International Conference on
Information, Communications and Signal Processing
ICICS '97
Singapore, 9-12 September 1997

TERNARY AND QUATERNARY LATTICE DIAGRAMS FOR LINEARLY-INDEPENDENT LOGIC, MULTIPLE-VALUED LOGIC, AND ANALOG SYNTHESIS

Marek A. Perkowski, Edmund Pierzchala, and Rolf Drechsler +,
Dept. Electr. Engn., Portland State University, Portland, OR 97207
+ Inst. Comp. Sci., Albert-Ludwigs-University, 79110 Freiburg in Breisgau, Germany

Abstract

Ternary and Quaternary Lattice Diagrams are introduced that can find applications to submicron design, and designing new fine-grain digital, analog and mixed FPGAs. They expand the ideas of Lattice diagrams [6, 11] and Linearly Independent (LI) Logic [5, 7, 8, 9, 10, 12, 17, 18]. In a regular layout, every cell is connected to 4, 6 or 8 neighbors and to a number of vertical, horizontal and diagonal buses. Various lattices and algorithms for their creation are presented.

1. INTRODUCTION.

The goal of Lattice Diagrams is layout-driven logic synthesis in cellular structures with mostly local connections. The concept of a lattice diagram [11] involves three components: **(1) expansion** of a function (the function corresponds to the initial node in the lattice), which creates several successor nodes of this node, (2) **joining** of several (not necessarily tautologic) nodes of a tree level to a single node, which is in a sense a reverse operation to the expansion, (3) a **regular geometry** to which the nodes are mapped, this geometry guides which nodes of the level are to be joined. The procedure of expanding and joining nodes in levels is iterated for (repeated) variables until all node functions become variables or constants. Cell with n inputs and m outputs is said to have n x m connectivity pattern. Below, we will present some ternary lattices (with 3 inputs and 3 outputs from a node) and quaternary lattices (with 4x4 connectivity pattern) for Shannon, Davio, nonsingular, fuzzy and analog expansions.

2. TYPES OF EXPANSION NODES.

Fig. 1 presents different expansion nodes for various kinds of expansions for binary, multi-valued, and fuzzy functions. Fig. 1a shows two views of a cell for Shannon (S) expansion: a multiplexer, and a general notation of a 2x2 cell in a Lattice that may be realized by this mux (the notation of inputs and outputs is preserved in next examples). When input a is inverted, the so-called Reversed Shannon (S') expansion is executed, which means that the role of inputs b and c is reversed. Fig. 1b shows the positive Davio expansion node (pD), and Fig. 1c the negative Davio node (nD). Such nodes are used in Positive-Polarity, Fixed-Polarity, Kronecker and Pseudo-Kronecker Lattices and their generalizations [5, 6, 7, 8, 9, 10]. Fig. 1e presents Shannon node for ternary logic, Fig. 1f Shannon node for quaternary logic, and Fig. 1g realization of the quaternary Shannon node from (f) in binary logic. Two binary signals routed together simulate a 4-valued signal. It can be observed [13], that a fundamental condition for existence of joining operations and thus, ability of creating lattice diagrams is that in the underlying algebraic structure any two literals are disjoint (in binary, this property reduces to $a \cdot \bar{a} = 0$). This leads to binary and multiple-valued (MV) Max-type lattices (we denote max-type operations by + and min-type operations by $\cdot$). The principle of operation of binary max-type lattices is that any path in a diagram that includes x and $\bar{x}$ cancells. EXOR function is: $a \oplus b = a \cdot \bar{b} + \bar{a} \cdot b$. Thus, $a \oplus a = a\bar{a} + \bar{a}a = 0$. This leads to Linearly-Independent type (LI) lattices [11]. The principle of operation of LI-type lattices is that any two identical paths to the root in the diagram cancel one another ($x \oplus x = 0$).

