Polarized Pseudo-Kronecker Symmetry with an Application to the Synthesis of Lattice Decision Diagrams

July 7, 1997

Abstract

Layout-driven logic synthesis takes into account logic design concurrently with layout issues in order to design, for deep submicron technologies, fast circuits with short connections. The approach of lattice diagrams [10] constructs for a combinational function a regular lattice with only local connections and input buses, that is directly mappable to layout. It decomposes function symmetrically, using Shannon and Davio expansions and repeating variables when necessary. Although it was proven that for every order of input variables and every expansion types applied in levels, a lattice can be constructed, the sizes of the lattices differ considerably on these parameters. This paper proposes new types of symmetries of Boolean functions, and shows how they can be used in a new approach to the important problem of finding variable-expansion orders for the lattice. Although this research originates from technological considerations, a general problem in functional decomposition is solved: how to decompose a function to a regular structure of simpler functions using symmetries. It has also applications in functional decomposition and Machine Learning.

1 Introduction

Traditionally, implementation of circuits has been broken into disjoint tasks of logic synthesis and physical design. First, a synthesizer estimates circuit area and speed independently of target technology. A layout engine would then map synthesis results to hardware.

This approach works well with small, sparsely connected circuits and with older manufacturing processes in which device delays dominate timing. For such circuits and processes, a good area metric is the total number of devices used and a good speed metric is the length of the longest chain of devices. Area metrics based entirely on device count are used because small circuits require relatively fewer wires than modern large-scale designs, and because wires can pass over regions of die already used to implement transistors. Speed metrics based entirely on device count are also fairly accurate because interconnect RC constants typically contribute negligible amounts to total delay for old processes.

As circuit size and connectivity swell and gate dimensions approach .1 micron, problems of logic synthesis and physical design become more closely interrelated. For ULSI and deep sub-micron processes, synthesis processes are required which optimize decomposition with respect to implementation resource topology. Synthesis results should map directly to hardware with minimal interconnect delays.

One way to control the complexity of such layout-driven synthesis is to use regular hardware architectures that are tuned to specific types of logic decomposition. An early example of this is Akers' rectangular logic array [1] which is a two-dimensional, planar grid of small cells. A benefit of this structure is that each cell connects to its four neighbors, thereby limiting the distance most signals are required to travel. However, since Akers designed his synthesis method for worst-case problems, it produces very large and sparse arrays for many common circuits.

Akers' approach could benefit from symmetry information of functions to be processed. Symmetry indicates parts of a function that are used more than once. Methods which detect symmetry can save area by implementing symmetrical parts only one time in hardware. Target architectures optimized for this type of subfunction sharing can also contribute to smaller interconnect delays.

This paper first describes symmetry as it is typically defined in literature, and then extends this concept to the more general **polarized pseudo-Kronecker symmetry** (PPKS). The following section introduces a two-dimensional, planar topology to illustrate how polarized pseudo-Kronecker symmetry can be applied to a specific architecture. The choice of architecture is not essential; we believe this concept is extensible to any geometry. Section 4 formally defines the problem of layout to our example architecture, section 5 describes a heuristic synthesis method based on polarized pseudo-Kronecker symmetry to solve this problem, and section 6 gives experimental results.

2 Standard Symmetry and Polarized Pseudo-Kronecker Symmetry

Most research on symmetry defines *symmetry* as a property of a function $f(X)$ with respect to a subset λ of variables in which f is invariant for any permutation of variables in λ. Subset λ is called a *symmetry set*, or if $|\lambda| = 2$, a *symmetry pair*. For the remainder of this paper, symmetry of this restricted type is prefaced with the modifier *standard*, e.g. standard symmetry, standard symmetry pair, etc.

In [4], the concept of basic symmetry is expanded to include several new classifications. Function $f(X)$ is *Non-equivalent symmetric* with respect to symmetry pair λ if equation 1 holds, and function $f(X)$ is *Equivalent symmetric* if equation 2 holds.

$$f(X-\lambda,0,1) = f(X-\lambda,1,0) \tag{1}$$
$$f(X-\lambda,0,0) = f(X-\lambda,1,1) \tag{2}$$

If f is invariant when only one variable in λ changes, f exhibits *Single-Variable Symmetry* with respect to λ.

Tsai and Marek-Sadowska [14] introduce *skew symmetries* for functions which change only polarity in response to a symmetry pair permutation described above. For example, if $f(X,1,1) = 0$ and $f(X,0,0) = 1$, f is said to be *skew equivalent symmetric*.

All symmetry relations described above involve comparing different cofactor functions with respect to a given pair of variables. Polarized pseudo-Kronecker symmetry is more general than standard symmetry in several ways. First, polarized pseudo-Kronecker symmetry is a relationship among subfunctions created during arbitrary **nonsingular expansions of functions** [7, 6, 8, 11, 12, 9] not just among positive and negative cofactors. This generalization allows for expansions other than Shannon. For instance, the simplest nonsingular expansions are Davio expansions [13] which generate exclusive sums of two cofactors. Polarized pseudo-Kronecker symmetry gets its name, in part, from the observation that this kind of symmetry can describe equivalence among nodes of free, pseudo-Kronecker Functional Decision Diagram (FDD) representations of functions (pseudo-Kronecker diagrams are explained in [13], free diagrams in [5] and applications of pseudo-Kronecker concepts for symmetic functions in [3]).

