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

Marek A. Perkowski, Edmund Pierzchala, and Rolf Drechsler +,

Dept. Electr. Engn., Portland State University, Portland, USA

+ Inst. Comp. Sci., Albert-Ludwigs-University,
Freiburg in Breisgau, Germany

PLAN OF TALK

- Introduction - layout-driven synthesis.
- Expansions and expansion nodes.
- Max-type versus LI-type lattices.
- Use of Shannon expansions to create a lattice.
- SOP expansions.
- Regular layout.

MAIN IDEAS

- **Lattice Structure** - new concept in VLSI layout.
- Applications in submicron design, quantum devices, and designing new fine-grain FPGAs.
- 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.
- This is a new approach to **layout-driven logic synthesis** of combinational functions.

BINARY DECION DIAGRAMS FOR SYMMETRIC FUNCTIONS

- **Expansions** of functions, are 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 $f_a = f\,|_{a=1}$, and $f_{\bar{a}} = f\,|_{a=0}$ 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.
- 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.

WHAT IF FUNCTION IS NOT SYMMETRIC?

- We show hot to extend this approach to **arbitrary** binary functions, not necessarily symmetrical.
- **Other expansions** of functions can also be used.
- The expansion nodes are mapped to neighborhood structures which are more powerful than those investigated theoretically in the past (Akers, Chrzanowska-Jeske).
- The proposed here structures are similar to those from commercial Fine Grain FPGAs (Atmel - Concurrent Logic, Xilinx - Algotronix, Motorola - Pilkington).

THREE COMPONENTS OF LATTICE DIAGRAMS

- The concept of a **lattice diagram** involves three components:

(A) **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.

(B) **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.

USE OF MULTI-VALUED LOGIC

- 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.
- This makes the lattice **"fat"** and encodes the multi-valued signal to binary.

SPECIAL CASES OF LATTICE DIAGRAMS

- Some **special cases** of Lattice Diagrams are **theoretical models of cellular logic:**
  - Akers Arrays.
  - Fat Trees,
  - Generalized PLAs,
  - Maitra cascades,
- **Another** special cases of Lattice Diagrams are **industrial** structures from several patents of **fine grain FPGAs** (Motorola, Atmel, Plessey, Pilkington).

OUR GENERALIZATIONS

- In addition to structure, we show constructive and efficient methods of designing discrete functions in these structures.
- The methods were also extended to continuous functions.
- We showed on many examples 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.**

