VLSI DESIGN
1999, Vol. 10, No. 1, pp. 35–55
Reprints available directly from the publisher
Photocopying permitted by license only

Logic Synthesis for a Regular Layout

MALGORZATA CHRZANOWSKA-JESKE[a,*], YANG XU[b,†] and MAREK PERKOWSKI[a]

[a] *Electrical and Computer Engineering Department, Portland State University, 1800 6th Avenue, Portland, OR 97207-0751;*
[b] *Lattice Semiconductor Corporation, 5555 NE Moore Count, Hillsboro, OR 97124-0118*

(Received 7 September 1998; In final form 20 November 1998)

New algorithms for generating a regular two-dimensional layout representation for multi-output, incompletely specified Boolean functions, called, Pseudo-Symmetric Binary Decision Diagrams (PSBDDs), are presented. The regular structure of the function representation allows accurate prediction of post-layout areas and delays before the layout is physically generated. It simplifies power estimation on the gate level and allows for more accurate power optimization. The theoretical background of the new diagrams, which are based on ideas from contact networks, and the form of decision diagrams for symmetric functions is discussed. PSBDDs are especially well suited for deep sub-micron technologies where the delay of interconnections limits the device performance. Our experimental results are very good and show that symmetrization of real-life benchmark functions can be done efficiently.

Keywords: Logic synthesis, deep submicron, regular array, decision diagrams

1. INTRODUCTION

Performance improvement of ULSI ICs fabricated in deep sub-micron technology depends very strongly on the delay of interconnects. For the current leading technologies with active-device count reaching the tens of millions, the delay of interconnects is responsible for about 40–50% of the total delay associated with a circuit. With constantly improving technology and more metal layers being available for interconnections, this contribution is predicted to increase. As the amount of interconnects among devices has a tendency to grow superlinearly with the number of transistors, the chip area is often limited by the area needed to accommodate the interconnects. Therefore, the interconnect dimensions are scaled as much as possible along with the device and voltage scaling. So with the width and the thickness of the metal stripes decreasing and the average length of the interconnects, using current approaches to layout synthesis, remaining the same or increasing slightly, the resistance of interconnects increases. For delay calculation it is also necessary to consider the

*Corresponding author. Tel.: 503 725 5415, Fax: 503 725 3807, e-mail: jeske@ee.pdx.edu
†This work was supported in part by the NSF grant MIP-9629419 and by the 1995 SIGDA/DAC Design Automation Award.

capacitance of interconnects. Luckily the capacitance per unit length usually does not change with scaling.

Therefore, it has become more and more important to be able to predict and control the length of interconnections. The obvious but quite challenging approach is to develop synthesis methods integrating logic and layout synthesis steps. Such integration could be accomplished by synthesizing functions with the objective being not a standard minimization of the number of gates or logic levels but generating a regular netlist with a defined interconnection structure. The regularity of the structure would permit the creation of a layout directly from the logic level description without any placement or routing, or alternatively a simplified placement and routing which would start from the "floor-plan" created with these regular structures.

The concept of a regular array to realize logic is an old one, but so far, only PLA-like structures have succeeded commercially. In 1972 Akers [1] proposed a universal, two-dimensional planar and regular array for arbitrary single-output functions. It was shown that any Boolean function can be mapped to such an array by the consecutive repetition of variables. But since the array size was calculated for worst case functions, such layout was very inefficient for real functions and the idea was not adopted. A multilevel, PLA-like array, called a Complex Maitra Logic Array (CMLA), was introduced in [8], but no efficient and effective algorithms to generate such representation have yet been developed.

In this paper we present a logic synthesis method for generating Pseudo-Symmetric Binary Decision Diagrams (PSBDDs), a new, regular two-dimensional function representation for both completely and incompletely specified multi-output Boolean functions. PSBDDs can be directly mapped to a two-dimensional layout without placement and routing. The idea originates from an OBDD [2] representation for totally-symmetric functions and generalizes the known switch realizations of symmetric binary functions [7]. The method is general and applicable to arbitrary multi-output incompletely specified Boolean functions, additionally the results can be improved when the original function is first decomposed to blocks realized as PSBDDs. We introduce synthesis methods for generating such regular representations and define the structure of an array to which PSBDDs can be mapped. *A Join-Vertex* operation, introduced in [9], is used to combine two geometrically-adjacent nodes such that the function is represented as a pseudo-symmetric network (two-dimensional regular array with only local connections between abutting cells) instead of a binary tree or DAG. To further optimize PSBDD sizes, we define a flipped Shannon expansion in respect to a geometrical realization of the diagram. We define an *array* as a geometrical concept. Thus, for instance, PSBDDs can be mapped to a Shannon Lattice Array which is a two-dimensional array of multiplexers, each with two inputs and one output, connecting with four neighbors of a node, and one input (control variable) from a diagonal bus. PSBDDs, which have a structure of contact networks for symmetric functions, can also be implemented using pass transistors with inverters and buffers added.

The remainder of this paper is structured as follows. Section 2 presents background on regular arrays. In Section 3 we describe Pseudo-Symmetric Binary Decision Diagram generation. The basic concept of function symmetrization is presented in Section 4. Newly developed heuristics for generating PSBDDs are described in Section 5. Comparison of PSBDD layouts to other function representations is given in Section 6. Section 7 presents our experimental results for the MCNC benchmark functions and Section 8 concludes the paper.

2. SYNTHESIS FOR REGULAR ARRAYS

The approach of Akers from 1972 was the only one in the literature that proposed an array which is similar to one of the possible implementations of

PSBDDs. The array is based on a rectangular grid and uses multiplexer cells which are associated with Shannon expansions. The arrays of Akers are unnecessarily large, because the order and repetition pattern of input variables is universal and calculated once for all functions by repeating consecutively the same variables. The number of times each variable has to be repeated, in Akers' approach, depends only on the variable's position in the variable order. No efficient procedures for finding the order of (repeated) variables are given, and it is easy to show simple functions that have very large arrays.

Our approach is quite different from that of Akers. We do not want to design a universal array for all functions, instead we developed a layout-driven logic synthesis method that gives efficient results for many real-life functions. We derived our ideas from the observation that a Binary Decision Diagram for a totally symmetric function is a regular two-dimensional structure. This is due to the fact that for totally symmetric functions every two geometrically-neighboring nodes on the same level of a planar drawing of a BDD are isomorphic and can be represented as one node. For non-symmetric functions the geometrically-neighboring nodes are usually non-isomorphic and, therefore, the BDD structure is an (irregular) Directed Acyclic Graph (DAG) not suitable for mapping to a regular array. In our approach we first create a tree expansion, but after expanding each level, we combine together non-isomorphic geometrically-neighboring nodes on that level using the *Join-Vertex* operation, thus creating a layered regular Directed Acyclic Graph. During the *Join-Vertex* operation, variables are reintroduced back into the function, therefore, combining nodes leads in turn to the requirement of variable repetition. The regular diagrams created with the *Join-Vertex* operation are called Pseudo-Symmetric Binary Decision Diagrams (PSBDDs). In contrast to Akers approach, the variables in PSBDDs can be repeated in any order and the number of repetitions of a variable depends on a particular function rather then being precalculated universally. The concepts of node joining and non-consecutive variable repetition add much power to the diagrams. We propose a number of heuristics for variable ordering, and show that with a good order a substantial minimization of diagram sizes can be achieved. We use Shannon (S), and flipped Shannon (fS) expansions. Flipped Shannon expansion, very important for PSBDDs where the actual positions of nodes on the plane are considered, can be viewed as Shannon expansion performed for the inverted control variable. It obviously does not apply for BDDs where relations between nodes are not defined geometrically. In addition, we take advantage of "constant pseudo-expansions", which means we assume constant values in control variables. In our approach, which is feasible for multi-output, incompletely specified functions, arbitrary non-symmetric functions are symmetricized by repeating variables in order to realize them as regular arrays.