Polarized pseudo-Kronecker symmetry further expands standard symmetry by loosening restrictions on polarities of variables and subfunctions used for expansion. For example, for a function f which is decomposed (i.e. expanded) twice by Shannon expansion – first by a, then by b – standard symmetry holds when:

$$f_{b_1}(f_{a_0}) = f_{b_0}(f_{a_1}) \tag{3}$$

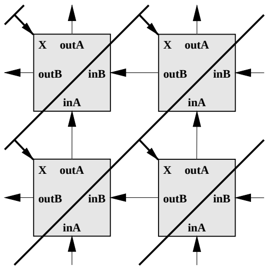

Figure 1: *Lattice L_0 Interconnect Pattern*

where f_{a_0} and f_{a_1} are negative and positive cofactors of f with respect to a. For the case of Shannon expansions, if either a or b is negated, equation 3 becomes equivalent symmetry as defined in [4]. Also, if either cofactor with respect to b is negated, equation 3 becomes skew symmetry as defined in [14].

The power of polarized pseudo-Kronecker symmetry over previous symmetry definitions can only be seen when applied to Davio expansions. For example, if function f is decomposed by positive Davio and next negative Davio expansions – using a and b respectively – one possible polarized pseudo-Kronecker symmetry holds when:

$$f_{\overline{b}_1}(f_{a_0}) = f_{\overline{b}_2}(\overline{f_{a_2}}) \tag{4}$$

where f_{a_2} is the exclusive sum of negative and positive cofactors of f with respect to a. In this example, subfunction f_{a_2} and expansion variable b are negated.

3 Polarized Pseudo-Kronecker Symmetry for 2x2 Lattices

To illustrate polarized pseudo-Kronecker symmetry, we use a lattice geometry similar to Akers' array. A *2x2 lattice* is planar array of hardware cells in which each cell is adjacent to four other cells. Cells input one signal each from two neighboring cells (in_A and in_B) and provide one output each for the remaining two neighbors (out_A and out_B).

In addition to such local interconnects which transmit information between adjacent cells, global interconnects distribute common signals to multiple cells. A third cell input, x, is a tap off a diagonal global wire. All cells on the same positive diagonal connect to the same global interconnect and therefore have the same value on their x input pins. (See Figure 1.)

Cells of our 2x2 lattice L_0 contain a pair of AND gates which feed an EXOR gate which, in turn, produces a signal for outputs out_A and out_B. The library of implementable logic functions includes three important equations,

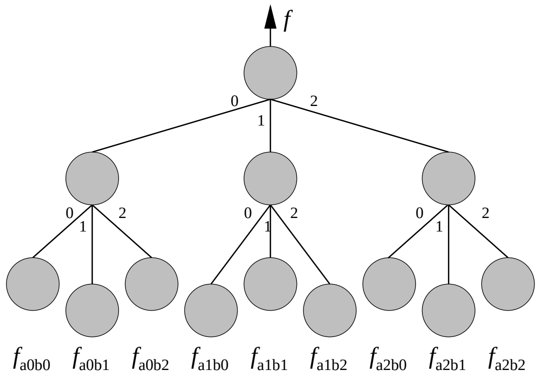

Figure 2: *Nine Subfunctions for EXOR Ternary Expansion with Respect to Variables a and b*

$$out_A = out_B = x \cdot in_A \oplus \overline{x} \cdot in_B \quad (5)$$
$$out_A = out_B = x \cdot in_A \oplus in_B \quad (6)$$
$$out_A = out_B = in_A \oplus \overline{x} \cdot in_B \quad (7)$$

Equations 5 through 7 describe the Shannon, positive Davio and negative Davio expansions, respectively. Input x corresponds to the variable of decomposition, and in_A and in_B correspond to the logic functions of negative cofactors, positive cofactors, or the exclusive sum of cofactors, depending on which type of expansion is used.

In addition, cells can implement various symmetry polarities by using normal or inverted versions of any input.

We make several assumptions to simplify our example. Functions are represented by ordered Kronecker FDDs (OKFDDs) rather than free pseudo-Kronecker diagrams. In addition, we consider only symmetry pairs rather than symmetry sets of greater cardinality. Also, only a subset of possible symmetry polarities are inspected. Given these simplifications, there are three general cases in which polarized pseudo-Kronecker symmetry among OKFDD nodes is useful in mapping functions to 2x2 lattices. These symmetry types are called *constant symmetry*, *single-variable symmetry* and *double-variable symmetry*.

To describe symmetry types, we introduce an EXOR-ternary diagram (ETDD) [13] of function $f(a, b, \cdots)$, shown in Figure 2. The ETDD shows all unique subfunctions of f that are generated by Shannon and Davio expansions with respect to variables a and b. Only the first two levels are shown. Subsequent levels of the tree are absent for the sake of clarity.

Each edge is labeled with a 0, 1, or 2, representing negative cofactors f_0, positive cofactors f_1, and exclusive sums of cofactors $f_0 \oplus f_1$, respectively. $f = \bar{a} f_{\bar{a}} \oplus a f_a (Shannon)$. Then, $f = (1 \oplus a) f_{\bar{a}} \oplus a f_a$ $f = a(f_{\bar{a}} \oplus f_a) \oplus f_{\bar{a}} = a f_2 \oplus f_0 (positive Davio)$. Similarly, $f = \bar{a}(f_{\bar{a}} \oplus f_a) \oplus f_a = \bar{a} f_2 \oplus f_1 (negative Davio)$. From these equations, labels 0 and 1 are associated with Shannon, 0 and 2 with Positive Davio, and 1 and 2 with Negative Davio expansions. Subfunctions of f are labeled with a and b and with type of subfunctions associated with a and b. For example, subfunction $f_{a_0 b_2}$ is obtained by traversing the tree in Figure 2 from the top (output) node to the next level via edge 0, and to the last level via edge 2. Out of nine subfunctions, we want to select those that have some of the discussed relations. Thus, 3 or less subfunctions will be created if the symmetrical subfunctions exist among 9 subfunctions.

