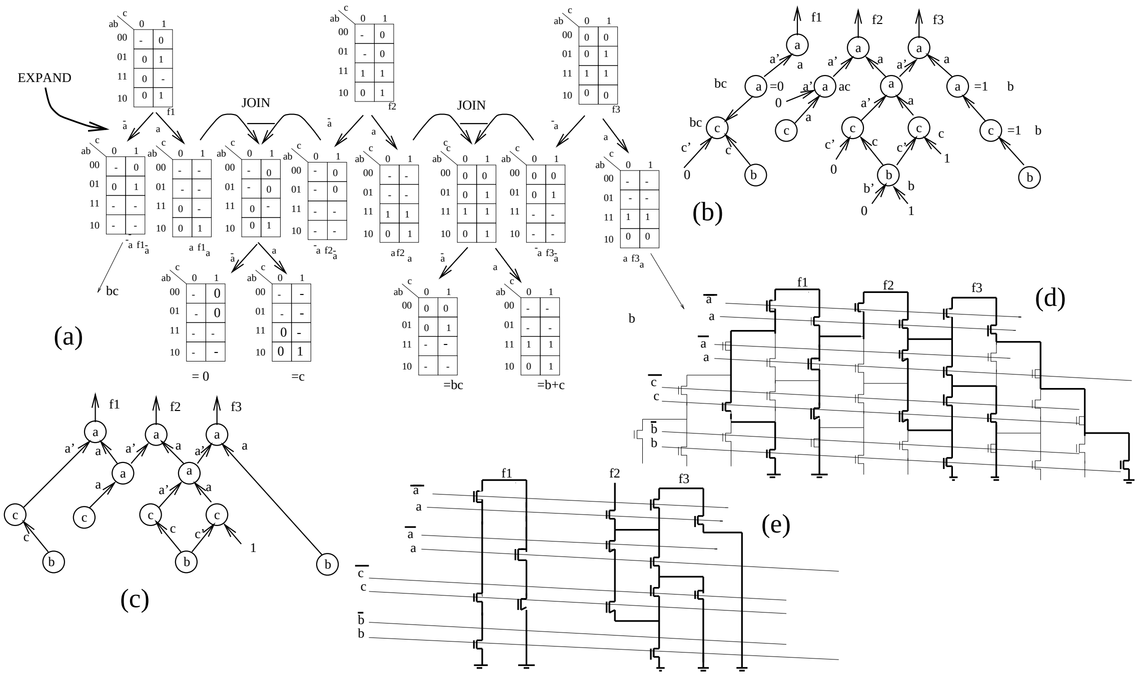

Figure 2: *Creation of Multi-Output Ordered Shannon Lattice for 3-input, 3-output function* $(f1, f2, f3)$. *(a) the illustration of the method to create the expansions and of joining the non-isomorphic nodes in first three levels of the lattice, (b) the complete lattice diagram derived using the method from Fig. 2a, (c) its realization in 2x2 Ordered Shannon lattice, (d) pass-transistor layout realization obtained directly from the lattice in Fig. 2c, (e) pass-transistor layout from (d) after logic/layout simplifications and compaction.*

similar, they are not the same.