One possible way to implement PSBDDs is with an array of multiplexers, and we can represent it as a geometric concept in the following way. First let us define a general two-dimensional structure which we call Lattice Array.

Lattice Array, shown in Figure 1, is the data structure that describes the regular geometry of a circuit. Each *non-zero entry* $L[i,j]$ in array L is

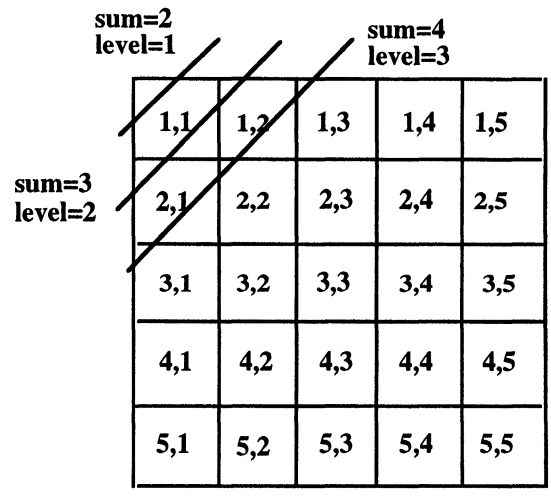


FIGURE 1 Concept of a lattice array.

called a node and includes a link to a data structure that describes the logic placed in this entry, in the simplest case it can be a logic constant. $L[1,1]$ is the *root node* of the lattice. Boolean functions in the links can be represented as BDDs or arrays of cubes. Inputs to each cell (i,j) come from their neighbors $(i,j+1)$ and $(i+1,\ j)$ or are set to Boolean constants, 0 or 1.

A Lattice Array for a single-output function is represented by a matrix L.

DEFINITION 1 *A diagonal* of the matrix L is a set of entries that have the same sum of indices. The sum in the first diagonal is 2, in the second diagonal is 3, and so on. A diagonal corresponds to an expansion level in the Decision Diagram.

DEFINITION 2 *An Ordered Lattice Array* is a lattice array in which there is one variable on a diagonal.

DEFINITION 3 *An Ordered Lattice Array with repeated variables* is a lattice array in which the same variable may appear on various diagonals, but there is only one variable on a diagonal.

DEFINITION 4 *A Free Lattice Array* is a lattice array in which there is more than one variable on a diagonal.

In this paper we consider a two-dimensional array of multiplexers each with two data inputs and one output, which can be connected to two other neighbors of a node, as well as one input (control variable) coming from a diagonal bus. Both polarities of variables are allowed for implementing Shannon and Flipped Shannon expansions. We say a variable has a positive polarity when it is present in a function, and has a negative polarity when a negation of the variable is present. Pseudo-expansions corresponding to constant values 0 and 1 of control variables are also included.

DEFINITION 5 An *Ordered Shannon Lattice Array is* an ordered lattice array in which all cells are multiplexers.

Thus in an Ordered Shannon Lattice Array, on a diagonal, all cells are of type S, 0 or 1, or all cells are of type fS, 0 or 1. It can be proven that every binary function can be realized with such a structure, but in the worst-case an exponential number of levels is necessary (which means the control variables in diagonal buses will be repeated very many times). In the Akers's array the same variable is subsequently repeated without other variables interspersed and such arrays are called *variable interval arrays*. In our approach, however, we assume no constraints on variable order. In many cases this allows a dramatic decrease in the number of variable repetitions and thus in the size and delay of the design. We will focus our attention on the neighbor-to-neighbor connections, which in the case of four neighbors and four I/O connections constitute planar routing resources. The shape of implemented PSBDDs is approximately triangular or trapezoidal in various sizes.

3. GENERATING PSBDDs FOR COMPLETELY SPECIFIED FUNCTIONS

The idea of our approach originates from symmetric networks and BDDs. Let us recall two definitions of the symmetries which will be used here to explain our approach.

DEFINITION 6 A function f exhibits **a non-equivalent-symmetry (NE-symmetry)** [6] in variables a and b, denoted as $aNEb$ or $\{a,b\}$ $(\{a',b'\})$, if $f_{ab'} = f_{a'b}$.

DEFINITION 7 A function f exhibits **an equivalent-symmetry (E-symmetry)** [6] in variables a and b, denoted as aEb or $\{a,b'\}(\{a',b\})$, if $f_{a'b'} = f_{ab}$.

If two variables exhibit non-equivalent symmetry, the function's two cofactors, $f_{ab'}$ and $f_{a'b}$, (in respect to these two variables) are equal and can be represented as one node in a planar drawing of the function's OBDD, as shown in Figure 2a. We assume that OBDDs are always drawn such that the true cofactor is drawn as the right child and the negative cofactor is drawn as the left child.

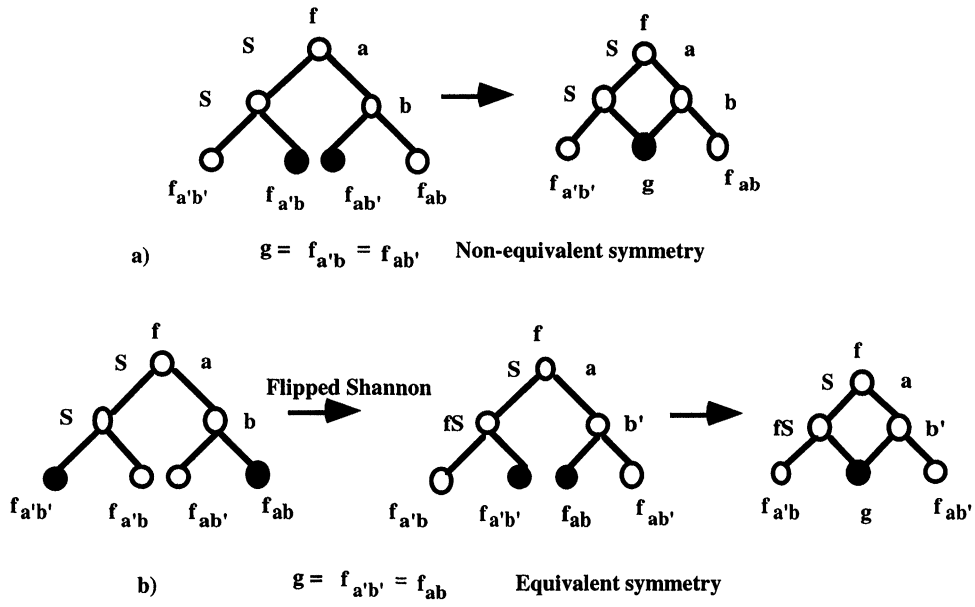
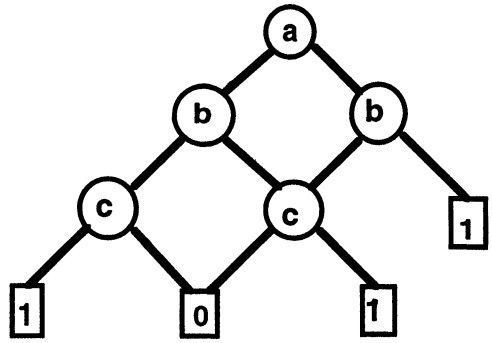


FIGURE 2 Use of isomorphic cofactors: (a) Shannon expansion, (b) Flipped Shannon expansion.