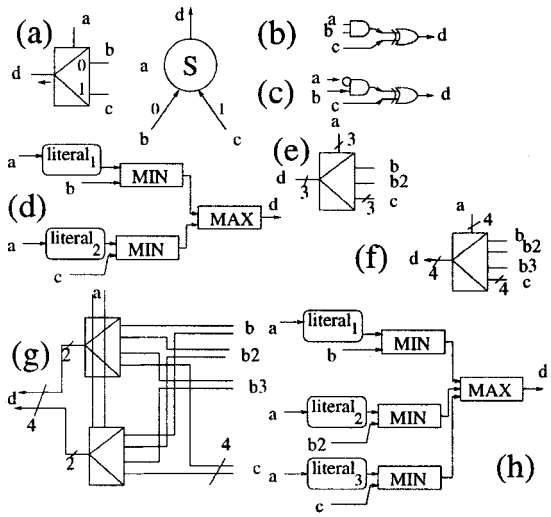

Figure 1: *Comparison of Expansion nodes for lattices.*

2. BINARY LI-TYPE LATTICES.

Fig. 2 presents a comparison of sizes of a standard binary Shannon lattice and two new types of lattices for EXOR/XNOR function. Fig. 2a presents a solution that would be obtained using the standard Shannon lattice from [1, 2, 6]. The order of control variables is a, b, c, d. Because the function is symmetric, variables are not repeated. Observe that arrows with 0 (for negated control variable)

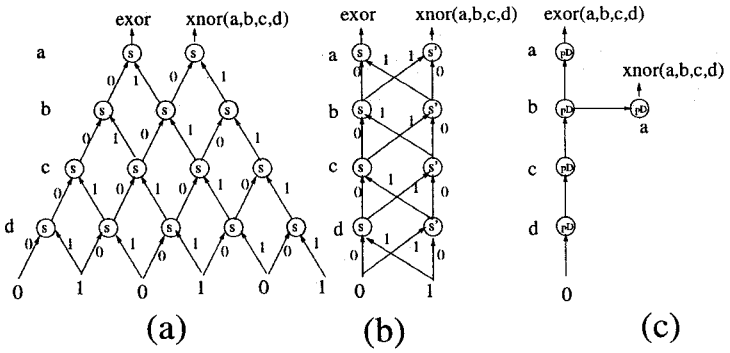

Figure 2: *Comparison of three types of Lattices for two-output EXOR/XNOR function.*

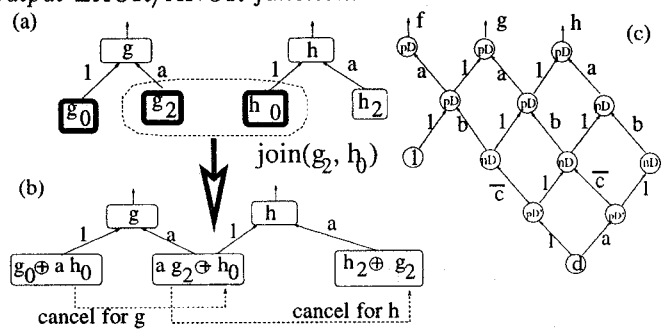

Figure 3: *Creation of a Positive Davio level in a Lattice: (a) two expanded nodes before joining, (b) layer of lattice after joining operation on nodes g_2 and h_0, (c) Fixed-Polarity RM Lattice for functions f, g, h.*

are always in left. The shape is a trapezoid and the size is 14 nodes. Connectivity pattern is 2x2. The Akers Array [1] would have (5 * 5) * 2 nodes (it realizes each of two functions separately, and uses a 5 * 5 fixed square for a 4 variable function). Fig. 2b presents our solution with 3x3 connectivity pattern array of multiplexers. It is linear in shape and has 2 * 4 = 8 nodes. In addition to Shannon (S), the Shannon expansions with negated control variables (S') are now used. Observe that arrows from the left have both 0 and 1 values. Fig. 2c presents Positive Polarity Reed-Muller Lattice 2x2 connectivity pattern array of positive Davio (pD) nodes. It is nearly linear in shape and has 5 nodes. This figure clearly demonstrates an advantage of having higher connection patterns and more general expansion types. Predictability and equality of delays should be appreciated in all lattices.