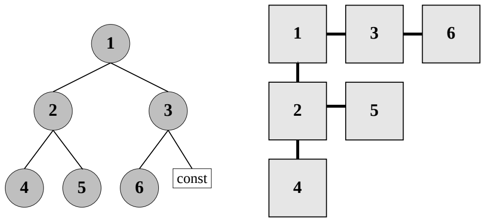

Figure 3: *Constant PPKS and its Realization in Lattice L_0*

$f_{a_0 b_0}$ = constant	$f_{a_0 b_1}$ = constant
$f_{a_0 b_2}$ = constant	$f_{a_1 b_0}$ = constant
$f_{a_1 b_1}$ = constant	$f_{a_1 b_2}$ = constant
$f_{a_2 b_0}$ = constant	$f_{a_2 b_1}$ = constant
$f_{a_2 b_2}$ = constant	

Table 1: *Nine Possible Constant PPKSs*

3.1 Constant Symmetry

For constant symmetry, one of the nine subfunctions on the second level of a function's ETDD is constant. This type of symmetry is not a relationship between two subfunctions, but rather a property of a single subfunction. If a subfunction is constant, all further expansions of the subfunction with respect to any variable will remain constant.

In the example shown in Figure 3a, node 2 only requires a single external signal because the constant can be generated within a cell of L_0, a two-level lattice. Figure 3b demonstrates how such a tree maps to our lattice, and Table 1 shows the nine possible constant PPKSs.

3.2 Single-Variable Symmetry

For *single-variable symmetry pair*, two subfunctions

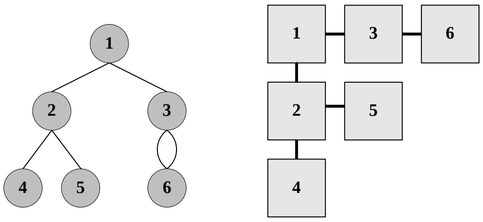

Figure 4: *Single-Variable Symmetric PPKS and its Realization in Lattice L_0*

$f_{a_0b_0} = f_{a_0b_1}$	$f_{a_0b_1} = f_{a_0b_2}$	$f_{a_0b_0} = \overline{f_{a_0b_1}}$	$f_{a_0b_1} = \overline{f_{a_0b_2}}$
$f_{a_0b_2} = f_{a_0b_0}$	$f_{a_1b_0} = f_{a_1b_1}$	$f_{a_0b_2} = \overline{f_{a_0b_0}}$	$f_{a_1b_0} = \overline{f_{a_1b_1}}$
$f_{a_1b_1} = f_{a_1b_2}$	$f_{a_1b_2} = f_{a_1b_0}$	$f_{a_1b_1} = \overline{f_{a_1b_2}}$	$f_{a_1b_2} = \overline{f_{a_1b_0}}$
$f_{a_2b_0} = f_{a_2b_1}$	$f_{a_2b_1} = f_{a_2b_2}$	$f_{a_2b_0} = \overline{f_{a_2b_1}}$	$f_{a_2b_1} = \overline{f_{a_2b_2}}$
$f_{a_2b_2} = f_{a_2b_0}$		$f_{a_2b_2} = \overline{f_{a_2b_0}}$	

Table 2: *Nine Single-Variable Symmetric PPKSs and Nine Skew PPKSs*

adjacent to the same parent node are equal and can be combined into a single node. Since cells of a 2x2 lattice can invert any input, we can also combine pairs of nodes which represent two subfunctions that differ only in polarity. If nodes are combined after an inversion, the pair of expansion variables associated with such nodes is a *skew single-variable symmetry pair.* Figure 4 shows a single-variable symmetry pair and its realization in a 2x2 lattice.

Of the nine possible subfunctions of a two-variable expansion, there are three sets of functions that are children of the same parent node: $f_{a_0b_0}$, $f_{a_0b_2}$ and $f_{a_0b_2}$ are adjacent to f_{a_0}; $f_{a_1b_0}$, $f_{a_1b_2}$ and $f_{a_1b_2}$ are adjacent to f_{a_1}; and $f_{a_2b_0}$, $f_{a_2b_2}$ and $f_{a_2b_2}$ are adjacent to f_{a_2}. Within each set, there are three different function combinations possible.

Table 2 summarizes nine instances of single-variable symmetric PPKS which use exclusively non-inverting edges and nine instances of skew single-variable symmetric PPKS.

3.3 Double-Variable Symmetry

For *double-variable symmetry pair*, two subfunctions nodes that are not adjacent to the same parent node are combined. Figure 5 shows a double variable symmetry pair and its realization in a 2x2 lattice.

There are more instances of double-variable symmetry pairs than single-variable symmetry pairs. Each subfunction can combine with any of six other functions that do not share its parent node. Table 3 summarizes twenty-seven instances of double-variable symmetries which use no inversions and twenty-seven instances of double-variable symmetries in which one