Therefore, in order to take advantage of the equivalent symmetry we introduce a Flipped Shannon expansion which is shown in Figure 2b, where black nodes represent the equivalent nodes. If a Boolean function is totally symmetric with only non-equivalent symmetries holding between its variables, then its OBDD can be drawn as shown in Figure 3. The OBDD there has a desired structure, regular and with only neighbor-to-neighbor connections. Decomposition variables are assigned to diagonal busses. Such structures can be directly mapped to Ordered Shannon Lattice Arrays.



$f = a'b'c' + a'bc + ab'c + ab$

FIGURE 3 An OBDD for a totally symmetric function.

Unfortunately, not all functions are totally symmetric. The regular OBDD for a totally symmetric function is a result of merging together isomorphic geometrically-adjacent nodes. This merging idea is extended here by us for the case of non-isomorphic nodes through the *Join-Vertex* [9] operation presented below. The underlying idea of the *Join-Vertex* operation using the BDD and the PSBDD representations is shown in Figure 4. The penalty to be paid is the reintroduction of the expansion variables back into the function. The reintroduction of the variable makes it necessary to use the same expansion variable more than once which increases the number of levels.

Function f can be expressed using four cofactors of any two of its input variables as shown in Eq. (1). By grouping together terms with cofactors $f_{a'b}$ and $f_{ab'}$, function f can be represented as in Eq. (2).

$$f = a'b'f_{a'b'} + a'bf_{a'b} + ab'f_{ab'} + abf_{ab} \quad (1)$$

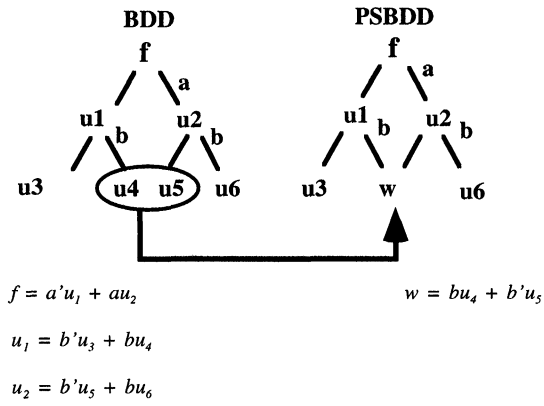


FIGURE 4 Generating a PSBDD using a *Join-Vertex* operation on nodes $u4$ and $u5$.

$$f = a'b'f_{a'b'} + (a'b + ab')g + abf_{ab} \quad (2)$$

For the Eq. (2) to be satisfied, function g has to be equal to $f_{a'b}$ for $a = 0$ and $b = 1$, and to $f_{ab'}$ for $a = 1$ and $b = 0$. So, if the two cofactors are not equal, to satisfy the Eq. (2), we use the *Join-Vertex* operation. The *Join-Vertex* operation given in Eq. (3) is defined for variable order $\{a, b\}$.

$$g = bf_{a'b} + b'f_{ab'} \quad (3)$$

We can verify the correctness of the *Join-Vertex* operation by substituting it back to Eq. (2).

$$f = a'b'f_{a'b'} + (a'b + ab')(bf_{a'b} + b'f_{ab'}) + abf_{ab} \quad (4)$$

$$f = a'b'f_{a'b'} + a'bbf_{a'b} + ab'bf_{a'b} + a'bb'f_{ab'} + ab'b'f_{ab'} + abf_{ab} \quad (5)$$

Two elements of the Eq. (5), $ab'bf_{a'b}$ and $a'bb'f_{ab'}$, are equal to 0 because the law of Boolean algebra states that $b'b = 0$. The resulting function is given in Eq. (6).

$$f = a'b'f_{a'b'} + a'bf_{a'b} + ab'f_{ab'} + abf_{ab} \quad (6)$$

As can be seen, Eqs. (1) and (6) are the same, therefore, with the *Join-Vertex* operation the function remains unchanged. The law $b'b = 0$ of Boolean algebra or similar properties in non-binary algebras must hold in any algebraic system on which more general lattices would be based. The concept of the *Join-Vertex* operation is very powerful and general in logic design as it applies to combining any two nodes which are not isomorphic (not necessarily combining neighbors), and therefore can find many other applications in decision diagrams and function representations.

4. THE BASIC CONCEPT OF FUNCTION SYMMETRIZATION

Applying the *Join-Vertex* operation to all geometrically-adjacent nodes on a level has the same effect as introducing a repeated variable in the process of function symmetrization using variable repetitions. The symmeterization of a function can be viewed in terms of introducing *don't cares* to the function. Let us discuss this in more detail.

A repetition of a single variable introduces *don't cares* into the half of the K_{map} minterms of a new function with the repeated variable, so the more variables are repeated, the more weakly specified the function becomes. If one starts from a completely specified function and repeats two variables, 75% of the minterms of the new function will be *don't cares* as shown in Figure 5.

DEFINITION 8 A *symmetry index* S^i of a totally symmetric function $F(x_1, x_2, \ldots, x_r, \ldots, x_n)$ is $S^i = 1$ when $F(x_1, x_2, \ldots, x_n) = 1$ for every i of its

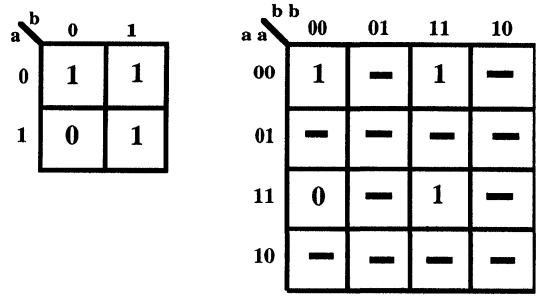


FIGURE 5 Introducing *don't cares* through the variable repetition.

argument variables x_r set to 1 and other variables set to 0 (see Fig. 6). The *layer of index* S^i, in an arbitrary function, is the set of cells of K_{map} that have exactly i of its argument variables x_r equal to 1. For an incomplete totally symmetric function $F(x_1, x_2, \ldots, x_r, \ldots, x_n)$, for every S^i, $i = 1, \ldots, n$, all cells are either all 1's and DC's or all 0's and DC's [7].

DEFINITION 9 An incomplete function is totally *symmetric* iff in the layer of every symmetry index all *minterms* are either 1's (and DC's), or all *minterms* are 0's (and DC's).

DEFINITION 10 A layer of symmetry index is called *consistent* if all *minterms* are either 1's and DC's, or all *minterms* are 0's and DC's.

It can be easily proved that Definition 9 is a generalization of Definition 8 for the case of the incompletely specified functions.