The functions in Fig. 2 where symmetric, but what about lattice realization of non-symmetric functions? **Firstly**, we defined the **Polarized Pseudo-Kronecker** symmetries [3] which are much more general than known symmetries of functions, so using them, more functions can be put to lattices without repeating variables. **Secondly,** functions that do not have the Polarized Pseudo-Kronecker symmetries can be still realized in lattices with repeated variables [14, 15]. This requires, however, a joining operation for nodes. Figs. 3a,b present the principle of joining operation for EXOR-based, and in particular LI logic. Although it is shown here only for pD nodes and an ordered lattice, the same principle is used for more complex expansions and lattice diagrams of the LI type. In Fig. 3a illustrates the local situation in a level of lattice after using pD expansion with respect to variable a to nodes g and h. In this figure, $g_0 = g(a=0)$, $g_1 = g(a=1), g_2 = g_0 \oplus g_1$ are the negative and positive cofactors and Booleand difference, respectively. Fig. 3b presents the result of joining successor nodes g_2 and h_0. The joining rule is: $g_2\ JOIN\ h_0 = ag_2 \oplus h_0$, which means that nodes representing functions $g_2 = g_0 \oplus g_1$ and h_0 are joined to a new node representing function $ag_2 \oplus h_0$. The **correction terms** ah_0 and ag_2 are propagated to left and right, respectively. It can be easily checked, that because of term cancelling (based on principle $x \oplus x = 0$), in the lattice level of variable a from Fig. 3b the pD expansions of g and h are still satisfied: $g = g_0 \oplus ag_2$, and $h = h_0 \oplus ah_2$.

Fig. 3c presents Fixed-Polarity Reed-Muller Lattice Diagram (expansions pD and nD) for functions: $f = a \oplus ab\bar{c}d$, $g = 1 \oplus b\bar{c}d \oplus a\bar{c}d \oplus abd \oplus ab\bar{c}d$, $h = \bar{c}d \oplus bd \oplus ab\bar{c}d \oplus a\bar{c}d \oplus abd \oplus ad$. Variable a is used first in pD level on top of the lattice from Fig. 3c. As shown in Fig. 3c, expansions for variables a and b in first two levels are pD, and the expansion for variable c in the third level is c. Variable a is repeated once more in the bottom level of the lattice. The expansion in this level is pD', which means, a reversed pD, that is a pD expansion with reversed role of data inputs. Observe, that although the function is not symmetric in a standard way, it is lattice-realizable without variable repetitions because it has polarized Pseudo-Kronecker symmetries. In some types of expansions the propagation of correction terms is only to right, or only to left. In some other expansions, especially the non-canonical ones, more powerful corrections types are created, and the algorithm selects the correction rule evaluated as the one leading to the simplest next level of the lattice. Selecting the order of (repeated) variables and the expansion type in each node are the most important and difficult problems to be solved.

3. TERNARY AND QUATERNARY LATTICES.

It is easy to generalize the binary Shannon expansions used in Fig. 2a,b to 3-valued and 4-valued Shannon expansions. The new lattices would require 3 inputs and 3 outputs from a node, and 4 inputs and 4 outputs from a node, respectively [11]. Next 3- and 4- valued counterparts of S' can be created, and respective ternary and quaternary lattices can be formed by expanding and joining formulas. This way, Post-type and Galois-type lattices are created in an uniform way, with only difference of using various expansion and joining rules. However, the two kinds of principles, of creating the expansion and of the joining rules, remain the same: disjoint literals for max-type lattices, and $a + (-a) = 0$ term cancelling for LI lattices (which generalizes the rule $a \oplus a = 0$ of Galois Field (2) from section 2). The lattices have advantages especially for (nearly) symmetric functions and strongly unspecified functions that can be completed to symmetric functions.

