LAYOUT-DRIVEN SYNTHESIS FOR SUBMICRON TECHNOLOGY: MAPPING EXPANSIONS TO REGULAR LATTICES

Marek A. Perkowski, Edmund Pierzchala, Rolf Drechsler †,

Dept. Electr. Engn., Portland State University, Portland, OR 97207

† Inst. of Comp. Sci., Albert-Ludwigs Univ. 79110 Freiburg in Breisgau, Germany,

Abstract: We introduce a concept in VLSI layout which can find applications in submicron design, quantum devices, and designing new fine-grain FPGAs. This concept is called Lattice Structure and it extends the concepts from [8] and [1,13,14,17,18]. In the regular arrangement of cells, every cell is connected to 4, 6 or 8 neighbors and to vertical, horizontal and diagonal buses. Methods for expanding arbitrary binary and multi-valued combinational functions to this layout are illustrated.

1. INTRODUCTION

This paper introduces a new approach to **layout-driven logic synthesis** of combinational functions. The fundament of our approach are **expansions** of functions, i.e. operators that transform a function to a few simpler functions. For instance, in canonical Shannon expansion function f is expanded with respect to input variable a as follows: $f = af_a + \bar{a}f_{\bar{a}}$, where $f_a = f\mid_{a=1}$, and $f_{\bar{a}} = f\mid_{a=0}$ are positive and negative cofactors of function f with respect to variable a, respectively. Tautological cofactor functions are combined to single nodes. Nodes for functions f_a and $f_{\bar{a}}$ and bus for variable a are mapped to layout, and procedure is repeated for the next input variable. This way, any single-output symmetrical binary function can be directly mapped to regular layout with 1,2,3,4,... nodes in successive levels corresponding to input variables. In our previous papers we showed how to extend this approach to **arbitrary** binary functions, not necessarily symmetrical. As shown below, **other expansions** of functions can also be used, and the expansion nodes are mapped to neighborhood structures which are more powerful than those investigated theoretically in the past [1,4], but similar to those from commercial Fine Grain FPGAs [2].

The concept of a lattice diagram involves three components: expansion, joining and geometry. **(1) Expansion** of a node function creates several successor nodes of this node. Function f corresponds to the initial node in the lattice, initially a tree. **(2) Joining** operation joins several nodes of a bottom of the lattice (this level before joinings looks like a tree). This is in a sense a reverse operation to expansion. **(3) Regular geometry**, to which the nodes are mapped, guides which nodes of the level are to be joined. Every signal in the Lattice can be treated as multi-valued (particularly, binary). A multivalued (MV) connection for logic with 2^k values can be realized by k binary wires which comprise a bus (making the lattice "fat") and encode the multi-valued signal to binary. The well-known: Fat Trees, Generalized PLAs, Maitra cascades, and Akers Arrays [1,4,18] structures are only a few special cases of this powerful concept. Our regular structure concept extends also some structures that are described in several patents of fine grain FPGAs (Motorola, Atmel [2], Plessey, Pilkington), but in addition we show constructive and efficient methods of designing discrete and continuous functions in these structures [16,17]. We showed on many examples [8,10,13,14,17,18,21] that this geometry is very powerful and more universal than the previously investigated general cellular structures. Here we will further extend and unify these notions to expansions **with more than 2 successor functions**, and **geometries with more than 4 neighbors.** In theory the lattices can be extended to any number of neighbors of a cell, but 8 is practically enough.