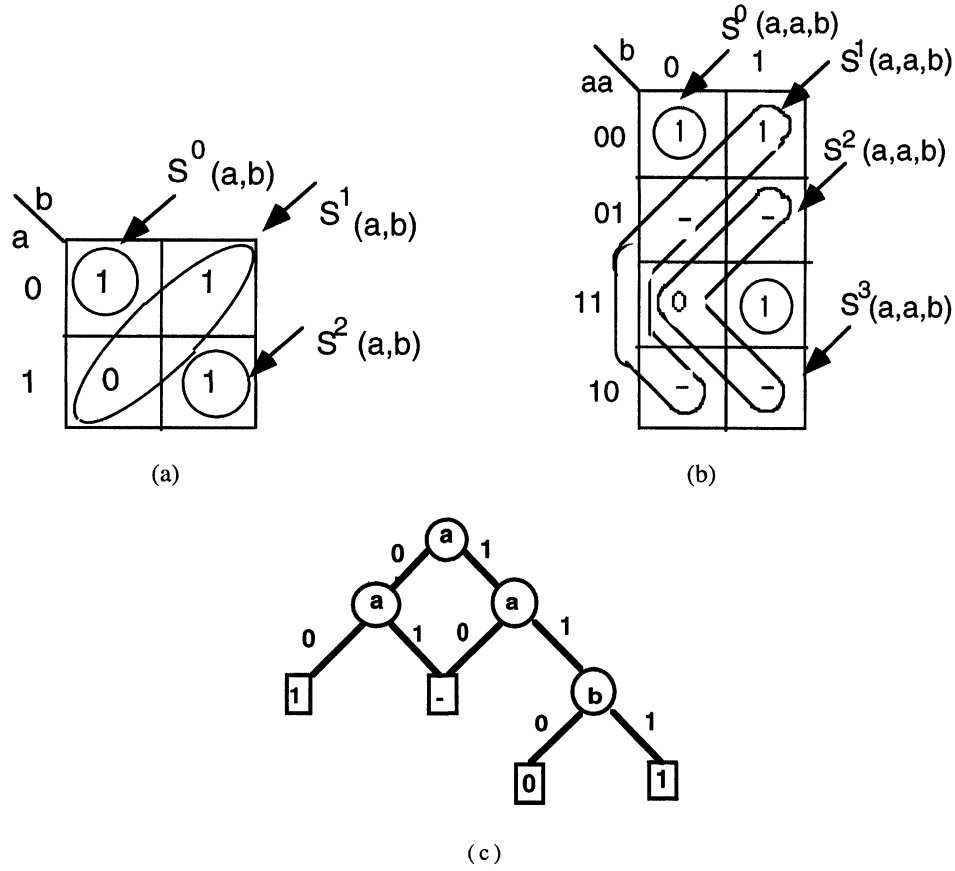


FIGURE 6 Symmetrization of the function using a variable repetition. (a) original function, (b) symmetricized function, (c) PSBDD.

THEOREM 1 *Every Boolean function can be made totally symmetric (symmetricized) by repeating some of its variables* [15].

The above theorem was presented for the first time by Arnold and Harrison [15] for a total of 2^n variables for the n-variable function. These results were further improved by others and finally Lee and Hong [16] presented an iterative algorithm which optimizes the number of repeated variables based on partial symmetries. Below we present an illustrative reasoning of the symmetrization process.

If for every value of the symmetry index the corresponding layer is consistent – the function can be converted to a completely specified totally symmetric function by replacing all don't care cells in a layer of at least one "1" with 1-cells, *don't care* cells in a layer of at least one "0" with 0-cells, and a layer with all "–" with either all 0-cells or all 1-cells.

We say that variable x separates zero-cell $z1$ from one-cell $o1$ in a K_{map} when $z1 \subseteq x$ and $o1 \subseteq x'$ or $z1 \subseteq x'$ and $o1 \subseteq x$. To simplify the discussion and the example given in Figure 6, we only consider non-equivalent symmetry. If for a given function there exists a layer with both ones and zeros (such as layer $S^1(a, b)$ in Fig. 6a) by repeating a variable that separates these ones and zeros a new K_{map} is created as shown in Figure 6b. In Figure 6a, both variables a and b are the separating variables for cell $z1 = a'b$ and cell $o1 = a'b$ so any of two can be used. In the new K_{map} (Fig. 6b) the 0-cell and the 1-cell that were in the same layer in the previous map become now partitioned to two different layers. Because adding one variable partitions the set of cells to two sets, after a finite number of partitions there will be a single "1" or a single "0" plus *don't cares* in each layer (in the worst case). Thus the process will always terminate (usually it terminates earlier without a need to have a single *care* cell in every layer).

In Figure 6b, just by repeating variable "a" all the layers become consistent. Figure 6c corresponds to a PSBDD for the function shown in Figure 6a and represents the totally symmetric function $S^{0,1,3}(a, a, b)$.

DEFINITION 11 *Symmetrization* is the process of converting an arbitrary function to a totally symmetric function by repeating some of its variables. It creates a new incomplete function.

DEFINITION 12 Function F is *lattice-realizable* in a lattice of type T_i when its diagram can be mapped (monomorphism) to the lattice of type T_i without variable repetitions.

THEOREM 2 *Every totally symmetric Boolean function is lattice-realizable in Ordered Shannon Lattice Array. Every Boolean function can be implemented in Ordered Shannon Lattice possibly with variable repetitions.*

Observe that many functions characterized as non-symmetric are still lattice realizable.

A function can be totally symmetric, partially symmetric, pseudo-symmetric or non-symmetric. The type of the function can be found from the analysis of cofactors and their negations [5, 6]. In the case of a PSBDD with two types of expansions, Shannon and flipped Shannon, we have two degrees of freedom for the algorithm; selection of S or fS for the level; and the order of variables (this takes into account repetitions, too). The selection of a good order of variables is based on generalized partial symmetries for cofactors.

DEFINITION 13 The generalized partial symmetries for cofactors are the following properties of cofactors and relations between pairs of (in general, multi-variable) cofactors.

(a) a single cofactor is an incomplete tautology with 1 ($fi = 1$).
(b) a single cofactor is an incomplete tautology with 0 ($fi = 0$).
(c) incomplete tautology of any two cofactors ($fi = fj$).
(d) incomplete tautology of a cofactor with a negation of any another cofactor ($fi = fj'$).

Note that cofactors are calculated while creating a PSBDD and these symmetries can be applied to *any* cofactors of a (sub) function.

If the function is symmetric and complete, it is represented as a diagram with an arbitrary order of variables without any repetitions. If a function is partially symmetric or non-symmetric, a PSBDD is generated level-by-level, using different variable ordering heuristics until the entire function is mapped to a diagram with some variables repeated only if necessary. In the process of generating a diagram we need to try not only to decrease the number of variable repetitions, but also to decrease the total area occupied by the diagram.

4.1. Multi-output Incompletely Specified Functions

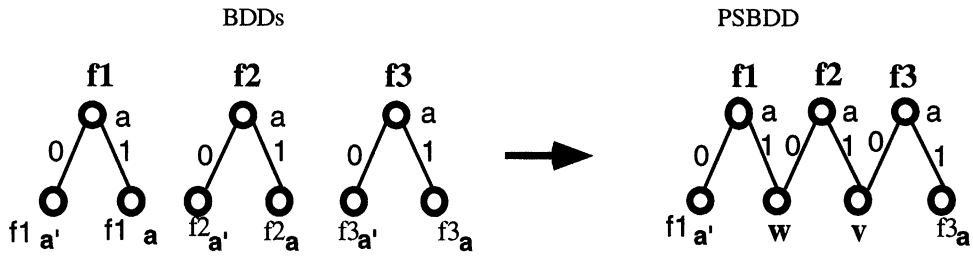
The method to create PSBDDs can be easily extended to multi-output incompletely specified functions as illustrated in an example. First, in Figure 7 we give the overview of how the *Join-Vertex* operation can be used without modification for any multi-output function. A step-by-step illustration of the PSBDD generation is explained in Example 1 and shown in Figure 8. The initial distances between root functions f_1, f_2, and f_3 as well as their orders can be arbitrary. The distance between two functions can be understand as the number of variables which are expanded in individual functions before the geometrically-adjacent nodes of two functions are joined together with the *Join-Vertex* operation. In Figure 7 the distances between all functions are equal to one (only variable a is expanded in the individual functions; all other variables are expanded after the functions are joined together). The distances and orders strongly affect the size of the solution layout and the delay value.

Example 1 In Figure 8 functions f_1, f_2, f_3 are represented by three K_{maps}. As shown in the figure, calculating the positive cofactor f_a means replacing the half of K_{map} corresponding to a' with don't cares. Similarly, calculating the negative cofactor $f_{a'}$ means replacing the half of K_{map} corresponding to a with don't cares. In Figure 8a, the PSBDD generation process using K_{map} is given and the Shared PSBDD is shown in Figure 8b. The *Join-Vertex* operation for the node is just the set-theoretical union of all care sets in its parent K_{maps}.