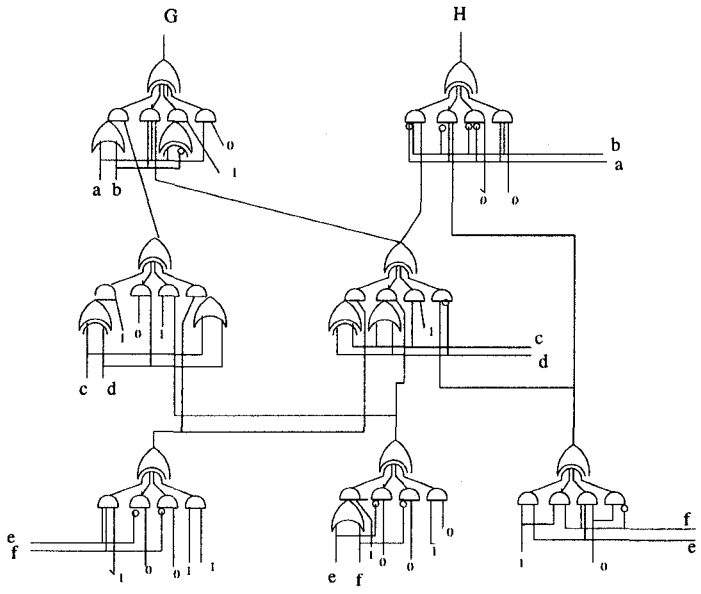

Figure 4: *A LI Pseudo-Kronecker Decision Lattice Diagram for variable blocks {a,b},{c,d},{e,f} to function from [11].*

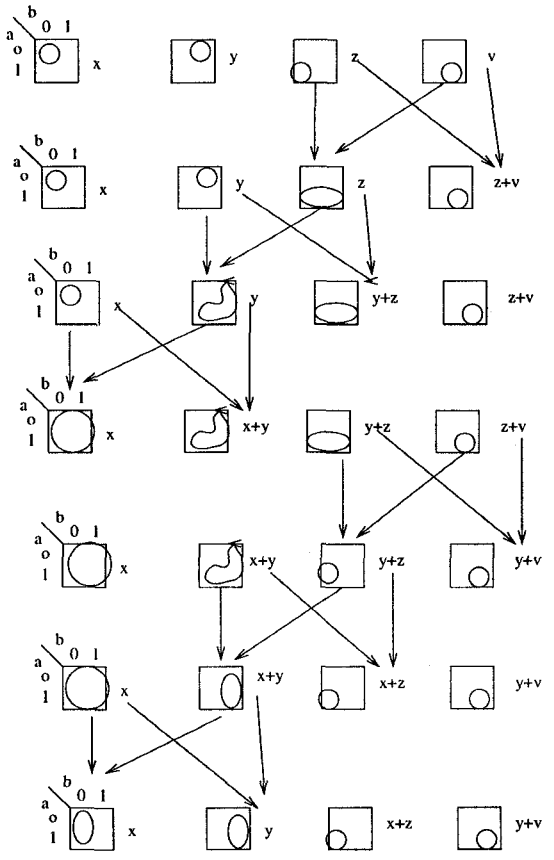

Figure 5: *First part of the Butterfly Diagram to find the best nonsingular expansion by creating expansions for all LI functions of a,b.*