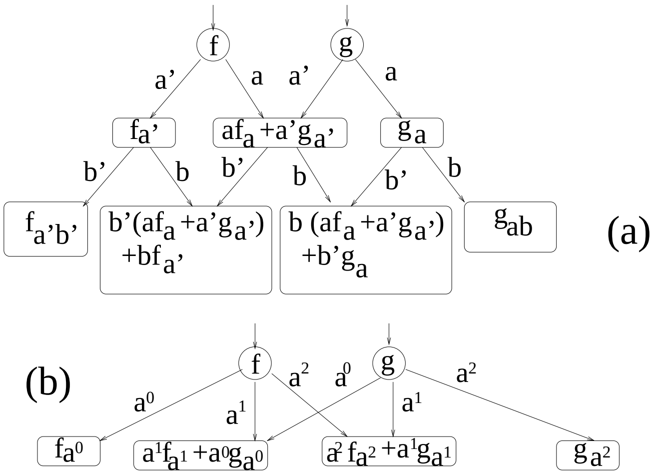

Figure 1: *Expansions and Joinings: (a) Shannon, (b) Ternary Post.*

2. EXPANSIONS AND JOININGS

The investigated by us types of expansions can be characterized as of **maximum-type**, or of **Linearly-Independent (LI) type.** Maximum-type expansions use the MAX gate (in binary logic - OR), and **disjoint** literals or subfunctions for cofactors. They include binary Shannon (S) [19] and Sum-of-Products (SOP) [15] expansions and their multiple-valued logic generalizations, such as in Post logic [10].

Below we will present only the maximum-type expansions: Shannon, SOP, and Post (MV Shannon). We assume that each binary function f is represented by a pair [ON(f),OFF(f)]. Thus all cofactors f_a for the product of literals a, are pairs: f_a = [ON(f_a),OFF(f_a)]. Fig. 1a explains the principle of creating a Shannon Lattice based on ordered Shannon expansions for a multi-output function (functions f,g). Direction of arrows shows the expansion flow. Observe that every cofactor f_a of the product a of an (in)complete function f can be interpreted as intersecting f with a and replacing all K-map cells outside product a with don't cares. A standard cofactor f_x where x is a variable does not depend on this variable. In our interpretation, though, f_x is still a function of all variables including x, but as a result of cofactoring the variable x becomes vacuous. We will call this a **vacuous cofactor**, and denote by **v-cofactor**. Thus, for any two disjoint products a_1 and a_2, the v-cofactors f_{a_1} and g_{a_2} are disjoint (observe that standard cofactors are in general not disjoint), Therefore functions f_{a_1} and g_{a_2} are in an incomplete tautology relation, and functions f and g are not changed when f_{a_1} and g_{a_2} are joined (OR-ed) to create a new function: $a_1 f_{a_1} + \overline{a_2} g_{a_2}$, as in Fig. 1a (where: $a_1 = a_2 = a$, and $\bar{a}$ is denoted as a'). This way, the entire lattice is created level-by-level, only three levels shown in Fig. 1a. Observe that functions in lattice nodes become more and more unspecified when variables in levels are repeated, and ultimately nodes become constants, which terminates the lattice creation process. This way, because every variable cuts a Kmap into two disjoint parts, arbitrary two functions f and g can always be expanded together to a Shannon lattice, with OR-ing as a join operation, provided that the same variable x_i is used in the level, and all expansions use negated literal $\bar{x_i}$ in the left, and positive literal x_i of the variable in the right. As seen in Fig. 2, arbitrary functions of the same arguments are cut in half by expansion of each variable and new functions in levels are created by rearranging the cofactors in joinings. This process can lead to a slight increase of the number of nodes in comparison with a shared OBDD of these functions. But a regular structure is created, thus simplifying layout and making delays predictable. In case when the products a_1 and a_2 are **not** disjoint, the v-cofactors f_{a_1} and g_{a_2} can, in some cases, still form an incomplete tautology of functions. When these two cofactors satisfy a tautology relation, then functions f_{a_1} and g_{a_2} can be joined (OR-ed) without changing functions f and g. Obviously, the same method works for arbitrary number of output functions.