The above example shows constructively that for every function we can design a PSBDD with repeated variables, thus every function can be symmetricized.

5. VARIABLE ORDERING HEURISTICS

The order of variables and the type of operation (S or fS) influences very strongly the size and shape of a PSBDD. As these diagrams are related to OBDDs, our first approach was to examine variable ordering methods used for BDDs. A number of successful variable ordering heuristics for OBDDs [13] are based on changing the positions of variables in the variable order and recalculating the sizes of the OBDD. Exchanging the



$$w = af_{1a} + a'f_{2a'} \qquad v = af_{2a} + a'f_{3a'}$$

FIGURE 7 PSBDD for a multi-output function.

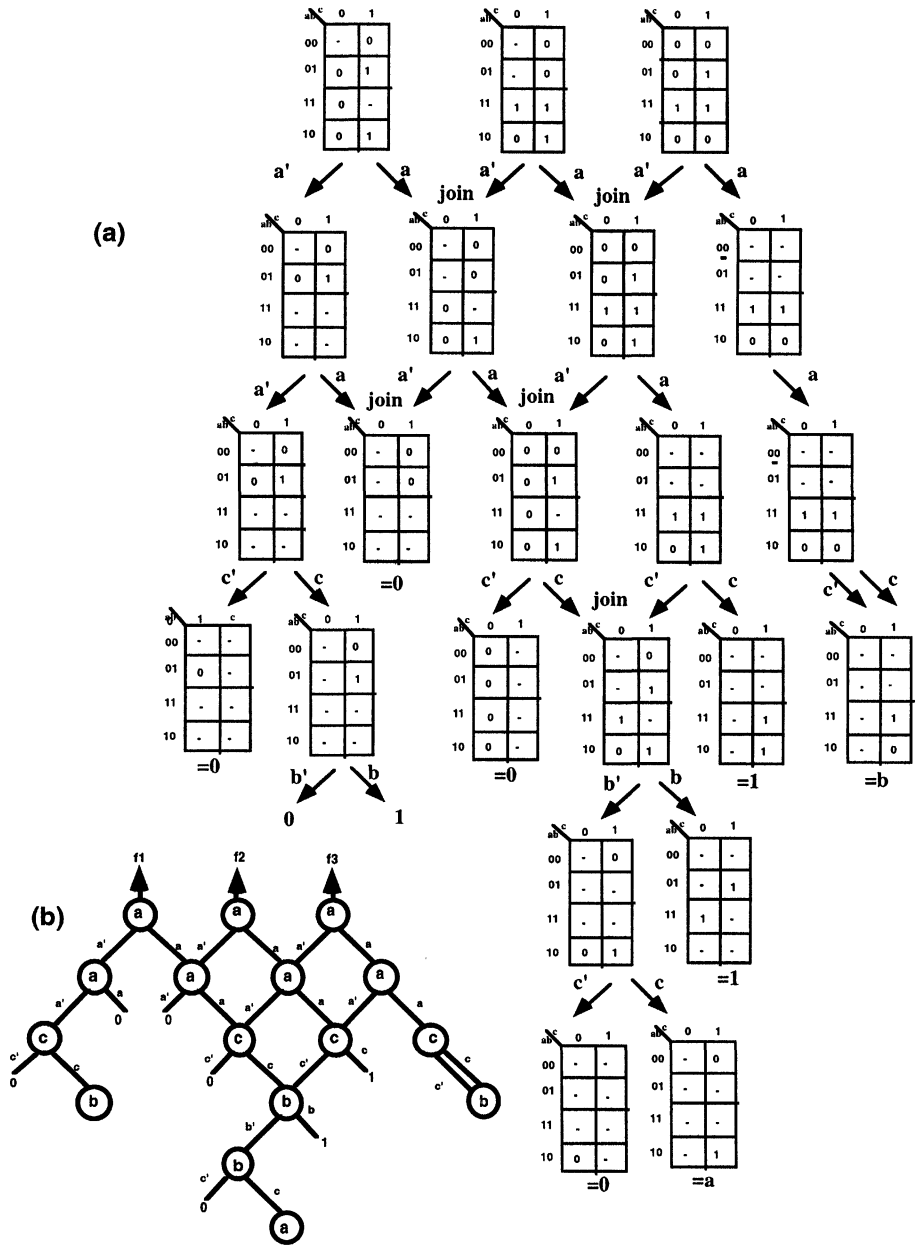


FIGURE 8 Creation of the Multi-output Shared PSBDD for the incompletely specified functions. (a) demonstration of the PSBDD generation using *K*-maps; (b) the complete Shared PSBDD.

positions of two consecutive variables requires the recalculation of the OBDD's vertices for only one level, while in PSBDDs such operation would require the recalculation of vertices in all levels following the levels of the exchanged variables. Therefore, these methods cannot be adopted for PSBDDs. The variable order has to be determined before or during PSBDD generation. We use function characteristics to determine the order of variables.

In this section we first focus on developing heuristics to generate PSBDDs without using any symmetry information. As it will be shown, such heuristics can be developed independently and then combined with symmetry information to improve results. We compare the orders of variables generated by our heuristic algorithms with the symmetry properties of the tested functions to evaluate our approaches. In most tested benchmark functions the symmetric variables are placed together at the beginning of the variable orders, which is the best strategy for minimizing the sizes of PSBDDs. Some characteristics of the function variables which are used in our methods are given below.

DEFINITION 14 A *variable appearance* is the number of cubes in an optimized two-level function representation in which a variable is present.

DEFINITION 15 A *positive appearance* is the number of cubes in an optimized two-level function representation in which a variable is present in positive polarity.

DEFINITION 16 A *negative appearance* is the number of cubes in an optimized two-level function representation in which a variable is present in negative polarity.

Initially we created an algorithm called "Fixed Order Method" in which the order of expansion variables was determined at the beginning of the PSBDD generation process based on variable appearances in the root function. For incompletely specified functions, the variable appearance is calculated using only "ON" (true) terms. All original function variables were used as expansion variables in the determined order, and the set of corresponding decomposition levels is called the first loop. Next, the variables which were reintroduced to the function by the *Join-Vertex* operations were used again as expansion variables in the same order as in the first loop and created the second loop. This procedure continues until a function is completely decomposed.

In Figure 9, three PSBDDs for the non-symmetric function f for variable orders $\{c, a, d, b\}$, $\{b, c, a, d\}$ and $\{c, b, a, d\}$ are presented. Variable loops are indicated. Please notice that the variable order in all loops is the same, however, some variables may be missing in higher loops. No *Join-Vertex* operation was necessary for the diagram in Figure 9c, therefore no repeated variables appear in the expansion. The influence of the variable order on the size of PSBDDs can be observed. The number of the expansion nodes is nine for the *cadb* order and *cbad* order, and eighteen for the *bcad* order. Vertices which are marked with "$*$" were created using the *Join-Vertex* operation. Please notice that in Figure 9b variables a and d appear twice in the path and variable c appears three times. A number of PSBDDs for different functions from the MCNC benchmark set were generated using Fixed Order algorithm, but their sizes were quite large.