EXPANSION AND JOINING OPERATIONS

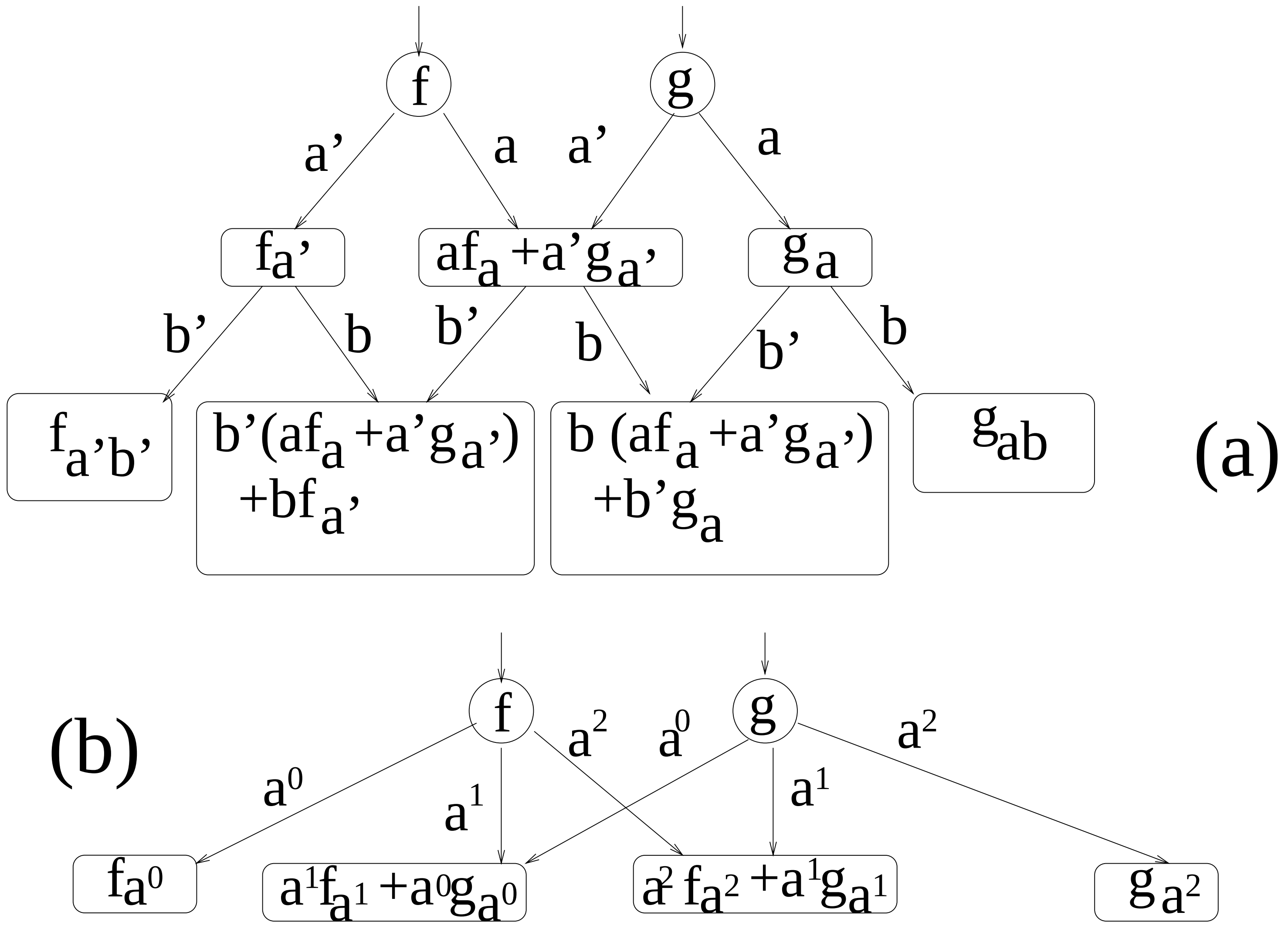

f
g
a'
a
b'
b
$f_{a'}$
$af_a + a'g_{a'}$
g_a
$f_{a'b'}$
$b'(af_a + a'g_{a'}) + bf_{a'}$
$b(af_a + a'g_{a'}) + b'g_a$
g_{ab}
(a)
(b)
a^0
a^1
a^2
f_{a^0}
$a^1 f_{a^1} + a^0 g_{a^0}$
$a^2 f_{a^2} + a^1 g_{a^1}$
g_{a^2}

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

TYPES OF EXPANSIONS

- Two types of expansions:
  - **maximum-type**,
  - **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) and Sum-of-Products (SOP) (Perkowski ULSI'97) expansions and their multiple-valued logic generalizations, such as in Post logic.
- Below we will present only the maximum-type expansions: Shannon, SOP, and Post (MV Shannon).

STANDARD COFACTORS VERSUS V-COFACTORS

- 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)].
- 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.
- Our cofactor, **vacuous cofactor**, denoted by **v-cofactor**, though, f_x is still a function of all variables including x, but as a result of cofactoring the variable x becomes vacuous.

STANDARD COFACTORS VERSUS V-COFACTORS

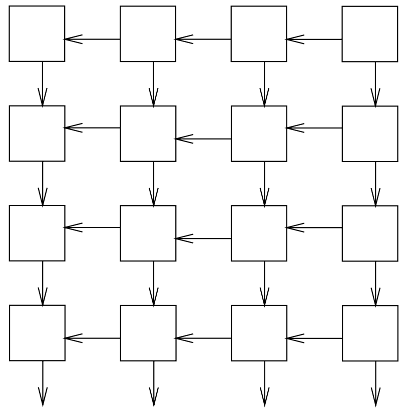
- Standard cofactors are in general not disjoint.
- For any two disjoint products a_1 and a_2, the v-cofactors f_{a_1} and g_{a_2} **are disjoint.**
- Therefore functions f_{a_1} and g_{a_2} are in an **incomplete tautology** relation, and functions f and g are not changed when f_{a_1} and g_{a_2} are joined (OR-ed) to create a new function:

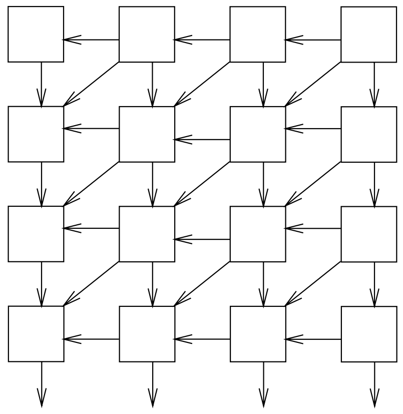
  $a_1 f_{a_1} + \overline{a_2} g_{a_2}$, as in Fig. 1a (where: $a_1 = a_2 = a$, and $\bar{a}$ is denoted as a').
- This way, the entire lattice is created **level-by-level**, Fig. 1a.
- Functions in lattice nodes become **more and more unspecified** when variables in levels are repeated.
- Ultimately nodes become constants.

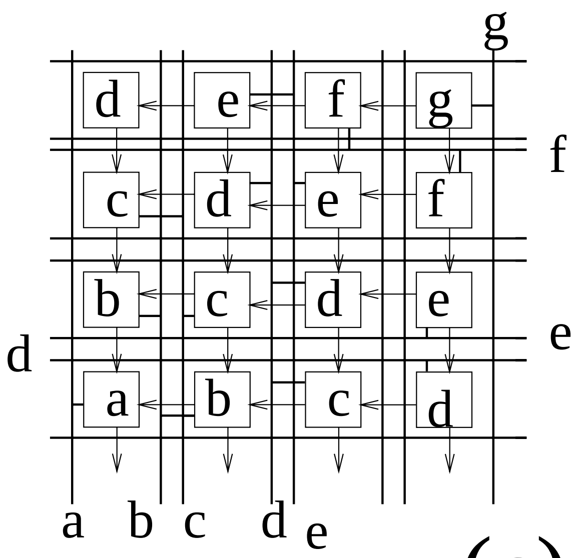
STANDARD COFACTORS VERSUS V-COFACTORS. II

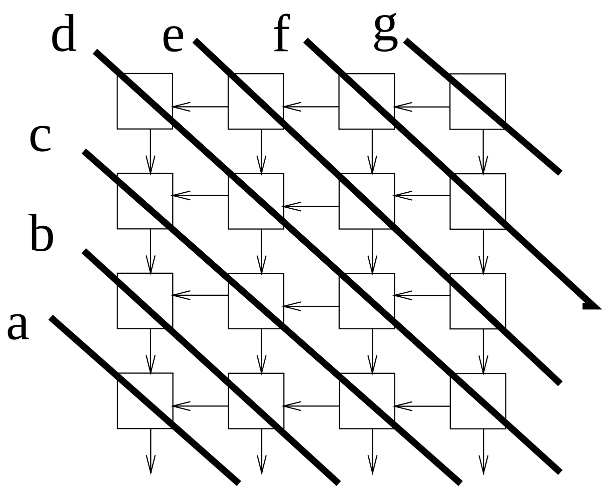
- Every variable cuts a Kmap into two disjoint parts.
- Thus, arbitrary two functions f and g can always be expanded together to a Shannon lattice, with OR-ing as a join operation, provided that:
  - the same variable x_i is used in the level,
  - and all expansions use negated literal $\bar{x}_i$ in the left, and positive literal x_i of the variable in the right.
- New functions in levels are created by rearranging the cofactors in joinings.
- This process can increase 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.

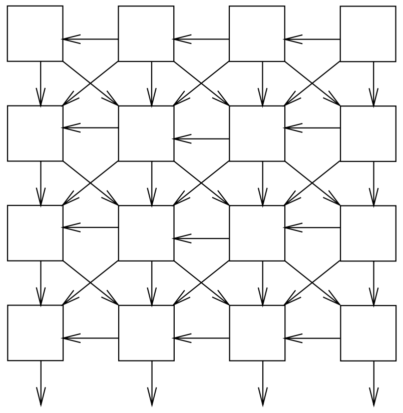
EXAMPLES OF REGULAR LATTICES. I

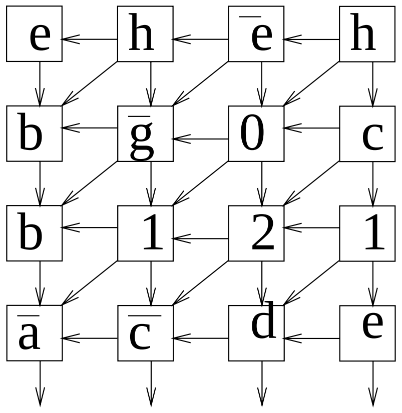

(a)