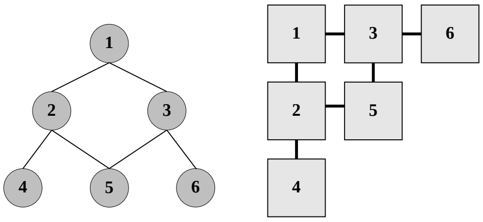

Figure 5: *Double-Variable Symmetric PPKS and its Realization in Lattice L_0*

$f_{a_0b_0} = f_{a_1b_0}$	$f_{a_0b_0} = f_{a_1b_1}$	$f_{a_0b_0} = \overline{f_{a_1b_0}}$	$f_{a_0b_0} = \overline{f_{a_1b_1}}$
$f_{a_0b_0} = f_{a_1b_2}$	$f_{a_0b_0} = f_{a_2b_0}$	$f_{a_0b_0} = \overline{f_{a_1b_2}}$	$f_{a_0b_0} = \overline{f_{a_2b_0}}$
$f_{a_0b_0} = f_{a_2b_1}$	$f_{a_0b_0} = f_{a_2b_2}$	$f_{a_0b_0} = \overline{f_{a_2b_1}}$	$f_{a_0b_0} = \overline{f_{a_2b_2}}$
$f_{a_0b_1} = f_{a_1b_0}$	$f_{a_0b_1} = f_{a_1b_1}$	$f_{a_0b_1} = \overline{f_{a_1b_0}}$	$f_{a_0b_1} = \overline{f_{a_1b_1}}$
$f_{a_0b_1} = f_{a_1b_2}$	$f_{a_0b_1} = f_{a_2b_0}$	$f_{a_0b_1} = \overline{f_{a_1b_2}}$	$f_{a_0b_1} = \overline{f_{a_2b_0}}$
$f_{a_0b_1} = f_{a_2b_1}$	$f_{a_0b_1} = f_{a_2b_2}$	$f_{a_0b_1} = \overline{f_{a_2b_1}}$	$f_{a_0b_1} = \overline{f_{a_2b_2}}$
$f_{a_0b_2} = f_{a_1b_0}$	$f_{a_0b_2} = f_{a_1b_1}$	$f_{a_0b_2} = \overline{f_{a_1b_0}}$	$f_{a_0b_2} = \overline{f_{a_1b_1}}$
$f_{a_0b_2} = f_{a_1b_2}$	$f_{a_0b_2} = f_{a_2b_0}$	$f_{a_0b_2} = \overline{f_{a_1b_2}}$	$f_{a_0b_2} = \overline{f_{a_2b_0}}$
$f_{a_0b_2} = f_{a_2b_1}$	$f_{a_0b_2} = f_{a_2b_2}$	$f_{a_0b_2} = \overline{f_{a_2b_1}}$	$f_{a_0b_2} = \overline{f_{a_2b_2}}$
$f_{a_1b_0} = f_{a_2b_0}$	$f_{a_1b_0} = f_{a_2b_1}$	$f_{a_1b_0} = \overline{f_{a_2b_0}}$	$f_{a_1b_0} = \overline{f_{a_2b_1}}$
$f_{a_1b_0} = f_{a_2b_2}$	$f_{a_1b_1} = f_{a_2b_0}$	$f_{a_1b_0} = \overline{f_{a_2b_2}}$	$f_{a_1b_1} = \overline{f_{a_2b_0}}$
$f_{a_1b_1} = f_{a_2b_1}$	$f_{a_1b_1} = f_{a_2b_2}$	$f_{a_1b_1} = \overline{f_{a_2b_1}}$	$f_{a_1b_1} = \overline{f_{a_2b_2}}$
$f_{a_1b_2} = f_{a_2b_0}$	$f_{a_1b_2} = f_{a_2b_1}$	$f_{a_1b_2} = \overline{f_{a_2b_0}}$	$f_{a_1b_2} = \overline{f_{a_2b_1}}$
$f_{a_1b_2} = f_{a_2b_2}$		$f_{a_1b_2} = \overline{f_{a_2b_2}}$	

Table 3: *Twenty-Seven Double-Variable Symmetric PPKSs and 27 Skew PPKSs*

subfunction is inverted, called *skew double-variable symmetries*.

It should be noted that the list of symmetries defined above is redundant and is not the minimum set of independent symmetries, but these symmetries were formulated with respect to the ease of their calculation, breadth of applications, and similarity of formulas for binary and multivalued logic. Some of these symmetries are implied by one or more of other symmetries. For instance, $f_{a_0b_2} = f_{a_0b_0} = f_{\bar{a}\bar{b}} \oplus f_{\bar{a}b} = f_{\bar{a}\bar{b}}$ implies $f_{\bar{a}b} = 0$, which is one of two cases, $f_{a_0b_1} = 0$, of "constant symmetry" $f_{a_0b_1} = \text{const}$. Similarly, symmetry $f_{a_0b_0} = f_{a_2b_1}$ is equivalent to $f_{a_0b_2} = f_{a_1b_1}$. This means, that not all above symmetries should be always checked, because positive or negative verification of some of them excludes the necessity of checking another ones. It is possible to create various smaller but "minimal" sets of symmetries. We are working on defining minimum sets of independent minimal symmetries to be checked for various binary and multi-valued applications.

3.4 Example of Finding Symmetry Pairs

This section presents an example of finding polarized pseudo-Kronecker symmetry pairs and illustrates the interdependence among such pairs.