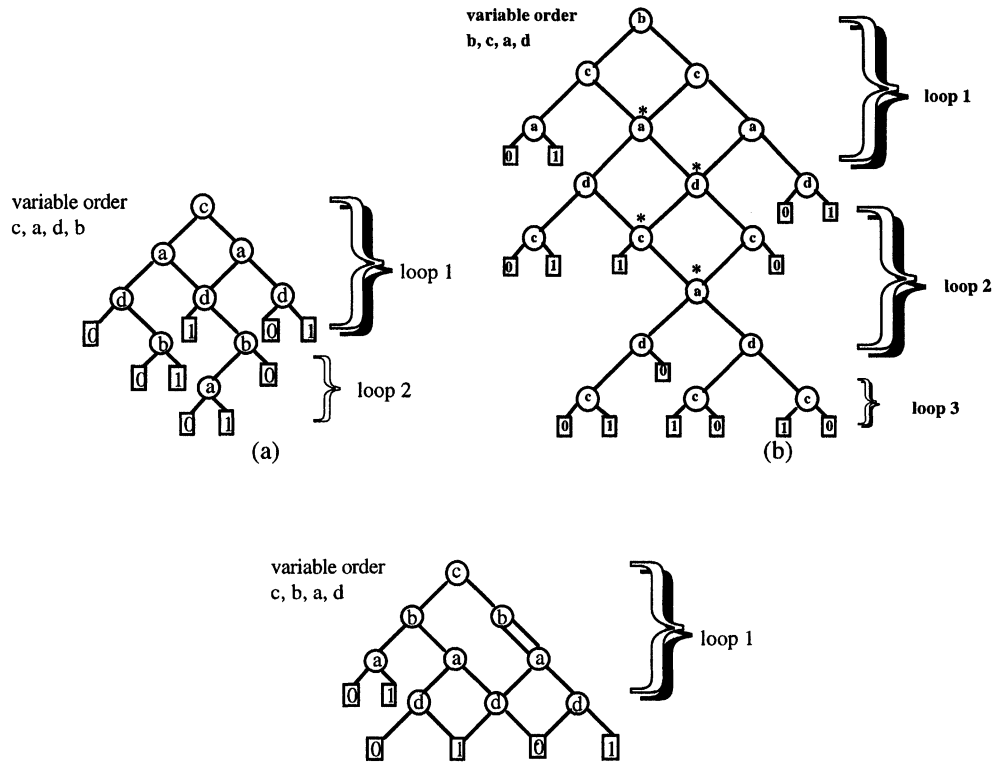
Our first improvement was to allow different orders of variables in different loops. Variable orders for each loop were defined based on variable appearances in the function representation at the beginning of each loop. The justification behind this heuristic is that a variable with higher appearance reduces a larger number of cubes/terms when cofactors are calculated. Such an algorithm was implemented and we noticed small improvements for some benchmark functions.

In our next approach we decided to relax loop restrictions and all variables appearing in the function were used in the selection of the next variable. Two groups of methods were developed; Greedy and Look-ahead.

5.1. Greedy Methods

We dynamically choose a variable for the next level expansion by recalculating appearances of all variables at each decomposition level. The variable with the maximum appearance number is chosen. Two ways of breaking ties were implemented.

G1 - choose a variable with the minimum difference between positive and negative appearances.



$$f = a'cd' + ac'd' + acd + ab'd' + a'bc'd$$

FIGURE 9 Influence of different variable orders on the number of nodes, loops and variable repetitions.

G2 - choose a variable with the maximum difference between positive and negative appearances.

5.2. Look-ahead Methods

To choose the expansion variable for the next level, the expansion to the next level is performed for all variables and both expansions, Shannon and Flipped Shannon. We use only one type of expansion per level. All adjacent isomorphic nodes are detected and combined together, and the number of nodes and literals for each expansion variable is calculated and used in the next-level variable selection. Comparing to the Greedy-methods the complexity of Look-ahead methods increases approximately $2n$ times.

L1. a variable which generates the minimum number of nodes in the next level is selected. In case of a tie, the variable which has the minimum appearance is selected.

L2. a variable with the minimum appearance number is selected. In case of a tie, the variable which generates the maximum number of nodes in the next level is selected.

L3. a variable with the minimum appearance number is selected. In case of a tie, the variable which generates the minimum number of nodes in the next level is selected.

The diagram presenting all methods is shown in Figure 10. There is a common-sense justification behind each of these heuristics. For the given order of variables and every variable using one of two

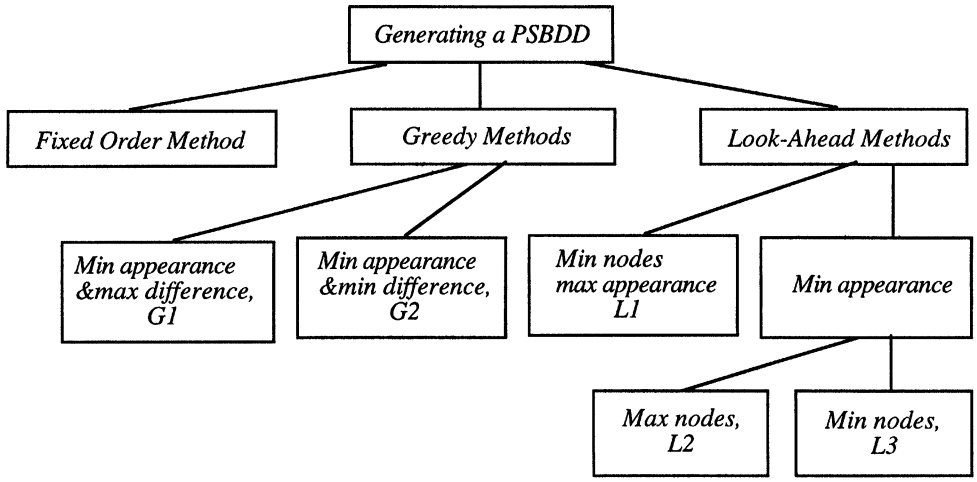


FIGURE 10 Methods for PSBDD generation.

expansions, Shannon or flipped Shannon (only one expansion per level), there exist 2^n different PSBDDs and our heuristics serve to select the best of them. With our heuristics, the symmetric variables are usually selected at the top of the set of ordered variables.

6. LAYOUT COMPARISON

One possible implementation of PSBDDs is with Sea-of Muxes as shown in Figure 11, although the Sea-of-Muxes can be substituted with Sea-of-Gates, or with a standard cell library composed of cells built with standard gates or pass transistors. Pass transistor networks as a direct choice for PSBDD realization are currently under investigation. Additional buffers for large fanout variables and large pass transistor networks should be considered where necessary. As realizations of PSBDDs can have different shapes (usually trapezoid or diamond), decompositional preprocessing and additional floorplanning is required for larger designs. Delays are proportional to the number of levels and can be accurately predicted.

To determine all advantages of this new layout-driven logic synthesis method, a detailed comparison between the final layouts of PSBDDs and the layouts of other multilevel representations should be performed. Such comparisons are currently

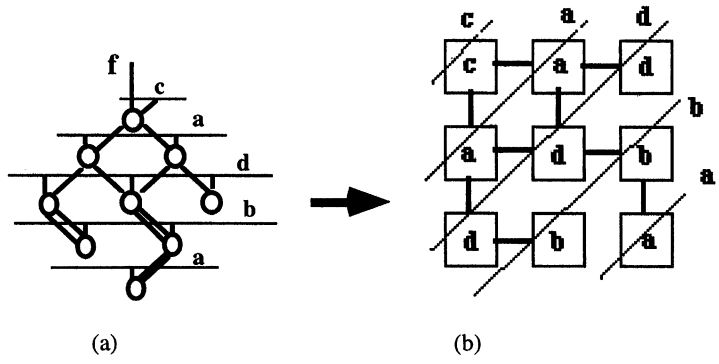
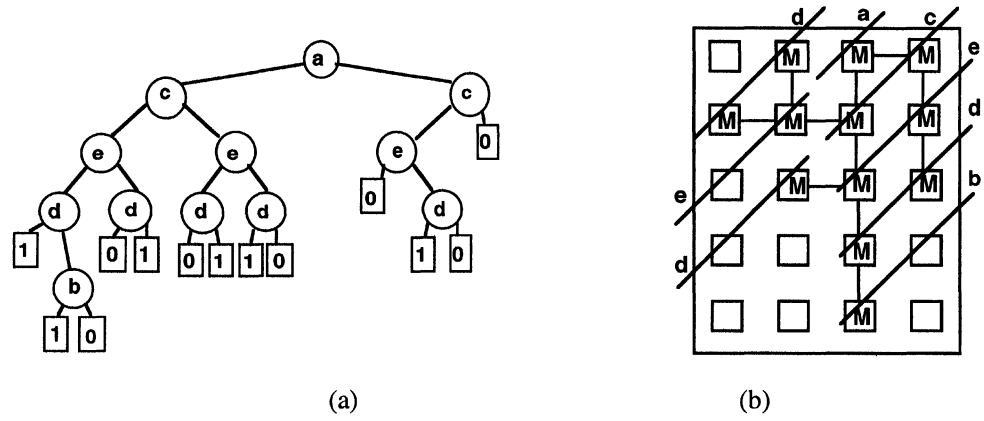