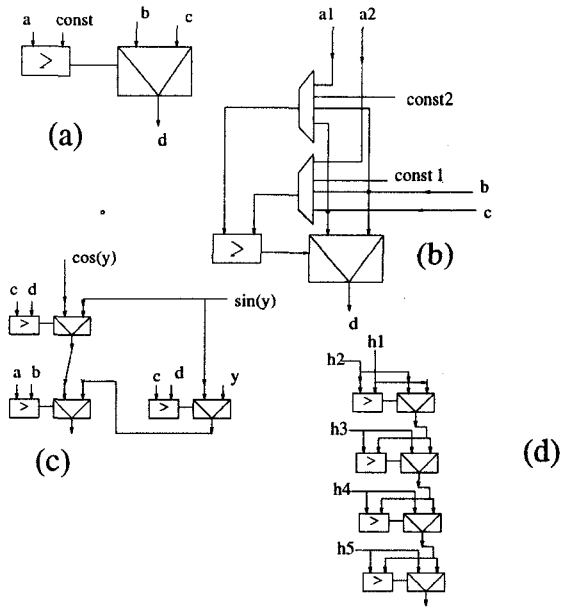

Figure 6: *Realization of analog functions.*

By a **regular layout** we understand a layout of indentical cells that connect by abutting. By a **complete layout structure** we understand connection pattern between cells, that allows to realize every symmetric function without repeating variables. It can be proved that in a 2x2 lattice **every** binary symmetric function can be realized without variable repetitions, and with connections between cells having the same length. Thus, **lattice layout for binary logic is regular and complete.** In contrast to binary functions, symmetric ternary functions cannot be realized in regular 2-dimensional 3x3 lattices. Although we created 3x3 lattices that can realize every symmetric ternary function without variable repetitions [11], it is not possible to find regular layouts for realizing them. Thus the cells distances in subsequent levels grow. Hopefully, it is not a practical problem for small functions realized in MV logic, but the beautiful simplicity of binary realizations does not longer exist. Thus, if mapped to a 2-dimensional space, the ternary lattices are either regular and not complete, or complete but not regular. It is still possible to obtain regular and complete 3x3 lattices assuming layout of cells in a three-dimensional space. But it is not possible to create regular layout for 4x4 lattices, because our Universe is 3-dimensional. Although these considerations are theoretically interesting, the two types of ternary and quaternary lattices that we developed; the regular and incomplete, and the irregular and complete, are very useful in practice. The difficulties would be only for very large functions, but in any case it is our assumption that the lattice approach is for not too large functions, because it is applied to functions that result from Curtis-like nondisjoint, hierarchical functional decomposition of MV relations [16].

4. QUATERNARY LI LATTICES.

As shown in Fig.1g, pairs of binary variables correspond to

4-valued variables. Although here we discuss LI lattices for only two variables in each **variable block**, all concepts and algorithms can be expanded to variable blocks of arbitrary size. Fig. 4 shows an example of a circuit obtained by substituting nodes of a quaternary LI lattice diagram with their circuits (this particular circuit is even 3x3 realizable, a dummy node is used in level 2).

The LI Lattice diagrams for pairs of variables are created similarly to lattices for single variables in section 2. Nodes are now for pairs of variables, and nonsingular expansions of LI logic [7, 8, 9] are used. Every node has at most 4 inputs. Instead selecting among only three expansions, S, pD and nD, the choice in every level of nodes is among all 840 nonsingular expansions in exact algorithm (this is the maximum number of nonsingular expansions for a pair of variables), or some subset of them in approximate algorithms.

The same type of expansion is selected in Kronecker type lattices, or various expansions are selected in nodes of Pseudo-Kronecker type lattices. The joinings are based on the same principles as in sect. 2. The lattices for all single outputs of a multi-output function are created together, level-by-level from their root nodes (outputs). In every level, the possible expansions are evaluated based on the complexity of the next level (look-ahead strategy). The best expansion found by the Polarity Selecting Algorithm for a level is next applied to all nodes (Kronecker types) from the level of the multi-output diagram. In Pseudo type of lattices, the expansion decision for each node is done separately. The algorithm from this section is used for small functions, and approximate algorithms from [6, 13] for larger functions.