The method to create Shannon Lattices can be easily expanded to MV Shannon expansions and associated Post Lattices for multi-output incomplete functions (see Fig. 1b). In ternary logic, each single-variable expansion cuts a function's map to three v-cofactors, and any two of them can be next recombined by a joining operation MAX - Fig. 1b. MAX is the maximum operation denoted by +. Let us observe that disjointness of literals a^0, a^1, a^2 is the fundamental condition that must be satisfied to create maximum-type lattices. It is a special case of Linear Independence of functions used in LI expansions [14,16]. In binary SOP expansions [15]

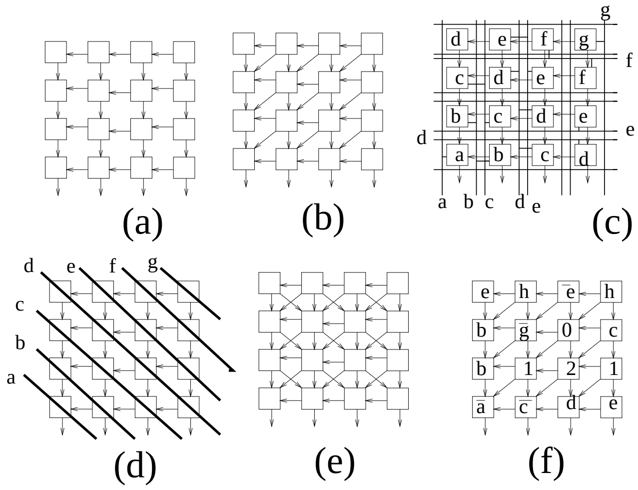

Figure 2: *Examples of regular lattices.*

a branching from node f is for any subset of literals l_j that their union covers the node function f. The SOP expansion is: $f = l_j f_{l_j} + l_r f_{l_r} .... + l_s f_{l_s}$. The method to create ordered Shannon lattices presented above can be expanded to free (non-ordered) Shannon Lattices and SOP Lattices. Any two nodes from the expansion that form an incomplete tautology can be joined as shown above. S and SOP expansion types can be mixed in levels, thus creating *"pseudo"* [19] type of lattices.

Linearly Independent expansions for binary case use EXOR gate, and are generalizations of Davio expansions [19]. For finite multiple-valued logic they are based on Galois Field Addition gate, and in general, for arbitrary algebras, they should have at least one linear (group) operation. But most often they are based on the algebraic structure of an arbitrary **field** [6,7,8,9,10,11,13,14] In particular, they include S (OR can be replaced with EXOR in Shannon expansion), Positive and Negative Davio (pD and nD, respectively), general Linearly Independent (binary and MV) [10], and EXOR Ternary expansions [9,20]. Joining operations for these expansions are more complicated and are presented in [5,8,14].

The lattice is created for each (multi-output) block obtained from Curtis-like functional decomposer [12] as follows. One level of function f is expanded to an assumed type of the Lattice for a selected variable (or a group of variables in case of LI expansion [16]. Then, the level of the tree is mapped to the assumed type of Lattice. This means joining together some nodes of the tree-like lower part of the lattice. The procedure requires repeating some variables in the lattice, the key point was thus to find good methods of variable and expansion types selections. One approach to the variable order and expansion types selection is based on generalized partial symmetries for cofactors [5]. We demonstrated that for real-life binary benchmark functions, and starting from the decompositional hierarchy of partitioning variables [12], the overhead of variable repeating in planary lattices was not excessive in each decomposed block [5,14]. This is because symmetric and nearly symmetric blocks are preferred by our decomposer [12]. We believe that good results will also be obtained for MV logic and non-