Given is function f of variables (a, b, c, d, e):

$$f = \bar{a}\bar{c}\,\bar{d} \oplus \bar{a}\bar{b} \oplus c\bar{d} \oplus \bar{a}cde \oplus a\bar{b}c$$

Symmetries between variables a and b are found by first calculating the three possible subfunctions of variable a:

$$f_{a_0} = \bar{c}\,\bar{d} \oplus \bar{b} \oplus c\bar{d} \oplus cde f_{a_1} = c\bar{d} \oplus \bar{b}c$$
$$f_{a_2} = (\bar{c}\,\bar{d} \oplus \bar{b} \oplus c\bar{d} \oplus cde) \oplus (c\bar{d} \oplus \bar{b}c) =$$
$$\bar{d} \oplus \bar{b}\,\bar{c} \oplus cde \oplus c\overline{dd}\,\bar{c} \oplus \bar{b}\,\bar{c} \oplus cde.$$

and then the nine possible subfunctions of variable b:

$$f_{a_0b_0} = \bar{c}\,\bar{d} \oplus 1 \oplus c\bar{d} \oplus cde$$
$$f_{a_0b_1} = \bar{c}\,\bar{d} \oplus c\bar{d} \oplus cde$$
$$f_{a_0b_2} = 1$$
$$f_{a_1b_0} = cd \quad f_{a_1b_1} = c\bar{d} \quad f_{a_1b_2} = c$$
$$f_{a_2b_0} = \bar{d} \oplus \bar{c} \oplus cde \oplus c\bar{d}$$
$$f_{a_2b_1} = \bar{d} \oplus cde \oplus c\bar{d} \quad f_{a_2b_2} = \bar{c}$$

Subfunctions $f_{a_0b_0}$ and $f_{a_0b_1}$ are mutual negations (and can be combined in a lattice using an inverter). Since these functions are children of the same function, f_{a_0}, there is a single variable symmetry between a and b.

There is also a constant symmetry between a and b because one of their nine subfunctions, $f_{a_0b_2}$, is constant.

In addition, subfunctions $f_{a_1b_2} = c$ and $f_{a_2b_2} = \bar{c}$ are skew symmetric. Since one function is derived from f_{a_1} and the other is a child of f_{a_2}, this type of relationship is a skew double-variable symmetry between a and b. There are often several types of symmetries for the same variable pair. To illustrate how symmetry choice effects other variable pairs, we construct the first two levels of a FDD of function f, using a variable ordering that starts with a and b.

In Figure 6a, we choose constant symmetry. Since we need subfunction $f_{a_0b_2}$, we decompose by a using Shannon or positive Davio. In the first expansion, we get f_{a_0} and f_{a_1}, and in the second expansion we get f_{a_0} and f_{a_2}. Then we decompose f_{a_0} by b using negative or positive Davio. These expansions will produce f_{a_2} and f_{a_1} or f_{a_2} and f_{a_0}, respectively.

In Figure 6b, we choose single-variable skew symmetry. Since we need subfunctions $f_{a_0b_0}$ and $f_{a_0b_1}$, we decompose by a using Shannon or positive Davio and then by b using Shannon. These expansions produce $f_{a_0b_0}$ and $f_{a_0b_1}$ regardless of how we expand our first level, then $f_{a_1b_0}$ and $f_{a_1b_1}$ or $f_{a_2b_0}$ and $f_{a_2b_1}$, for Shannon or positive Davio, respectively. We negate one subfunction in order to combine functions.

In Figure 6c, we choose double-variable skew symmetry. Since we need subfunctions $f_{a_1b_2}$ and $f_{a_2b_2}$, ($f_{a_1b_2} = \overline{f_{a_2b_2}} = c$ was the only identity), our only choice is to decompose by a using negative Davio and by b using positive Davio.

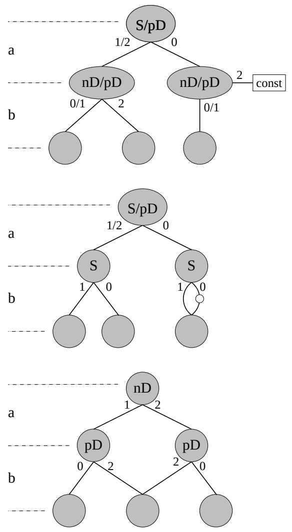

Figure 6: *Selection of Expansions (a) Constant PPKS Between Variables a and b (b) Single-Variable Symmetric PPKS Between Variables a and b (c) Double-Variable Symmetric PPKS Between Variables a and b*

4 Definition of Lattice Layout Problem

The final goal of our synthesis approach is to order input variables of a function such that the OKFDD representation of the function, when expanded by variables in the given sequence, fits a lattice pattern and occupies as small a region of lattice cells as possible. At present, we simplify the metric for quality of results to be the total number of levels in a decision diagram. Therefore, the best variable order will be defined as one with the fewest number of repeated variables (Every function can be realized in a lattice with repeated variables [2, 10]).

LATTICE DECISION DIAGRAM: A decision diagram D is a *lattice decision diagram* (LDD) with respect to a lattice L if the logic of any node in D can be implemented by any cell of L, and if the attributes of any edge in D can be implemented by a wire of L, and the pattern of edges connecting nodes of D can be implemented by interconnect resources of L in a regular pattern. Figure 6 shows examples of an LDDs for L_0.

For a decision diagram to be a LDD with respect to our lattice, the decision diagram must be a Kronecker Decision Diagram (KDD) with standard or inverted edges. This requirement assures that all graph nodes and edges can be realized by cells and wires of our lattice. Not all KDDs are LDDs with respect to our lattice, however. The local connection topology of a KDD must conform to the pattern of possible interconnects present in L_0.

LATTICE PATTERN: Two adjacent FDD levels a and b fit a *lattice pattern* if the set of edges between nodes of a and b of a decision diagram conform to the pattern of interconnects of a given lattice.

To simplify our synthesis method, we require that the order of variables used to expand a LDD is constant for all branches of a tree, i.e. LDDs must be ordered Kronecker decision diagrams (OKFDD). For this kind of representation, a one-to-one mapping exists between diagram levels and function variables. Subsequent definitions and discussion refer to FDD level a as the group of decision diagram nodes which are expanded by variable a.