One of interesting concepts of Reed-Muller logic are the **butterfly diagrams** that allow to create all fixed polarity expansions by transforming from polarity to polarity, and doing this just by incremental exoring of some terms from the forms. This way, all forms of certain type are systematically created without even creating their **expansion matrices** M [7, 9] and without calculating their inverse matrices M^{-1}. The concept of Gray-code ordering of all Generalized Reed-Muller polarities was applied to find the exact minimum GRM form [22]. We introduce here similar ideas for the LI forms.

Property 1. The following rule **BR** holds

$f_1(x_1,x_2)SF_2(x_3,...,x_n) \oplus f_3(x_1,x_2)SF_4(x_3,...,x_n) =$
$[f_1(x_1,x_2) \oplus f_3(x_1,x_2)]SF_2(x_3,...,x_n)$
$\oplus f_3(x_1,x_2)[SF_2(x_3,...,x_n) \oplus SF_4(x_3,...,x_n)]$

where $f_1(x_1,x_2)$ and $f_3(x_1,x_2)$ are arbitrary LI functions, and $SF_2(x_3,...,x_n)$ and $SF_4(x_3,...,x_n)$ are the corresponding to them data input (DI) functions. [1].

Property 2. Any nonsingular expansion can be obtained by a repeated application of Rule BR to pairs of functions
$[f_1(x_1,x_2), SF_2(x_3,..,x_n)],[f_3(x_1,x_2), SF_4(x_3,..,x_n)]$.

This way, rule BR describes simultaneous EXOR-ing of columns in matrix M and corresponding columns in M^{-1}.

[1] It is easy to verify that this rule is true by simple Boolean manipulations comparing its left and right sides

But how to select the pairs of functions?

Property 3. In matrix M, as well as in matrix M^{-1}, any column can be replaced by a linear combination of itself with other columns. Thus, any *polarity expansion* can be obtained by a repeated application of the basic rule BR to certain selected columns.

Even if in general there is no recursive way to define the universal Butterfly-like diagram for *arbitrary* LI matrix, a specific diagram can be once created for a set of variables with *certain number* of elements and for any set of expansion polarities. This diagram can be stored in memory, and next used for evaluations for each particular function of the respective number of variables. We will call this a "precomputed" Butterfly diagram.

Our exhaustive algorithm goes through all polarities. The set of all polarities is created as levels (rows) in a butterfly-like diagram from Fig. 5 (for the lack of space, only first few levels are shown). Small K-maps correspond to some LI functions $f(x_1,x_2) = f(a,b)$ and x,y,z,v correspond to the original cofactors $SF_{00}(x_3,...,x_n)$, $SF_{01}(x_3,...,x_n)$, $SF_{10}(x_3,...,x_n)$, $SF_{11}(x_3,...,x_n)$, respectively (top row of the diagram). EXOR-ing on LI functions according to BR rule is shown here graphically on K-maps. EXOR-ing of the respective DI functions is shown on formulas that stand on the right sides of the respective K-maps. However, the simplification rule $X \oplus X = 0$ is used in these formulas, in order to express them all in terms of EXORs on some subset of the initial cofactors. It can be observed, that by applying the law $(x \oplus y) \oplus (y \oplus z) = (x \oplus z)$, the DI functions $SF_i(x_3,...,x_n)$ are repeating in the levels of the diagram and do not have to be computed repeatedly in the diagram. Thus, the diagram for all nonsingular expansions for pairs of variables can be created only once, and next the values of EXOR-sums of subsets of functions $SF_i(x_3,...,x_n)$ can be just inserted for any particular initial cofactors. Thus the number of EXOR operations on subsets of cofactors x,y,z,v is essentially decreased (for efficiency of EXORing, the cofactors can be represented as BDDs or in any other way). Because the algorithm goes through all polarities, it can be used to find the best polarity for a lattice level, and thus it can be used as part of an algorithm to create minimum lattices, for given partitioning of variables to blocks, and given ordering of blocks.