Figure 2 presents the method to calculate the Ordered 2x2 Shannon Lattice Diagram for a multi-output, incomplete function. Ordered means the same order of variables and one variable per level. 2x2 means each cell has at most 2 predecessors and at most 2 successors in a regular planar grid. Because function's outputs $f1, f2, f3$ are initially placed in the closest proximity [14], we call this a *"Distance-1 Lattice"* type of realization of a multi-output function. Creating the lattice is based on calculating the producted cofactors in expansions, starting from the function outputs, and next joining some nodes together to produce a regular level of the lattice. Kmaps are used to illustrate calculating producted cofactors and combining non-isomorphic nodes on the bottom of the lattice. In Figure 2a, only the first three levels are shown. The nodes are of two types, S type nodes realize the Shannon expansion, S' nodes realize the "rotated" Shannon expansion, it means an expansion with negated control variables. Example of S' expansion application is in the second from left node of the second row in Fig. 2b. Observe in Fig. 2a that this method creates incomplete subfunctions in lattices, even starting from complete functions. The figure shows the nodes of the lattice diagram, and how they are joined to form next levels of the lattice. As we see, at every level, more and more don't cares are introduced, which produces an improved quality of results (The method for single-output complete functions is in [4]). At every node of the diagram, the subfunction is represented by OFF and ON cube sets, or by pointers to respective ON- and OFF- BDDs. Kmaps are used here just for an ease of explanation.) Fig. 2b presents the entire lattice diagram. Shannon expansions have control variable a, and have negated data inputs from left (denoted by a' or $\bar{a}$), and positive data inputs from right (denoted by a). Rotated Shannon expansions have control variable a', and have negated data inputs from right and positive data inputs from left. By writing $a = 1$ in a node we denote that by selecting a constant $a = 1$ for control variable a we fix the right direction (corresponding to the non-negated variable), so that no expansion with respect to variable a is actually executed. By writing $a = 0$ in a node we denote that by selecting a constant $a = 0$ for control variable a the left direction was fixed, and no expansion with respect to variable a is executed. Fig. 2c presents the corresponding 2x2 lattice array realization.

Layout from pass transistors is the future of sub-micron technologies and is recently investigated by many researchers, but no papers on lattice realizations with pass transistors have appeared yet. Fig. 2d presents the pass-transistor realization obtained directly from Fig. 2c (For simplification, technology-specific details such as inverters and/or pullups are not shown. Input buses for variables and their negations are slightly skewed for better visibility only). The bold connections are the actual pass-transistor layout found. We impose this layout on the standard grid structure of pass-transistor multiplexer cells drawn with thin lines, just to demonstrate how the layout was mapped directly from the lattice diagram.

Fig. 2e presents the pass-transistor layout after logic simplifications $aa = a$ and horizontal compaction. Such layout model derived directly from Lattice Diagram data structure can be used in a custom-layout software generator. In such case, further compaction can be done by shifting in Fig. 2e function $f1$ one cell to the right. Another approach is to use this layout to develop mask-programmable Lattice-type "building block" (PGA or EPLD type) for PLA/EPLD technology (such block is a generalization of AND and OR planes used now in commercial devices).

CAD tools for lattices and other regular layouts are presented in more detail elsewhere [4,3,5,14,18,21]. In short, the Lattice for a multi-output function is expanded level-by-level, and from left to right in every level. Usually, this means combining some non-isomorphic nodes of trees. If only one non-constant successor of a node exists, it is not combined with the neighbor. For nonsymmetric functions, this procedure means repeating some variables in lattices. The variable and expansion type controlling heuristics serve to avoid too many repetitions, and also to create as few as possible branches of the lattice. The effort is made to complete every branch of the lattice as soon as possible. We have implemented a set of algorithms for generating Lattices in the C language which runs in the UNIX environment on SPARC workstations [4,3,14]. It can be easily seen that for the real-life functions we have generated uniform and simple Lattices, which are the most restrictive representation from the family of the regular layouts we will introduce in this paper, with the reasonable number of nodes and levels. Comparing a number of function variables and a number of levels it can be seen that for the worst case function in this set of benchmarks the average number of time a variable is repeated is three. It is even more important that for the majority of tested function the **average repetition is close to two.** Our worst total number of levels was 19 (misex64.esp), while it would be $\approx 2^{11}$ in Akers method for the same function. We have thus demonstrated that the lattice representation of a **single-output** function leads to practical solutions and that the size of that representation could be very attractive especially for technologies limited by the interconnections delay.