NATURAL LATTICE RELATIONSHIP: Variables a and b are defined to be in a *natural lattice relationship* if their corresponding FDD levels fit a lattice pattern. Variables a and b form therefore the *NLR variable pair*. Observe that the existence of one of PPKS symmetries is a neccessary but not sufficient condition of two variables to form an NLR variable pair. Heuristically, the more symmetries between two variables, the more arguments that they will become an NLR variable pair.

FORCED LATTICE RELATIONSHIP: Variables a and b have a *forced lattice relationship* if one or both of the

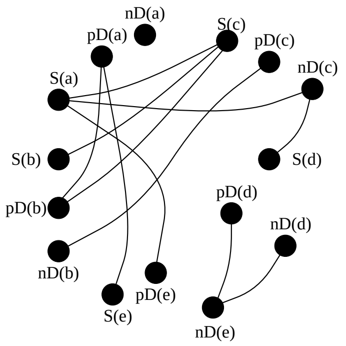

Figure 7: *Symmetry Compatibility Graph for* $f_2(a, b, c, d, e)$

variables are used for decomposition multiple times. Variables a and b form an *FLR variable pair*. Methods for creation of such lattices are presented in [2, 10].

5 A Synthesis Method for 2x2 Lattices Based on Polarized Pseudo-Kronecker Symmetry

The ultimate goal of our synthesis approach is to order input variables of a function such that the OKFDD representation of the function, when expanded by variables in the given sequence, fits a lattice pattern and occupies as small a region of lattice cells as possible. At present, we simplify the metric for quality of results to be the number of levels in a decision diagram. Therefore, the best variable order is one with the fewest number of repeated variables.

Below we will present a heuristic approach to select the order of repeated variables and their associated expansion types. Our algorithm uses the concept of a Symmetry Compatibility Graph (G_{SYM}) to record polarized pseudo-Kronecker symmetry pairs (Figure 7). A node of G_{SYM} represents an expansion (Shannon, positive or negative Davio) with respect to an input variable. For an input a, therefore, G_{SYM} contains nodes *a-S*, *a-pD*, and *a-nD*, representing a Shannon expansion by a, a positive Davio expansion by a and a negative Davio expansion by a, respectively. We call these *variable-expansion pairs*. Edges between nodes indicate symmetries.

Our top-level function, `FindOrder`, is given a function f and a list of variables *Inputs*. (See Figure 8) Initially, G_{SYM} is cleared of all edges. For every unique variable pair, function `ExorTernaryExpansion` creates an EXOR-ternary functional decision diagram (ETDD) representation of f, using variables of the current variable pair. ETDD nodes on the second level below the tree's output node represent functions called the *subfunctions* of f with respect the current variable pair. Note that unlike binary expansions Shannon and Davio, the EXOR-ternary expansion creates three edges from each node. After expansion with respect to two variables, nine subfunction nodes are created.

Function `FindSym` inputs an array of subfunction nodes and checks the values of all nodes for PPKSs. For each PPK symmetry found by `FindSym`, an edge is added to G_{SYM}.

After all variable pairs have been considered and the Symmetry Compatibility Graph has been completed, function `PseudoHamiltonian` analyzes G_{SYM} for pseudo-Hamiltonian paths. We call it Pseudo-Hamiltonian because this is not a Hamiltonian path in known mathematic sense, but is quite similar. Finding such paths in G_{SYM} is also unlike the traditional traveling salesman problem (TSP) in two ways. First, edges in TSP are weighted while all edges are of unit weight in our version of the problem. Second, in TSP there is no relationship among nodes while nodes of the Symmetry Compatibility Graph are grouped with other nodes of the same variable. If a pseudo-Hamiltonian path through G_{SYM} contains node a-S, for example, it is not desirable to also contain a-pD. This would mean that variable a is used twice for expansion, once using Shannon and once using positive Davio.

The function `FindFlr` takes path information from function `PseudoHamiltonian` and generates the final variable-expansion ordering by forcing lattice relationships between variables in separate paths until all paths are connected and every input variable appears in the final ordering. For each FLR ordering, a variable is repeated in the ordering and the FDD height increases. If FLR appears between paths of symmetric variables, that symmetry is lost. This algorithm does not guarantee creating a lattice but the OKFDD created with the order found in it has upper part being a lattice and is useful for further transformations. When a subset of variables is totally symmetric, a maximum clique corresponds to them in the G_{SYM} graph. Thus a good heuristic to find a pseudo-Hamiltonian path that would lead to good variable-expansion orders is to find a path that the first nodes in it belong to the largest clique or "near-complete" clique with just few edges missing to make it a clique.

6 Results

We tested our algorithms with benchmarks taken from the *Wright Laboratory Data Set*. This benchmark set was selected because all circuits have only one output and a small number of inputs (fewer than 9 inputs). The benchmark set contains over 50 benchmarks, which gives us a good overview of how our NLR approach should work on much larger functions.

algorithm `FindOrder`(f, *Inputs*)
  G_{SYM} = *0*
  for (∀ *unique* (u,v) — u,v ∈ Inputs)
    subf*[9]* = *ExorTernaryExpansion(*f,u,v*)*
    edges = *FindSym(*subf*)*
    AddEdgesTo(G_{SYM}, edges*)*
  1stOrdering = *PseudoHamiltonian(*G_{SYM}*)*
  2ndOrdering = *FindFlr(*1stOrdering,Inputs*)*
  *CreateNewOrder(*1stOrdering,2ndOrdering*)*

Figure 8: *Top-level Algorithm for the LDD Synthesis Method*

6.1 Benchmark testing

Column 3 in Table 4 shows the results of running a basic symmetry check on each of the benchmark sets. Symmetry, in this case, refers to the invariance of a function when two of its input variables are swapped. Column 4 shows results for the new symmetries. Column 1 counts pairs of only non-vacuous columns thus excluding vacuous variables from comparisons. From Table 4, it is obvious that searching for all possible PPKSs will produce much better results than just looking for standard symmetry between pairs of variables.