5. ITERATIVE CIRCUITS, CIRCUITS WITH MEMORY, FUZZY AND ANALOG DESIGN.

Because in standard fuzzy logic $a \cdot \bar{a} \neq 0$, and $a \oplus a = a\bar{a} + \bar{a}a \neq 0$, both the Max-type and LI methods from [11] would not work. However, let us observe that one can define a **negation-less fuzzy logic**, which we call a **Disjoint Fuzzy Logic (DFL)**, in which all fuzzy logic axioms besides those related to negation are satisfied, and negation is simulated by using special type of literals. In DFL logic, any two literals $literal_i$, $literal_j$ can have arbitrary shapes, but must be disjoint; for any value of $x \in [0,1]$ $literal_i(x) \cdot literal_j(x) = 0$. Fig. 1d presents the binary expansion node for binary DFL, and Fig. 1h

the ternary expansion node for ternary DFL. The literals $literal_1$, $literal_2$, $literal_3$ are all mutually disjoint. DFL expansions are realized in ternary fuzzy lattices, similar to MV ternary lattices. Because of disjoint literals, joining operation can be always performed [11]. Observe, that k-valued Post logic is a special case of DFL with k literals, (so the ternary Shannon expansion is a special case of the ternary DFL expansion). Non-Lattice memory-less arrays for fuzzy logic are also presented in [6, 18].

Hierarchical design of **iterative** one- and two-dimensional structures is possible, which are cellular connections of logic blocks, each block realized as a multi-output lattice. This can be done also for discrete circuits with memory. Analog counterparts use sample-hold analog memories, which play the same role as flip-flops in discrete technologies. Lattices allow thus the realization of cellular memory-less functions, finite state machines, and infinite state machines; realized in analog, binary, or multivalued logic. For instance, the digital and analog: filters, pipelined image processors, or systolic processors. An elliptic ladder filter was mapped to this structure [17]. Fig. 6a shows a basic circuit with analog comparator and analog multiplexer, and Fig. 6b a full cell with SRAM-controlled muxes to switch the inputs. Two simple lattices for analog functions are shown in Fig. 6c,d. Fig. 6c presents a lattice realization of the piecewise continuous function **if** $((c > d)$ **and** $(a > b))$ **then y else if** $((a \leq b$ **and** $(c \leq d))$ **then cos(y) else sin(y)**. Fig. 6d shows a lattice realization of analog function $\max(h_1, h_2, h_3, h_4, h_5)$. Similar realizations can be created for rank and median filters, cellular neural nets, equation solvers, and (analog and digital) image processing circuits [19].

6. CONCLUSION.

We showed examples of ternary and quaternary lattices for binary, multi-valued, DFL and analog logic, and how to create them. In particular, we showed how to create quaternary lattice for binary LI logic. Such diagrams are the most general binary lattice diagrams introduced so far. The method presented for LI logic can be applied to both completely specified and incompletely specified functions; single-, and multi-output. Both Kronecker-like and Pseudo-Kronecker-like generalizations can be created. Further generalization to LI Free lattices is also possible along the lines from [5]. Generalizations to Mixed, Ordered, Free, Lattice and other LI representations [10] are also possible. We developed also a very similar approach to map multiple-output *Boolean relations* to trees, diagrams, and lattices.

In general, the presented methods allow for layout-driven synthesis approaches to binary, multivalued, linearly-independent, Galois, fuzzy, analog and mixed functions, unifying many known expansions, decision diagrams, regular layout geometries and FPGA/FPAA structures. These methods are of special interest to various new technologies based on regularity and locality of connections: deep submicron technology, binary and MV pass-transistor designs, quantum logic devices, OTA circuits, and new fine grain digital and analog FPGAs.

REFERENCES

[1] S.B. Akers, "A rectangular logic array," *IEEE TC.*, Vol. C-21, pp. 848-857, Aug. 1972.