3 Partitioning Lattices for Multi-Output Functions

Although every single- and multi- output function can be realized in a single lattice with repeated variables, for some functions the solutions are quite wasteful. It can be shown that even for symmetric multi-output functions it is necessary to repeat variables when the functions are in "Distance-1 Lattices". (An example of such a "Distance-1 Lattice" where functions are realized together and in the closest geometrical distance is shown in Fig. 2b). So, may be each function should be realized separately? The problem is this. If output functions are defined on only a subset of variables each, especially disjoint subsets, mixing the positive and negative cofactors of functions, as shown in Fig. 2, can often only worsen the case by creating more complicated subfunctions. Then, it is reasonable to realize the output functions separately. But now much of the gain of common symmetries can be lost, and big "empty" subareas are created (as shown between $S^4, S^{3,2}$ and $S^{3,1}$ in Fig. 3d, for a

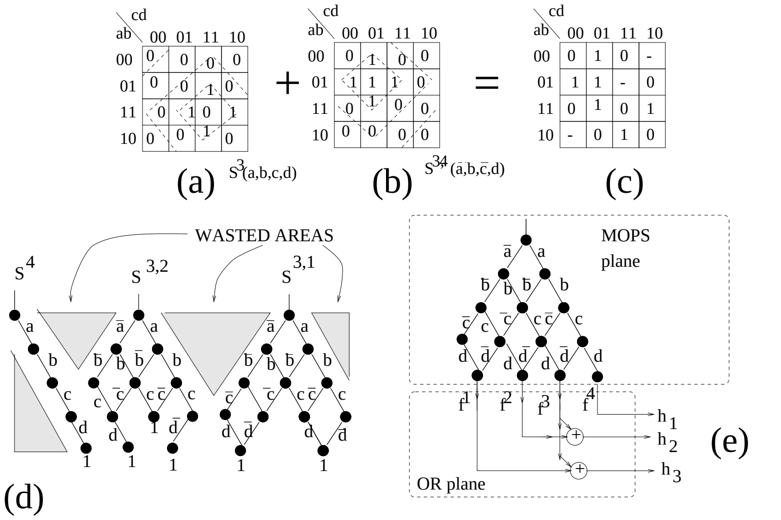

Figure 3: *(a) Symmetric function $S^3(a,b,c,d)$, (b) Symmetric function $S^{3,4}(\bar{a},b,\bar{c},d)$, (c) Sum of $S^3(a,b,c,d)$ and $S^{3,4}(\bar{a},b,\bar{c},d)$, (d) multi-output Lattice for "counter of ones", (e) MOPS/OR PLA for the "counter of ones".*

multi-output Lattice for "counter of ones" function, realized with separate realization of outputs). While the common realization of functions in Distance-1 Lattice or Distance-k lattice with small k leads in general to variable repetition and in consequence to the "vertical growth" of the layout, their independent (disjoint) realization leads to the horizontal growth of the layout. Thus, large area realizations of such functions are an argument **against** using simple lattices (see example "counter of ones" in Fig. 3).

The solution to this problem is offered by the theory of functional decomposition. It was long known that the functional (Ashenhurst/Curtis) decomposition leads to simpler and thus more manageable functions. In addition, the decomposition can be controlled to create symmetric (multi-output) predecessor function blocks [15]. Thus, "decomposed" (partitioned) regular structures, and corresponding design methods should be devised, in order to balance the horizontal and vertical growth and to optimize the area and speed for (incompletely-specified) multi-output functions.