FIGURE 11 (a) an initial PSBDD, (b) its mapping to a Sea-of-Muxes.

being pursued. In this paper, we present a simplified analysis and comparison between two representations, a PSBDD and a BDD, and between their mappings to a two-dimensional array, built with neighbor-to-neighbor connected multiplexer cells and diagonal busses. For illustration we use the function given in Eq. (7).

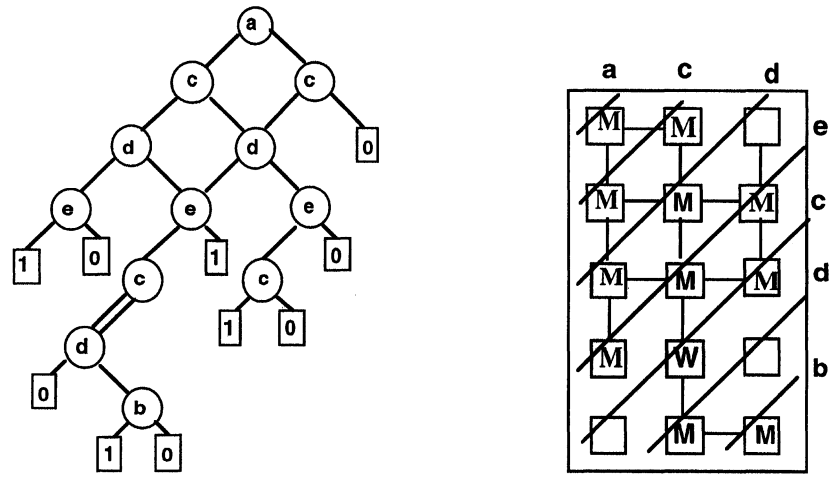
$$f = a'b'c'e' + ac'd'e + a'cde' + a'cd'e + a'c'd'e' + a'c'de \quad (7)$$

The diagram and its implementation, for the function from Eq. (7), realized as a BDD and as a PSBDD are given in Figures 12 and 13, respectively. It can be observed that the overall BDD layout is larger than the PSBDD layout and not as regular. In addition, the BDD representation requires more diagonal buses (8 *versus* 7 for the PSBDD) and has two variables assigned to the same bus for two diagonals (e with a, and d with c), which results in additional connections. Obviously, creating a layout using a BDD representation is more complicated than using a PSBDD representation. Based on the above example we expect that there exist functions for which the



(a) (b)

FIGURE 12 (a) OBDD for the function from Eq. (7), (b) its two-dimensional layout. M-multiplexer cell.



(a) (b)

FIGURE 13 (a) PSBDD for the function from Eq. (7), (b) its two-dimensional layout. M-multiplexer cell. W-wire.

PSBDD layout is more compact and delay is shorter than the BDD realization, however more detailed comparison is needed to determine the quantitative relations. The comparison between the number of nodes and levels in BDDs and PSBDDs for the MCNC benchmark functions are given in Table II.

PSBDDs are regular structures, therefore, it seems natural to compare them with PLAs, the regular two-level representation which is still used in design implementations. For totally symmetric functions, it is easy to find examples when PSBDD layout is better than PLA layout, because there are no repeated variables. For instance assume the function given below.

$$S^{2,3}(a,b,c,d) = abc' + ac'd + bc'd + a'cd + b'cd + bcd' + acd' \quad (8)$$

The diagram and its implementation, for the function from Eq. (8), realized as a PLA and as a PSBDD, are given in Figures 14 and 15, respectively. When realized as a PLA (Fig. 14a) it has 14 AND gates and 6 OR gates, therefore, the area cost of its realization can be estimated as 20 gates. Such estimations are favorable for the PLA as in reality if not folded, it will occupy an area equal to the number of variables and their complements multiplied by the number of terms (for this example $8 \cdot 5 = 40$ units). The delay cost of this implementation is equal to 8 gate delays plus 7

TABLE I Comparison between Fixed-order, Greedy and Look-ahead methods

Benchmark functions				Fixed-order method (only S)			Greedy methods (only S)				Look-ahead methods (S and fS)					
							G1		G2		L1		L2		L3	
Name	# in	out #	SOP	# lo	# le	# no	# le	# no	# le	# no	# le	# no	# le	# no	# le	# no
apex7	17	29	28	2	21	124	–	–	–	–	–	–	25	150	25	148
c8	10	13	47	–	–	–	19	118	–	–	–	–	10	29	10	29
clip	9	1	44	–	–	–	–	–	–	–	–	–	18	103	20	108
clip	9	2	45	–	–	–	–	–	–	–	–	–	27	223	27	220
cm162a	10	2	12	3	18	48	12	35	12	35	11	19	11	29	11	24
count	20	15	26	2	37	226	22	53	20	53	20	37	20	55	20	55
cps	22	0	15	–	–	–	–	–	–	–	113	657	26	134	27	136
cps	18	1	27	–	–	–	–	–	–	–	–	–	26	164	28	158
cps	22	2	30	–	–	–	–	–	–	–	–	–	39	342	41	338
cps	18	4	16	–	–	–	–	–	–	–	62	325	24	71	25	68
duke2	17	2	10	–	–	–	–	–	–	–	–	–	18	54	18	52
duke2	17	13	13	–	–	–	–	–	–	–	–	–	21	105	20	90
duke2	18	17	15	–	–	–	–	–	–	–	–	–	22	92	23	86
duke2	15	18	7	–	–	–	–	–	28	107	28	108	15	48	16	53
duke2	18	19	6	–	–	–	–	–	–	–	–	–	18	58	18	48
duke2	16	21	10	–	–	–	–	–	–	–	–	–	19	76	20	81
duke2	17	6	7	–	–	–	–	–	–	–	38	119	17	63	18	47
duke2	18	7	15	–	–	–	–	–	–	–	–	–	21	109	25	134
example2	16	22	12	–	–	–	–	–	–	–	58	256	17	52	19	57
example2	14	58	6	1	14	30	14	30	14	30	–	–	15	31	15	31
example2	13	62	11	–	–	–	–	–	–	–	44	203	14	37	15	34
frg2	20	98	20	–	–	–	–	–	–	–	66	337	22	189	32	228
frg2	19	99	39	–	–	–	–	–	–	–	–	–	28	164	32	177
k2	12	0	6	6	37	144	15	34	16	57	25	94	12	27	12	20
sao2	10	1	20	1	10	45	–	–	–	–	–	–	18	75	18	71
sao2	10	2	22	1	10	55	–	–	–	–	–	–	17	87	16	73
sao2	10	3	26	1	10	47	–	–	31	171	–	–	16	68	17	70
ttt2	14	15	13	–	–	–	15	43	15	43	133	1001	14	31	14	31
vg2	14	1	10	–	–	–	–	–	–	–	28	74	14	30	14	30
vg2	25	1	10	–	–	–	40	421	30	187	–	–	25	91	25	91
vg2	10	3	10	–	–	–	–	–	38	342	–	–	18	77	18	77