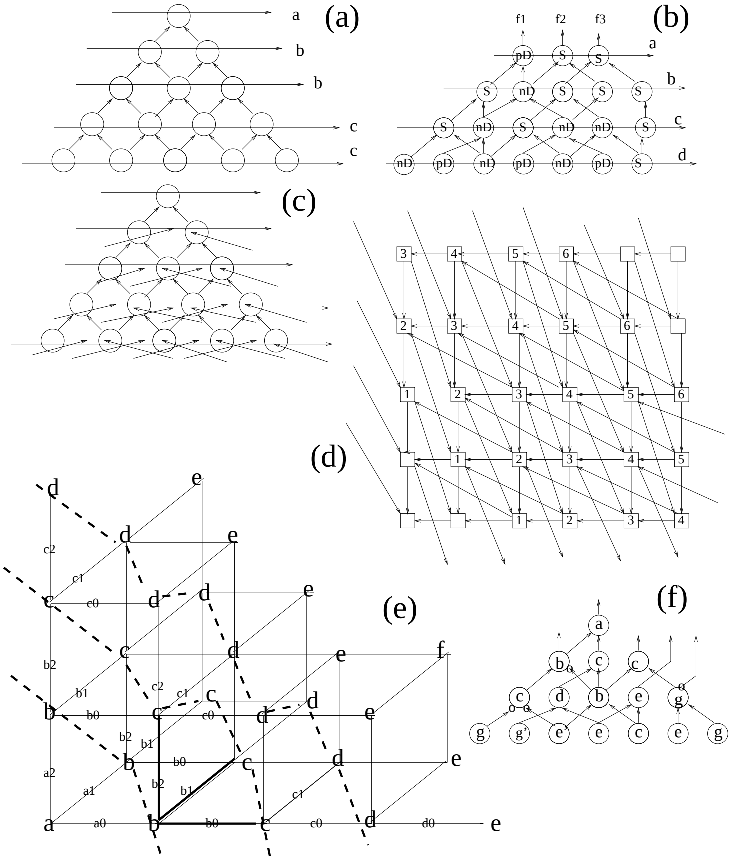

Figure 3: *Other regular lattices.*

planar lattices.

3. LAYOUT GEOMETRIES

In **case of 4 neighbors**, 2x2 cells, Fig. 2a,c,d and Fig. 3a, the lattice is planar and it is based on a rectangular grid. Each cell has two inputs (from N and E) and two outputs (to S and W) [1,4,18,21]. Our structure generalizes the known switch realizations of symmetric binary functions [1,4], based on Shannon expansion, but it allows also for Positive and Negative Davio expansions [19], negated variables and constants as control variables of the nodes, nodes controlled not by variables but by functions, and inverted edges between nodes. Lattice diagram counterparts of Kronecker [7], Pseudo-Kronecker [19] and Free Diagrams can be realized. It can be shown [8] that every function that is not symmetric can be symmetrized by repeating variables in the lattice layers, and the selection of the next variable is done using the Repeated Variable Maps from [12]. With respect to possible technological realization, the condition of only a single control variable in a level [4,14] is no longer required, and all three types of buses (vertical, horizontal and diagonal) are used to lead any variable to the circuit's levels. Similarly we do not always require the existence of the diagonal buses, as well as the condition of having only constant values on the envelope of the circuit, or having the outputs only on the envelope. Now, arbitrary variables can occur on the envelope, and outputs can be taken from the middle by use of buses (this is an approach from commercial Fine Grain FPGAs [2]). Moreover, pairs or triplets of binary control variables can be used in nodes for arbitrary Linearly-Independent expansions [10]. Or equivalently, multivalued controls are used. In short, for a 4-neighbor lattice geometry, any canonical form of Reed-Muller logic and its Linearly Independent generalizations can be realized (recall that the Reed-Muller logic is a special case of Galois Field logic, the GF(2)). Also, any MV logic (for instance in GF(4), [6]) can also be realized in the 4-neighbor lattice, but this would often require many repetitions of variables. Continuous and fuzzy functions can also be realized: in one example we realized an arbitrary piecewise continuous function with a generalized Shannon expansion [18]. Ternary and quaternary lattices for binary, multiple-valued and continuous functions are discussed in more detail in [16].

In **case of 6 neighbors**, 3x3 cells, Fig. 2b,f and Fig. 3b,f,e, the rectangular grid is enhanced with one diagonal connection, thus every cell has three inputs (from N, NE and E) and three outputs (to W, SW and S). This allows a realization of the generalized ternary diagrams (for binary EXOR logic) introduced in [9], as well as realizations of arbitrary expansion-based Post logic or GF(3) logic functions [10]. Finally, any binary, or MV logic can be mapped as in the case of the 4-neighbor lattice, but now larger full trees are mappable to subsets of lattices.