Assuming that all variables in control inputs of a lattice level are not negated (which means, all expansions are of S type), we obtain the *simple totally symmetric functions*. Such functions are also called *totally symmetric functions of positive polarity*. They group to one level S^k all minterms that have k ones in their binary number (see Figure 3a). Because there are n variables, and each of them can be negated or not, there are 2^n different *polarities of symmetric functions*, each polarity being a binary vector of polarities of variables. If a variable is negated, the corresponding bit in the vector is 0, otherwise it is 1. Fig. 3a shows function $S^3(a,b,c,d)$ of polarity (1,1,1,1), with characteristic pattern centered in polarity cell (1111). Fig. 3b shows function $S^{3,4}(\overline{a},b,\overline{c},d)$ of polarity (0,1,0,1) with characteristic pattern centered in polarity cell (0101). Dotted lines demonstrate characteristic patterns for symmetric functions of various polarities. The polarity cell is always in the center of the pattern. An incomplete function to be synthesized, realizable as a sum of symmetric functions $S^3(a,b,c,d)$ and $S^{3,4}(\bar{a},b,\bar{c},d)$, is shown in Fig. 3c.

The concept of polarity expands thus greatly the power of symmetric functions at a very low price of adding S' expansions to the diagrams and layoyts.

It can be proved that:

Theorem 1. Every multi-output Boolean function can be decomposed to vector-OR of symmetric (multi-output) functions of various polarities

By a vector-OR we understand that each output signal is an OR of some symmetric functions from the MOPS plane. This is illustrated in Fig. 3e, where MOPS/OR PLA for the "counter of ones", a totally symmetric function of polarity (1111) is realized. In the worst case, the symmetric functions in this decomposition become prime implicants of a standard AND/OR PLA.

Theorem 2. Every multi-output Boolean function can be decomposed to vector EXOR of symmetric functions of various polarities.

Theorem 3. Every multi-output Boolean function can be decomposed to vector AND of symmetric functions of various polarities.

Theorem 4. Every multi-output Boolean function can be decomposed to a serial composition of MOPS of symmetric functions of various polarities and AND/OR PLA.

Theorem 5. Every multi-output Boolean function with subsets $SV_i, i = 1, ...K$ of mutually symmetric variables can be decomposed to a serial composition of K MOPS arrays, $MOPS_i,\ i = 1, ..., K$ followed by an AND/OR PLA (Fig. 6d). Each $MOPS_i,\ i = 1, ..., K$ realizes symmetric multi-output functions on set of variables SV_i, and feeds its outputs to the AND/OR PLA. Function F is decomposed as $F = H(G_1(SV_1), G_2(SV_1), ..., G_K(SV_K), REM)$, where REM are the remaining variables of F that did not occur in any symmetric set SV_i.

It results from[1], other references, and our tests, that many interesting functions such as counters, threshold, EXOR, parity, and arithmetic have good decomposed realizations. It can be also shown, that whenever some partial symmetries exist in the data (which is often the case), the number of the MOPS planes to realize the component symmetric functions in these decompositions

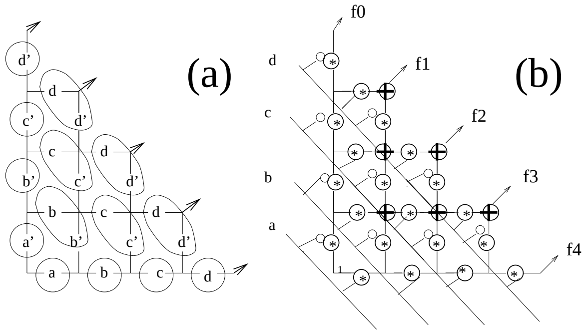

Figure 4: *Derivation of Multi-Output Pseudo-Symmetric Arrays (MOPS) from switch realization of symmetric functions: (a) switch array with multiplexers and AND gates shown for MOPS, (b) MOPS derived from the switch array in (a).*

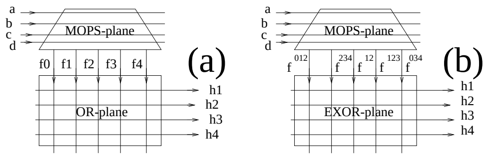

Figure 5: *Multi-Output Pseudo-Symmetric Arrays (MOPS) PLAs for symmetric functions: (a) MOPS-based PLA with OR-plane for totally symmetric functions, (b) MOPS-based PLA with EXOR-plane for totally symmetric functions.*

is much smaller than the number of prime implicants in AND/OR decomposition used in PLAs. The results are especially good for strongly unspecified functions. Thus, the goal of the next sections will be to develop architectures that will make use of these decompositional properties of real-life functions (in contrast to the randomly generated functions that are the worst case [17]).