We use the benchmark function "add2.blif", describing a combinational adder, to illustrate the difference between a standard symmetry check and a PPKS check. Several symmetrical pairs are obvious from adder's specification: (X_1, X_5),(X_2, X_6),(X_3, X_7) where variables X_0 and X_4 are vacuous and thus have no effect on the output of the "add" circuit. This observation is reflected in the symmetry compatibility graphs shown in Figures 9 and 10. Figure 9 is sparse and only shows symmetries that are obvious from inspection of the adder's circuit diagram. Figure 10, on the other hand, is close to being a complete graph and reveals symmetries that are not obvious from the circuit.

These two compatibility graphs suggest that the number of relationships found with NLR checks can exceed the number of standard symmetries by a great margin. Thus, by using the NLR approach, a much wider range of possibilities for mapping onto a lattice diagram are available.

benchmark	Non-vacuous Variable Pairs	Standard Symmetry	Polarized Pseudo-Kronecker Symmetry
add0	28	4	4
add2	15	3	11
add4	1	1	1
and_or_chain8	28	2	28
ch15f0	15	0	0
ch176f0	3	1	3
ch177f0	1	1	1
ch22f0	15	6	6
ch30f0	15	1	2
ch47f0	15	0	1
ch52f4	28	0	3
ch70f3	28	1	18
ch74f1	28	0	9
ch83f2	28	0	3
ch8f0	15	2	5
contains_4_ones	28	28	28
greater_than	28	4	4
interval1	28	0	0
interval2	28	0	0
kdd1	3	3	3
kdd10	15	3	11
kdd2	10	4	10
kdd3	6	3	6
kdd4	0	0	0
kdd5	28	1	16
kdd6	6	6	6
kdd7	28	4	4
kdd8	3	1	3
kdd9	15	7	9
majority_gate	28	28	28
monkish1	6	2	6
monkish2	28	8	8
monkish3	10	4	10
mux8	15	0	14
or_and_chain8	28	2	28
pal	28	4	4
pal_dbl_output	21	0	0
pal_output	28	0	0
parity	28	28	28
primes	28	0	7
primes8	28	0	7
rnd1	28	0	0
rnd2	28	0	0
rnd3	28	0	0
rnd_m1	28	28	28
rnd_m10	28	0	8
rnd_m25	28	0	0
rnd_m5	28	1	22
rnd_m50	28	0	0
rndvv36	15	0	0
substr1	28	0	4
substr2	28	0	8
subtraction1	28	4	4
subtraction3	1	1	1

Table 4: Frequency of Standard Versus Polarized Pseudo-Kronecker Symmetries

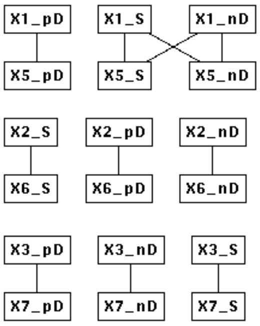

Figure 9: G_{SYM} *Graph of benchmark add2 with standard symmetries*

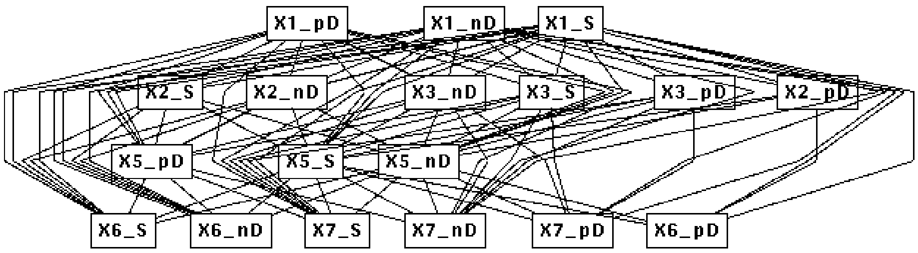

Figure 10: G_{SYM} *Graph of benchmark add2 with PPK symmetries*

6.2 Pseudo-Hamiltonian Path

The graph from Fig. 11 shows the PPKS relationships between variables. Now the problem is how to map this graph into a lattice structure. The idea is to find the longest pseudo-Hamiltonian path through the graph to produce the top of the lattice. It is expected that the longest series of variables and their expansions where each variable-expansion node has a PKK relation with immediately preceding and following variable-expansion nodes. Then by using Forced Lattice Relationship (FLR) the top of the lattice can be combined with parts of the graph that are not in the pseudo-Hamiltonian path. As in Figure 11 X_0 and X_1 are a disjoint pair from the rest of the graph. Thus the pseudo-Hamiltonian path may produce a path going from X_4 (Shannon), X_5 (Shannon), X_6 (Negative Davio) to X_2 (Negative Davio). This is mapped into the top of the lattice, leaving the bottom of the lattice to X_0 and X_1 which can be added to the lattice by using some FLR expansion on the lattice diagram.

Examples of OKFDDs obtained for function rnd_m10 with variable-expansion orderings found by `FindOrder` algorithm are shown in Figures 12 and 13. Observe that, in general, the method does not lead to a lattice (there is 5 nodes in level 4 from top of Shannon Lattice in Fig. 12, instead of 4 nodes expected in a planar 2x2 lattice). However, there is always some "upper part" of the diagram, at least with 2 levels, that is either already a lattice, or can be modified to a lattice by single repetition(s) of bottom variable(s). The order of variables found as above can be either used as a (partial) forced order to control look-ahead heuristic search algorithms like those from [2, 10] that create planar binary lattices, or to create higher order lattices, such as the ternary (with non-planar 3x3 neighborhood pattern), or quaternary (with non-planar 4x4 neighborhood pattern), presented in [11, 12]. For instance, six upper levels of the diagrams from Figures 12 and 13 can be mapped to ternary lattices. Next, other means are used to realize the bottom of the diagram.