In **case of 8 neighbors**, 4x4 cells, Fig. 2e, and Fig. 3c,d, the rectangular grid is enhanced with one more diagonal connection, so that every cell has four inputs (from N,NE,NW, and E), and four outputs (to S, SW, SE, and W). This allows a realization of the generalized quaternary diagrams (for GF(4)), as well as arbitrary expansion-based Post or GF(k), $k < 4$ functions. Again, any binary or MV logic can be mapped, and more efficiently so. In other geometry variant, the neighborhoods are: NWW, NW, NE, NEE for inputs, SWW,SW SE, and SEE for outputs.

Good results were found for new ternary diagrams [9,20] and Galois(4) expansions for multi-output incompletely specified functions. Lattice Kronecker Decision Diagrams (LKDDNEs) with negated edges are based on three orthogonal expansions (Shannon, Positive Davio, Negative Davio) [16]. Many other new families of functional decision diagrams were created, including: Pseudo Kronecker Lattice DDs, Free Kronecker Lattice KDDs, Boolean Ternary Lattice DDs, and others [13]. The Boolean Ternary Lattice DDs introduce nodes with three edges and requires AND, OR and EXOR gates for expansion circuit realizations [14,20]. Galois GF(3) and GF(4) diagrams require three and four neighbors on inputs, respectively. The families of lattice diagrams we introduced are counterparts and generalizations of several diagrams known from the literature (BDDs, FDDs, KFDDs). Due to this property, our diagrams can provide a more compact representation of functions than either of the standard decision diagrams, because they do not require any placement or routing. Placement and routing come as a side-effect of logic synthesis. Our methods are very efficient especially for strongly unspecified functions, the more unspecified the function, the better the results.

4. APPLICATION EXAMPLES

Figs. 1a, 2a present the standard lattice structure investigated in [1,4,18]. Arrows show the signal flow. Fig. 2b presents this structure expanded with NE/SW diagonal connections, and Fig. 2e presents this structure expanded with NE/SW and NW/SE diagonal connections. Fig. 2c illustrates that standard vertical and horizontal buses can be used instead of diagonal buses for small functions. Fig. 2d shows a symmetric function of 7 variables realized in a standard lattice structure with diagonal buses (no repeated variables). Fig. 2f shows a non-symmetric function realized in a standard lattice structure with control input run from diagonal, vertical and horizontal buses (buses not shown, observe the constants 0 and 1 inside the lattice, as well as the folded and repeated variables). Fig. 3a shows another view of a standard lattice for symmetricized non-symmetric function (variables b and c are repeated). Fig. 3b shows 3x3 lattice with S, pD and nD expansions for a 3-output function with generalized (but no standard) symmetries. Fig. 3f presents a lattice with 3x3 neighborhood and repeated and folded variables. All expansions are S and variable names are in nodes. Small circle on input to first from left node c in the third level corresponds to an inverted edge. Figs. 3c,d are two lattice geometries with 4x4 neighborhood. Fig. 3e presents a lattice with 3x3 neighborhood drawn to emphasize ternary symmetries in 3-dimensional space. For each node, its expansion variable is shown. Some edges show variables' values. Bold edges in the figure correspond to three values of variable b expanded for $a = 0$. Observe, that the same figure can also be viewed as a flat connection plan and diagonal NW/SE buses can be added (interrupted lines). Observe also, that in this figure the number of nodes in the layers is 1,3,6,9,... while in the regular lattice from Fig. 3f the numbers were 1,3,5,7... so that not all partial ternary symmetries could be realized there in one level. This is an example of a trade-off between regular geometries and the logic, that must be solved in selecting the type of lattice for a given type of function.

5. CONCLUSION

The main idea of the presented approach can be summarized as follows: starting from **all possible** neighbor geometries in two and three dimensional spaces, we create **all possible regular structures**. This is more powerful than in the previous structures [1,4] which considered limited planar geometries. Next we design **arbitrary expansions** for any of the structures. New expansions can be constructed based on the Linearly-Independent function theory, or any other canonical or non-canonical function expansions. There exists a very high number of various new expansions, in contrast to only Shannon expansion types used in lattice diagrams from [1,4,5]. This way, the same, in principle, layout-driven synthesis approaches are created for binary, multivalued, linearly-independent, Galois and continuous functions [8,16,18]. Thus, the presented approach generalizes and unifies many known expansions, decision diagrams, and regular layout geometries. These methods are of special interest to deep sub-micron technology and pass-transistor design for binary and MV gates.