(b)

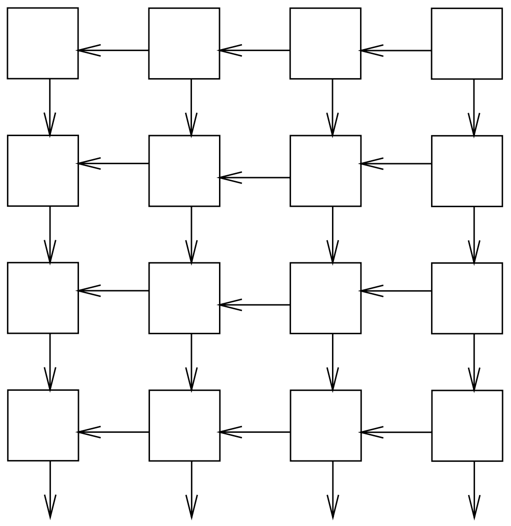

g
d e f g
f
c d e f
b c d e
d
e
a b c d
a b c d e
(c)

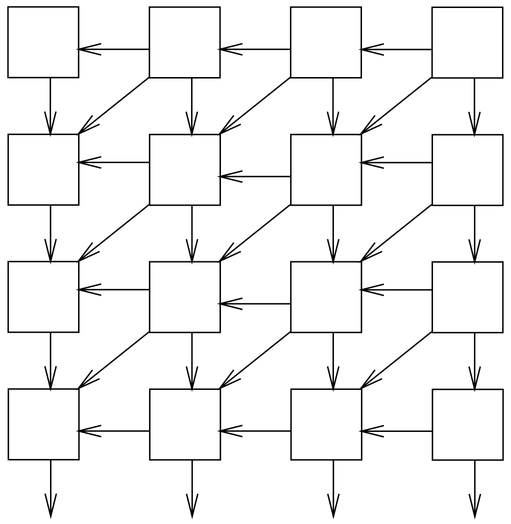

d e f g
c
b
a
(d)


(e)

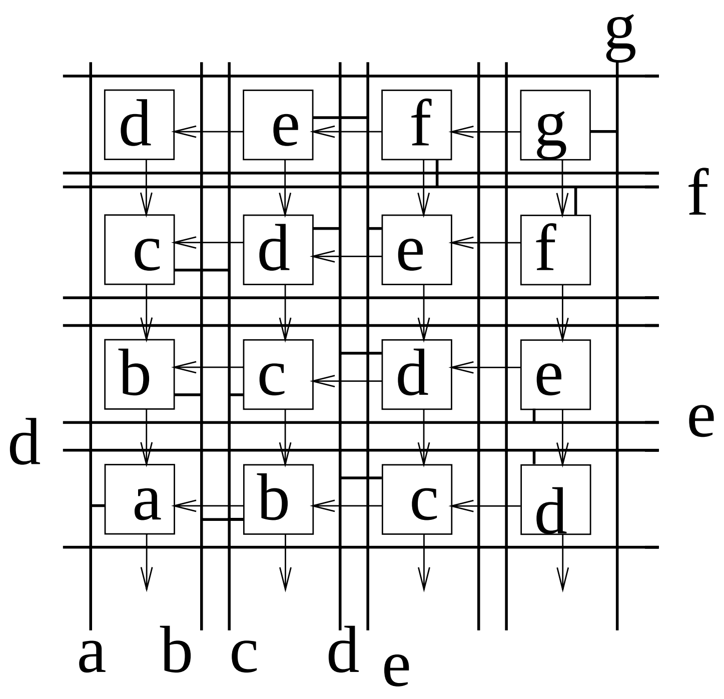

e h $\bar{e}$ h
b $\bar{g}$ 0 c
b 1 2 1
$\bar{a}$ $\bar{c}$ d e
(f)