4 MOPS Arrays and Generalized PLAs for Multi-Output Symmetric Functions

In Figure 4a we present another way of grouping switches to the oval loops representing multiplexers, and the circle loops representing AND gates. This way, a multi-output function is created, but the same connection structure as in Fig. 1a is basically preserved. Connections from diagonals are exactly the same, but the direction of signals and location of OR gates is reversed. The corresponding MOPS is presented in Figure 4b. Observe, that the whole MOPS can serve as a term generator, with terms being symmetric functions fi, corresponding to symmetry indices S^i. The final output functions are created by OR-ing function fi (Fig. 4c). Sometimes better results are obtained when ORing, EXOR-ing (Fig. 4d), or AND-ing is done on **arbitrary** symmetric functions f^I in the collecting plane, where I is an arbitrary set of indices i.

The presented decompositions lead to a Generalized MOPS PLAs in which MOPS plays the role of a generating plane, and OR, EXOR, or AND plane is used for collecting [10,11,13]. As an example, the single-plane MOPS/OR PLA has a single-polarity MOPS plane for non-repeated variables, followed by an OR plane. Such PLA can realize only symmetric functions.

"Counter of ones" is a known totally symmetric function that calculates in natural binary code the number of bits "1" among its all arguments. For instance, for four input variables, a, b, c, d, the "counter of ones" is described by the following output functions $h_1 = S^4(a, b, c, d)$, $h_2 = S^{2,3}(a, b, c, d)$, $h_3 = S^{1,3}(a, b, c, d)$.

Figure 3d,e compares disjoint realization of "counter of ones" function in an Ordered Shannon Lattice and in MOPS/OR PLA with single MOPS plane of single polarity. Observe, that because Boolean unions (ORs) of symmetric functions are symmetric, only symmetric functions can be realized in MOPS/OR PLA with single-polarity MOPS, as well as in all other MOS PLAs with only one single-polarity MOPS plane.

5 MOPS PLA Architectures for Arbitrary Multi-Output Functions

To allow realization of **arbitrary multi-output functions** we have done further generalizations to MOPS PLA from section 4:

(1) OR plane is used for summing outputs from several MOPS planes (see Fig. 6a for an example with 2 MOPS planes). Each MOPS plane realizes a symmetric function of a different polarity. Practically all such planes are realized as a single plane with the same input buses, possibly inverted inside the plane to create various polarities. Alternatively, inverters can be parts of the mux gates. Finally, both a variable and its negation can be used simultaneously in these input buses.

(2) Each MOPS plane can have different input variables (and polarities), see Fig. 6b. This requires variable folding.

(3) Variables in each MOPS can be repeated, as in standard Lattices (see Fig. 6c).

(4) Variables in each MOPS can be repeated, and AND/OR PLA is used for collecting, see Fig. 6d. This structure realizes functions decomposed according to Theorem 5.