[2] M. Chrzanowska-Jeske, Z. Wang and Y. Xu, "A Regular Representation for Mapping to Fine-Grain, Locally-Connected FPGAs," *Proc. ISCAS'97.*

[3] B.T. Drucker, C.M. Files, M.A. Perkowski, and M. Chrzanowska-Jeske, "Polarized Pseudo-Kronecker Symmetry with an Application to the Synthesis of Lattice Decision Diagrams," *subm. Proc. ICCIMA'98.*

[4] B. J. Falkowski, S. Rahardja, "Family of fast transforms for GF(2) orthogonal logic," *Proc. RM'95*, pp. 273-280.

[5] P. Ho, M. A. Perkowski, "Free Kronecker Decision Diagrams and their Application to ATMEL 6000 FPGA Mapping," *Proc. Euro-DAC'94/VHDL'94*, pp. 8 - 13, Sept. 19-23, 1994, Grenoble France.

[6] M.A. Perkowski, and E. Pierzchala, "New Canonical Forms for Four-valued Logic", *Report, Dept. Electr. Engn.* PSU, 1993.

[7] M. Perkowski, "A Fundamental Theorem for Exor Circuits," *Proc. RM'93*, pp. 52-60.

[8] M. Perkowski, A.Sarabi, F. Beyl, "XOR Canonical Forms of Switching Functions," *Proc. RM'93*, pp. 27-32.

[9] M. Perkowski, A.Sarabi, F. Beyl, "Fundamental Theorems and Families of Forms for Binary and Multiple-Valued Linearly Independent Logic," *Proc. RM'95*, pp. 288-299.

[10] M.A. Perkowski, L. Jozwiak, R. Drechsler, "Two Hierarchies of Generalized Kronecker Trees, Forms Decision Diagrams, and Regular Layouts," *Proc. RM'97 Symposium*, Oxford University, U.K., September 1997.

[11] M.A. Perkowski, E. Pierzchala, and R. Drechsler, "Layout-Driven Synthesis for Submicron Technology: Mapping Expansions to Regular Lattices," *Proc. ISIC-97*, Singapur, Sept. 10-12, 1997.

[12] M.A. Perkowski, L. Jozwiak, R. Drechsler, and B. Falkowski, "Ordered and Shared, Linearly-Independent, Variable-Pair Decision Diagrams," *Proc. ICICS'97*, Singapur, Sept. 10-12, 1997.

[13] M.A. Perkowski, E. Pierzchala, L. Jozwiak, R. Drechsler, and B. Falkowski, "Approximate Algorithms for Selection of Good Non-Singular Expansions for an Incompletely Specified Multi-Output Boolean Function," *PSU Report*, 1997.

[14] M.A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Lattice Diagrams Using Reed-Muller Logic," *RM'97.*

[15] M.A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Minimization of Lattice Diagrams for Deep Sub-Micron Technology", *subm. ICCIMA*, 1997.

[16] 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.

[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] E. Pierzchala, and M.A. Perkowski, "A High-Frequency Field-programmable Analog Array (FPAA). Part 1. Design, Part 2. Applications," *Analog Integrated Circuits and Signal Processing*, Kluwer Academic Publishers, to be published.

[20] A. Sarabi, N. Song, M. Chrzanowska-Jeske, 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.

[21] I. Schaefer, M. Perkowski, "Synthesis of Multi-Level Multiplexer Circuits for Incompletely Specified Multi-Output Boolean Functions with Mapping Multiplexer Based FPGAs," *IEEE Tr. CAD,*, Vol. 12, No. 11, Nov. 1993. pp. 1655-1664.

[22] X. Zeng, M. Perkowski, K. Dill, A. Sarabi, "Approximate Minimization of Generalized Reed-Muller Forms," *Proc. RM'95*, pp. 221-230.