EXAMPLES OF REGULAR LATTICES. II


Akers Array
and standard 2x2 lattice

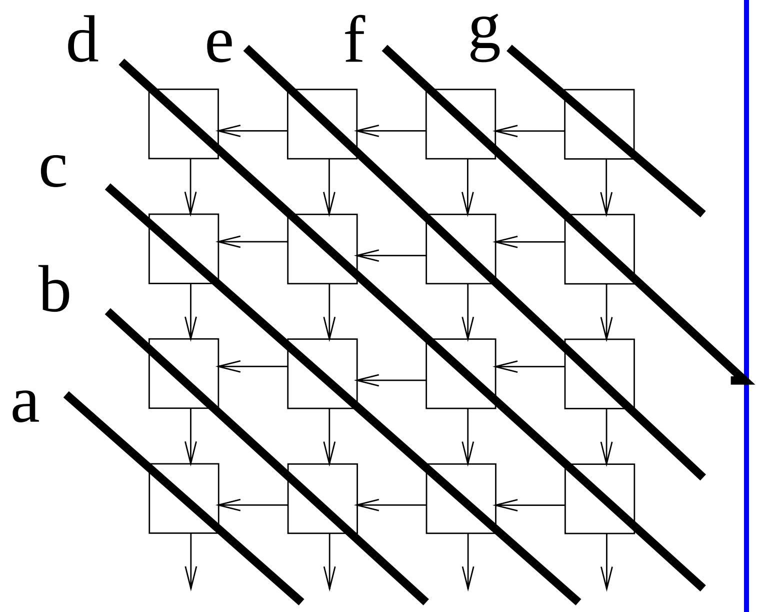

3x3 lattice,
one diagonal connection added

EXAMPLES OF REGULAR LATTICES. III



Array without diagonal buses

horizontal and vertical buses
used for connections



Symmetric function,
Variables on diagonals
are not repeated

MULTI-VALUED SHANNON EXPANSIONS AND JOININGS

- The method to create Shannon Lattices can be easily expanded to MV Shannon expansions 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.

SOP EXPANSIONS

- In binary SOP expansions 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"** type of lattices.

LINEARLY-INDEPENDENT EXPANSIONS AND JOININGS

- Linearly Independent expansions for binary case use EXOR gates.
- Generalizations of Davio expansions.
- For finite multiple-valued logic they are based on Galois Field Addition gate.
- For arbitrary algebras, they should have at least one **linear** (group) operation.
- Usually based on the algebraic structure of an arbitrary **field**.
- 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),
  - EXOR Ternary expansions.
- Joining operations for these expansions are more complicated.

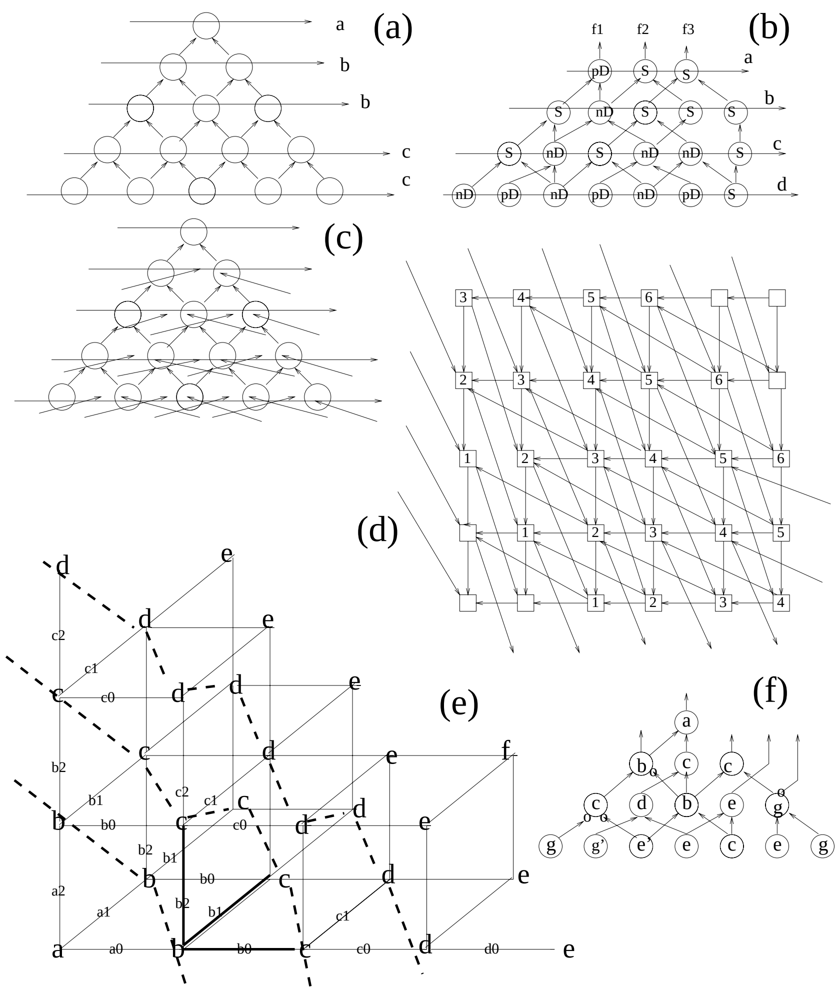
THE PROCESS OF LATTICE CREATION

- 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).
- 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.