It can be proved that every function is realizable in a generalized PLA

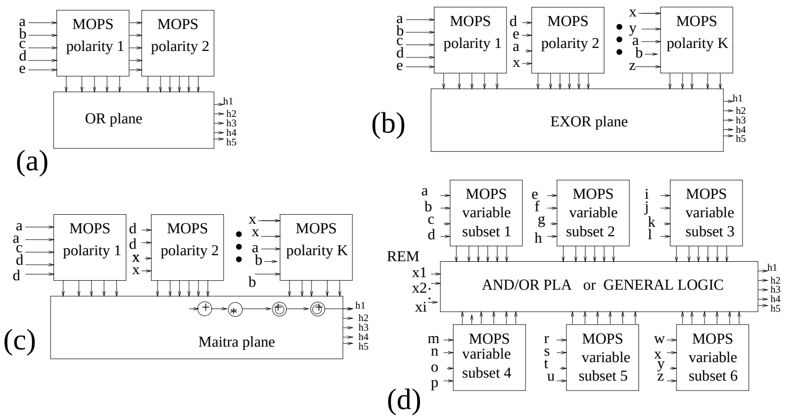

Figure 6: *MOPS PLA architectures for arbitrary multi-output functions: (a) an example of a general scheme of MOPS/OR PLA with two MOPS planes of various polarities, (b) a general scheme of MOPS/EXOR PLA with many MOPS of various polarities, and various variable sets, (c) MOPS/OR/EXOR/AND PLA with MOPS using repeated variable and Maitra functions in collecting Maitra plane, (d) MOPS AND/OR PLA with MOPS using subsets of symmetric variables and collecting AND/OR PLA according to Theorem 5.*

having only one of these generalizations, thus it is also realizable if more than one of these generalizations are applied together. Fig. 6a is a MOPS/OR PLA with two MOPS planes, each for different polarity of the same variables. Such structure can be used to implement function from Fig. 3c, decomposed to two symmetric function from from Fig. 3a,b, respectively. The same extension types as for MOPS/OR PLAs can be done for MOPS/EXOR PLAs (Fig. 6b).

A very interesting and general realizations can be obtained using a collecting plane in which each function is a Maitra cascade [21], Fig. 6c. The collecting plane is then called a Maitra plane. Every row of the Maitra plane is a Maitra function, i.e. a sequence of arbitrary two-input gates.

Another powerful decomposition is based on Theorem 5 (Fig. 6d, drawn in CPLD-like form). Each MOPS is created for a maximum group of totally symmetric variables in a function. For instance in Fig. 6d, the maximum groups were: $\{a, b, c, d\}$, $\{e, f, g, h\}$... etc. Variables $x1, ..xi$ were not in any group of symmetric variables.

In conclusion, we can state that the Generalized MOPS-based PLAs unify two former concepts of regular arrays: PLAs and Lattices; and borrow from CPLDs the concept of partitioning.

6 Conclusions

The idea of partitioned lattices is very captivating from the point of view of **submicron technologies**, because: **(1)** connections are short, **(2)** delays are equal and predictable, **(3)** late-arriving variables can be given closer to the output, and **(4)** logic synthesis can be combined with layout, so that no special stage of placement and routing is necessary (similarly as in PLAs/EPLDs).

We showed new extensions to the concept of lattice layout developed previously. These extensions produce better results, especially for multi-output symmetric functions with many outputs. Although presented examples used non-repeated input variables to MOPS, we generalized also the decomposition algorithms to **arbitrary** functions realized in any MOPS plane, which requires repeating variables in them. Thus, our algorithms for repeating variables in lattices[3,4,5,14] are still useful for the most general MOPS PLAs. However, the number of repetitions is smaller than in standard lattices, thus leading in most cases to more efficient function realizations with MOPS PLAs.