References

[1] S.B. Akers, "A rectangular logic array," *IEEE TC*, Vol. C-21, pp. 848-857, Aug. 1972.
[2] Concurrent Logic Inc., "CLI 6000 Series Field Programmable Gate Arrays," *Prelimin. Inform.*, Dec. 1, 1991, Rev. 1.3.
[3] R.E. Bryant, "Graph-based algorithms for boolean function manipulation, *IEEE TC*, Vol. C-35, No. 8, pp. 667-691, 1986.
[4] M. Chrzanowska-Jeske, Z. Wang and Y. Xu, "A Regular Representation for Mapping to Fine-Grain, Locally-Connected FPGAs," *Proc. ISCAS'97*.
[5] M.A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Minimization of Lattice Diagrams for Deep Sub-Micron Technology," *subm. to ICCIMA'97*.
[6] K.M. Dill, K. Ganguly, R. Safranek, and M.A. Perkowski, "A New Linearly Independent, Galois-Field Reed-Muller Logic," *Proc. RM'97*.
[7] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski, "Efficient representation and manipulation of switching functions based on Ordered Kronecker Functional Decision Diagrams," *Proc. DAC*, pp. 415-419, 1994.
[8] M.A. Perkowski, and E. Pierzchala, "New Canonical Forms for Four-valued Logic", *Report, Dept. Electr. Engn.* PSU, 1993.
[9] M. Perkowski, M. Chrzanowska-Jeske, A. Sarabi, and I. Schaefer, "Multi-Level Logic Synthesis Based on Kronecker and Boolean Ternary Decision Diagrams for Incompletely Specified Functions," *VLSI Design*, Vol. 3, Nos. 3-4, pp. 301-313, 1995.
[10] 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.
[11] M. Perkowski, L. Jozwiak, and R. Drechsler, "A Canonical AND/EXOR Form that includes both the Generalized Reed-Muller Forms and Kronecker Reed-Muller Forms," *Proc. RM'97*.
[12] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J. Zhang, "Decomposition of Multi-Valued Relations," *Proc. ISMVL'97*, Nova Scotia, May 1997, pp. 13 - 18.
[13] M.A. Perkowski, L. Jozwiak, and R. Drechsler, "Two hierarchies of Generalized Kronecker Trees, Forms, Decision Diagrams, and Regular Layouts," *Proc. RM'97*.
[14] M.A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Lattice Diagrams Using Reed-Muller Logic," *Proc. RM'97*.
[15] M.A. Perkowski, "Bidirectional Decision Diagrams for Synthesis with Complex Pass Transistor Gates," *Booklet Intern. ULSI Workshop*, Antigonish, Nova Scotia, May 27, 1997, pp. 37 - 40.
[16] M.A. Perkowski, E. Pierzchala, and R. Drechsler, "Ternary and Quaternary Lattice Diagrams for Linearly-Independent Logic, Multiple-Valued Logic, and Analog Synthesis," *Proc. ICICS'97*, Singapur, Sept. 9 - 12, 1997.
[17] E. Pierzchala, and M.A. Perkowski, "High Speed Field Programmable Analog Array Architecture Design," *Proc. FPGA'94*, Berkeley, California, February 1994.
[18] E. Pierzchala, M.A. Perkowski, and S. Grygiel, "A Field Programmable Analog Array for Continuous, Fuzzy, and Multi-Valued Logic Applications," *Proc. 24-th ISMVL*, pp. 148-155, Boston, May 25-27, 1994.
[19] T. Sasao (editor), "Logic Synthesis and Optimisation," *Kluwer Academic Publishers,* 1993.
[20] T. Sasao, "Ternary Decision Diagrams: Survey," *Proc. ISMVL'97*, pp. 241 - 250.
[21] 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,* San Diego, June 1994, pp. 321 - 326.