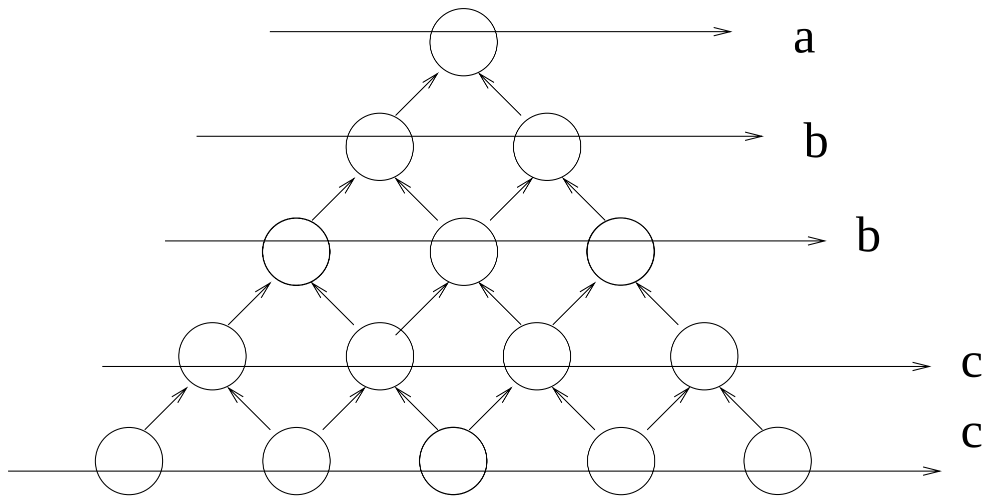
THE PROCESS OF LATTICE CREATION. II

- One approach to the variable order and expansion types selection is based on **generalized partial symmetries** for cofactors.
- We **demonstrated** that for real-life binary benchmark functions, and starting from the decompositional hierarchy of partitioning variables, the overhead of variable repeating in **planary lattices** was not excessive in each decomposed block.
- This is because symmetric and nearly symmetric blocks are preferred by our Curtis-like decomposer.