7 References

1. S.B. Akers, "A rectangular logic array," *Trans. IEEE*, Vol. C-21, pp. 848-857, Aug. 1972.
2. Concurrent Logic Inc., "CLI 6000 Series Field Programmable Gate Arrays," *Prelim. Info.*, Dec. 1, 1991,
3. M. Chrzanowska-Jeske, and Z. Wang, "Mapping of Symmetric and Partially-Symmetric Functions to CA-Type FPGAs," *Proc. Midwest'95*, 1995, pp. 290-293.
4. M. Chrzanowska-Jeske, Z. Wang and Y. Xu, "A Regular Representation for Mapping to Fine-Grain, Locally-Connected FPGAs," *Proc. ISCAS'97*, 1997.
5. B. Drucker, C. Files, M.A. Perkowski, and M. Chrzanowska-Jeske, "Polarized Pseudo-Kronecker Symmetry with and Application to the Synthesis of Lattice Decision Diagrams," *Proc. ICCIMA*, 1997.
6. J. Froessl, and B. Eschermann, "Module Generation for AND/XOR-Fields (XPLAs)," *Proc. ICCD '91,* pp. 26-29, 1991.
7. P. Ho, and M. A. Perkowski, "Free Kronecker Decision Diagrams and their Application to ATMEL 6000 FPGA Mapping," *Proc. Euro-DAC '94 with Euro-VHDL ' 94*, pp. 8 - 13, 1994.
8. Motorola MPA10XX Data Sheet, 1994.
9. M.A. Perkowski, and E. Pierzchala, "New Canonical Forms for Four-valued Logic", *Internal Report, Dept. Electr. Engn.,* PSU, 1993.

10. M. Perkowski, "A Fundamental Theorem for Exor Circuits," *Proc. RM'93*, pp. 52-60.
11. M. Perkowski, A.Sarabi, and F. Beyl, "XOR Canonical Forms of Switching Functions," *Proc. RM'93*, pp. 27-32.
12. M. Perkowski, and M. Chrzanowska-Jeske, "Multiple-Valued TANT Networks," *Proc. ISMVL'94*, May 25-27, 1994, pp. 334-341.
13. M. Perkowski, A.Sarabi, and F. Beyl, "Fundamental Theorems and Families of Forms for Binary and Multiple-Valued Linearly Independent Logic," *Proc. RM'95*, pp. 288-299.
14. M. Perkowski, M. Chrzanowska-Jeske, and Yang Xu, "Lattice Diagrams Using Reed-Muller Logic," *RM'97*, Oxford Univ., Sept. 1997, pp. 85-102.
15. M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J.S. Zhang, "Decomposition of Multiple-Valued Relations," *Proc. ISMVL'97,* pp. 13-18.
16. M.A. Perkowski, E. Pierzchala, and R. Drechsler, "Logic/Layout Synthesis for a Submicron Technology: Mapping Linearly-Independent Expansions to Fat Regular Lattices," *Proc. ISIC'97*, Sept 9-12, 1997.
17. T. D. Ross, M.J. Noviskey, T.N. Taylor, D.A. Gadd, "Pattern Theory: An Engineering Paradigm for Algorithm Design," *Final Technical Report WL-TR-91-1060,* Wright Laboratories, USAF, Aug. 1991.
18. A. Sarabi, N. Song, M. Chrzanowska-Jeske, and M. A. Perkowski, "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays," *Proc. DAC '94,* June 1994, pp. 321 - 326.
19. T. Sasao (editor), "Logic Synthesis and Optimisation," *Kluwer,* 1993.
20. I. Schaefer, and M. Perkowski, "Synthesis of Multi-Level Multiplexer Circuits for Incompletely Specified Multi-Output Boolean Functions with Mapping Multiplexer Based FPGAs," *IEEE Trans. on CAD,*, Vol. 12, No. 11, November 1993. pp. 1655-1664.
21. N. Song, M. A. Perkowski, M. Chrzanowska-Jeske, and A. Sarabi, "A New Design Methodology for Two-Dimensional Logic Arrays," *VLSI Design,* Vol. 3, No. 3-4, 1995.
22. N. Ramineni, M. Chrzanowska-Jeske, and N. Buddi, "Tree Restructuring Approach to Mapping Problem in Cellular Architecture FPGAs," *Proc. of IEEE EURO-DAC'95*, pp. 60-65, Sept. 1995.
23. L.-F. Wu, and M. A. Perkowski "Minimization of Permuted Reed-Muller Trees for Cellular Logic Programmable Gate Arrays," *2nd Int. Workshop on FPGAs*, Sept. 1992, Vienna, Austria.