7 Conclusion

The presented definition of PPK symmetries can also be checked for incompletely specified functions, and applied to construct lattices for them. The results of presented algorithms are very valuable for our continuing research into constructing lattices. There are some other relationships that we need to explore, like how often the skew symmetry relations occur in real life. Early analysis shows that they don't occur very often, or never occur. Of course, there are exceptions, but from running through this benchmark set, skew is not a factor.

Future work will include implementing FLR algo-

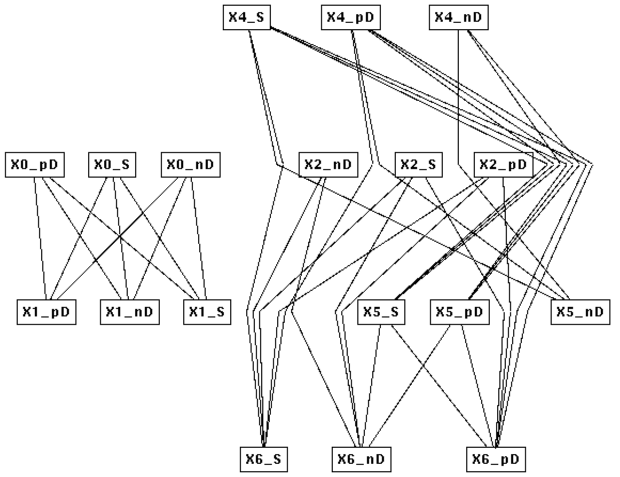

Figure 11: *Example G_{SYM} graphs for finding of pseudo-Hamiltonian Paths*

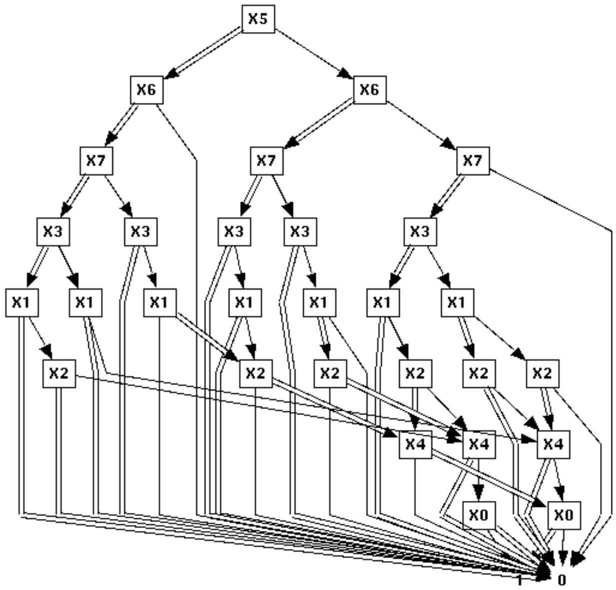

Figure 12: *rnd_m10 Shannon Lattice*

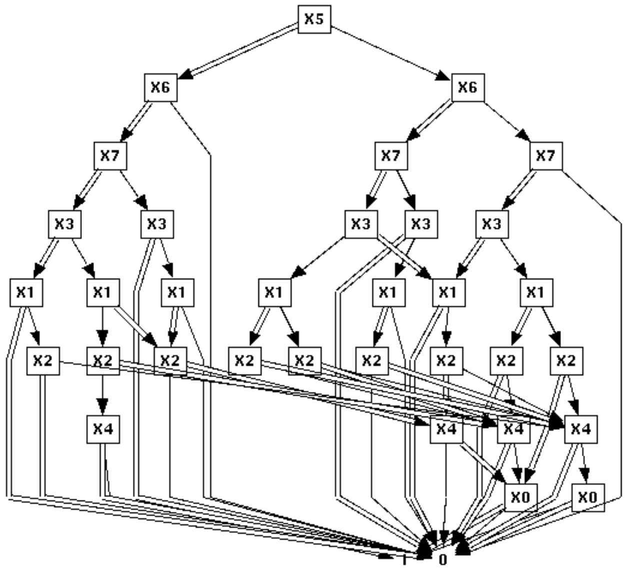

Figure 13: *rnd_m10 Shannon, PDavio, NDavio Lattice*

rithms based on the G_{SYM} graphs. Thus, searching for solutions should be much faster than searching through all possible variable-expansion orderings, and will hopefully give better results than current heuristic search methods based on look-ahead selection of variable-expansion orders while building a lattice [10]. The presented method can be also combined with other variable-expansion ordering methods that we are currently developing.

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] R. Drechsler, "Pseudo Kronecker Expressions for Symmetric Functions," *Proc. VLSI Design Conference*, 1997.

[4] C.R. Edwards, and S.L. Hurst,"A digital synthesis procedure under function symmetries and mapping methods," *IEEE Trans. Computers,* Vol. 27, No. 11, pp. 985-997, Nov. 1978.

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

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

[8] 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.

[9] 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.

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

[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, 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.

[13] T. Sasao (editor), "Logic Synthesis and Optimisation," *Kluwer Academic Publishers,* 1993.

[14] Ch. Tsai and M. Marek-Sadowska, "Detecting Symmetric Variables in Boolean Functions using Generalized Reed-Muller Forms," *Proc. ISCAS*, pp. 1,287-1,290, 1994.