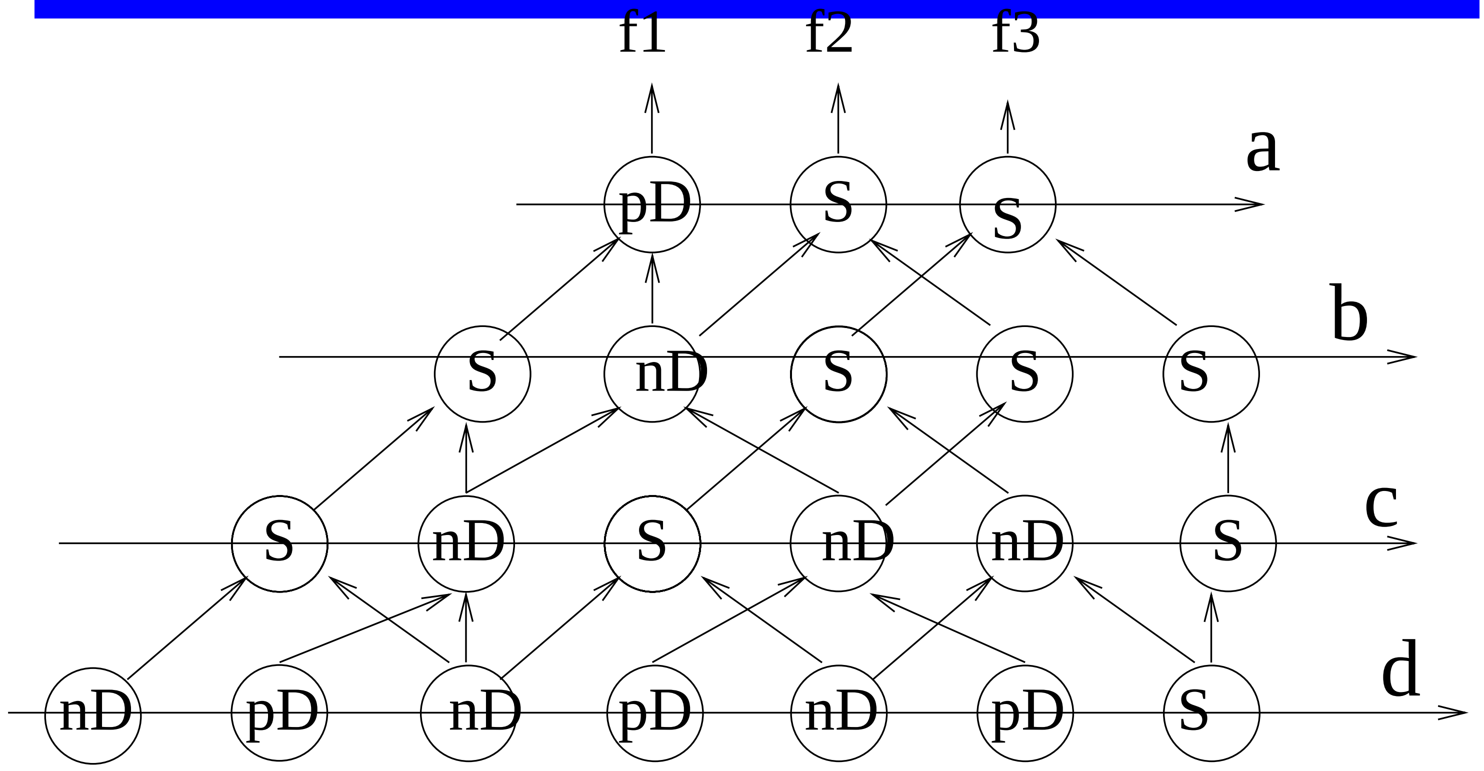
OTHER REGULAR LATTICES. I



BINARY LATTICE WITH REPEATED VARIABLES IN DIAGONAL BUSES



3x3 LATTICE, 3-OUTPUT FUNCTION, PSEUDO-KRONECKER



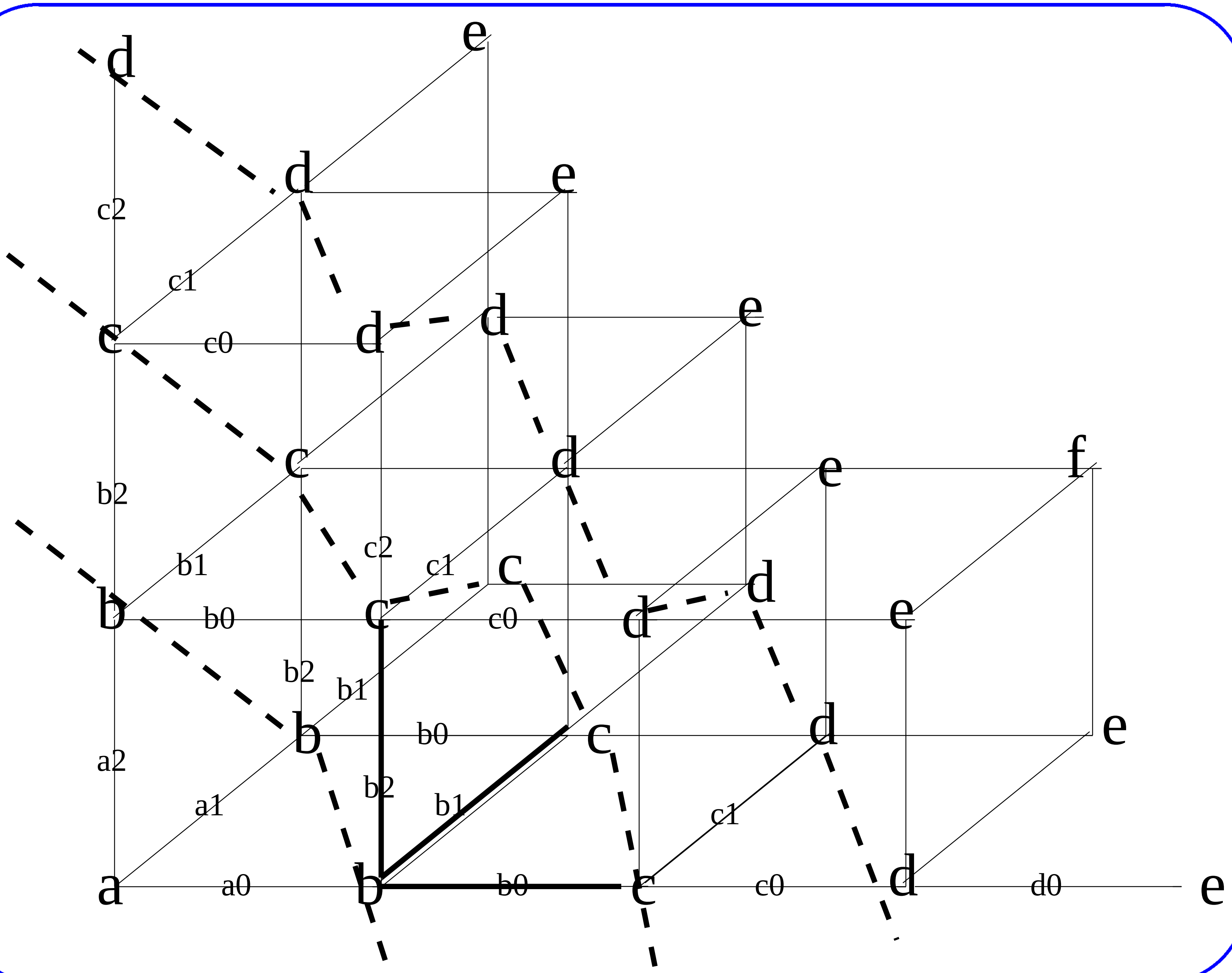


Figure 2: *Ternary Lattice in Three-Dimensional Universe, and a way of*

2x2 footnotesize REGULAR LAYOUT GEOMETRIES

- In **case of 4 neighbors**, 2x2 cells, the lattice is **planar** and it is based on a rectangular grid.
- Cell has two inputs and two outputs.
- The structure generalizes the known **switch realizations** of symmetric binary functions, based on Shannon expansion.
- The same structure for Positive and Negative **Davio** expansions, **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, Pseudo-Kronecker, and Free Decision Diagrams.
- **Theorem** Every non-symmetric function can be **symmetrized by repeating variables**.

- The selection of the next variable explained using the **Repeated Variable Maps**.
- Modern technological realizations allow to have **more than one** control variable in a level.
- **All three types of buses** (vertical, horizontal and diagonal) are used to lead any variable to the circuit's levels.

GENERALIZATIONS TO 2x2 LATTICE DIAGRAMS

- Structures **without diagonal** buses are possible.
- Inputs and outputs can occur **at avery point** inside the lattice.
- **Pairs or triplets** of binary control variables can be used in nodes.
- **Multivalued controls** can be used.
- For a 4-neighbor lattice geometry, **any canonical form** of Reed-Muller logic and its Linearly Independent generalizations can be realized.
- Any MV logic (for instance in GF(4)) 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.
- Ternary and quaternary lattices for binary, multiple-valued and continuous functions.

3x3 REGULAR LAYOUT GEOMETRIES

- Every cell has three inputs (from N, NE and E) and three outputs (to W, SW and S).
- This allows for realizations of:
  - generalized ternary diagrams (for binary EXOR logic).
  - arbitrary expansion-based Post logic.
  - GF(3) logic functions.
  - 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.

4x4 REGULAR LAYOUT GEOMETRIES

- Every cell has four inputs (from N,NE,NW, and E), and four outputs (to S, SW, SE, and W).
- This allows realizations of:
  - generalized quaternary diagrams (for GF(4)),
  - arbitrary expansion-based Post or GF(k), $k < 4$ functions.
  - Any binary or MV logic can be mapped, and more efficiently so.

APPLICATIONS OF LATTICE DIAGRAMS

- 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.
- Design methods are very efficient especially for **strongly unspecified functions,** the more unspecified the function, the better the results.

CONCLUSION

- New methods introduced, of interest to **deep sub-micron technology** and **pass-transistor** design for binary and MV gates.
- starting from **all possible** neighbor geometries in two and three dimensional spaces, we create **all possible regular structures**.
- This **extends** previous planar geometries (Akers, Chrzanowska-Jeske).
- We design **arbitrary expansions** for the new 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.**

- The same, **layout-driven synthesis approaches** are created for binary, multivalued, linearly-independent, Galois and continuous functions.
- The presented approach **generalizes and unifies** many known expansions, decision diagrams, and regular layout geometries.