TABLE II Comparison between PSBDD, BDD and PLA representations

Benchmark functions				Comparison		Look-ahead methods (S and fS)						Comparison			
				ROBDD [14]	OBDD [5]	L1		L2		L3		PLA		PSBDD	
Name	# in	out #	SOP	# no	# no	# le	# no	# le	# no	# le	# no	delay	area	delay	area
apex7	17	29	28	62	141	–	–	25	150	25	118	52	980	50	578
b9	11	0	6	25	49	–	–	11	29	11	28	28	138	22	242
c8 × 13	10	13	47	22	85	–	–	10	29	11	28	57	987	22	200
cm 162a	10	2	12	19	30	11	29	11	29	11	24	32	252	22	242
duke2	8	23	4	16	23	11	16	9	20	9	16	20	68	18	162
example2	16	22	12	32	50	58	256	17	52	19	57	44	396	48	722
example2	14	58	6	27	31	–	–	15	31	15	31	29	319	28	392
example2	13	62	11	24	24	44	203	14	37	15	34	37	297	30	450
k2	12	0	6	20	30	25	94	12	27	12	20	30	150	24	288
Average	13	–	15	38	55	–	–	16	72	17	72	37	399	29	364

$f = abc' + ac'd + bc'd + a'cd + b'cd + bcd' + acd'$

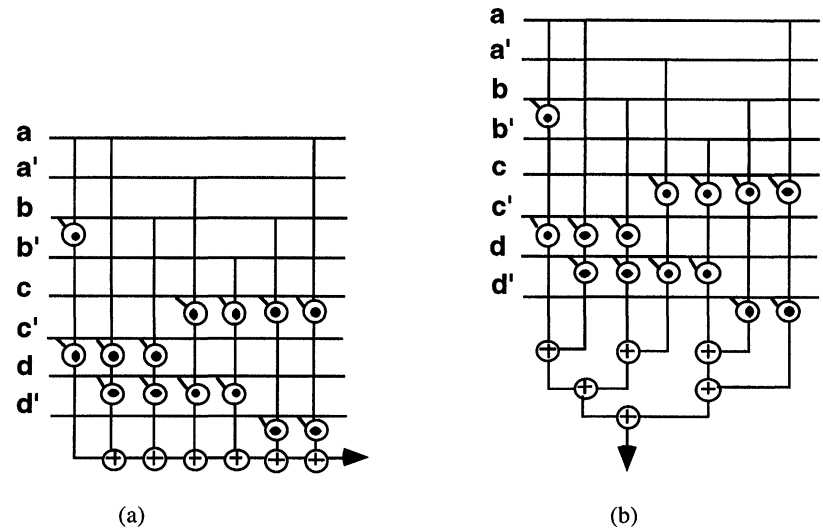
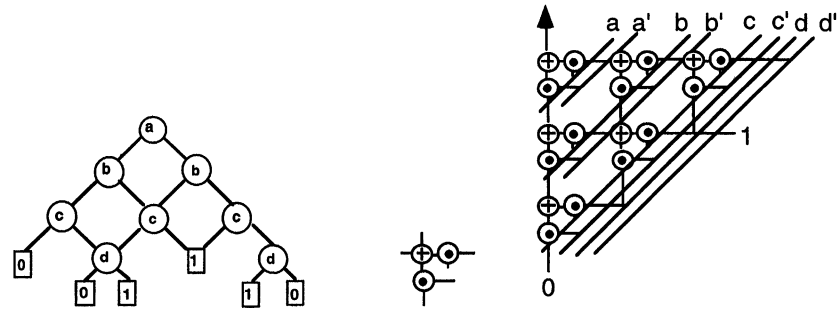


FIGURE 14 PLA realization of the function from Eq. (8). (a) AND-OR planes, (b) AND plane-OR tree.

wire delays. So, the total cost of this realization is 20 gates and 8 delays, not counting wire unit delays. When the OR cascade plane is replaced with the OR tree (Fig. 14b) it requires 6 OR gates. The OR tree introduces 3 gate delays and 2 wire delays. Therefore, the cost of AND-OR tree realization is 20 gates and 7 delays for the complete design.

For comparison we implement the same function, from Eq. (8), using the PSBDD shown in Figure 15a. It requires 6 mux cells and 3 gate delays when implemented in the Ordered Shannon Lattice Array, as shown in Figure 15c. The multiplexer array is built with abutted multiplexer cells, and a unit multiplexer cell is shown in Figure 15b. Assuming the cost of a mux cell to be equal to that of two AND gates and one OR gate and the delay of that cell to be equal to two gate delays, the total cost of the PSBDD realization is 18 gates and 6 gate delays without wire delays. Therefore, we

$f = abc' + ac'd + bc'd + a'cd + b'cd + bcd' + acd'$



(a) (b) (c)

FIGURE 15 PSBDD realization of the symmetric function from Eq. (8). (a) PSBDD, (b) multiplexer cell, (c) arrays of abutted multiplexer cells.

can observe, that for the given totally symmetric function, the PSBDD layout is superior both to the PLA and the PLA with the OR plane realized as a tree.

Now, we will generalize our observations by analyzing the PLA and PSBDD layouts for a symmetric function on n variables. Because the PSBDD representation is developed primarily for deep-submicron technologies, we assume that wire unit delay is equal to gate delay. The total number of cells (muxes) needed to realize an arbitrary (thus including the worst case) symmetric function of n variables as a PSBDD is $n^2/2$ and the area is equal to $2n^2$ gate area (the area of the multiplexer is equal to the area of 4 gates). The delay is equal to the number of levels, which is n, multiplied by two (two gate delays per level). For an arbitrary symmetric function realized as a PLA, the total number of columns is k (k is a number of terms in SOP) and the total number of rows is $2n$. Thus, assuming wire delay equal to gate delay, the delay in PLAs is $2n+k$ unit delays. The ratio given in Eq. (9) shows that for totally symmetric functions a PSBDD implementation has smaller delay than a PLA implementation. And from the area comparison in Eq. (10) we conclude that the area of the PSBDD layout is smaller for symmetric functions with k larger than n. It should be noted that for this comparison we have chosen the multiplexer implementation of PSBDDs, which is the least favorable.

$$\frac{\text{delay } PLA}{\text{delay } PSBDD} = \frac{2n+k}{2n} > 1 \quad \text{for } k > 0 \qquad (9)$$

$$\frac{\text{area } PLA}{\text{area } PSBDD} = \frac{(2n+1)k}{2n^2} > 1 \quad \text{for } k > n \qquad (10)$$

Using the same delay and area assumptions and the same evaluation procedure as above, we compare areas and delays for a number of benchmark functions implemented as PLAs and as PSBDDs, and we present results in Table II. For PSBDD calculations we substitute the number of function variables, n, with the number of levels in a PSBDD, l. ($l \geq n$).

7. EXPERIMENTAL RESULTS

The algorithms are coded in the *C* language, and run in the UNIX environment on SPARC workstations. In Table I the results for functions from the MCNC benchmark set are presented. The function name, the number of input variables, the output number, and the number of terms in the Sum-Of-Product (SOP) representation, optimized with Expresso, are all given in columns one, two,

three and four, respectively. In the next three columns the results for Fixed-Order Method are given; the number of loops (#lo), the number of levels (#le) and the number of nodes (#no). The results, the number of levels and the number of nodes, for Greedy Methods, G1 and G2, are given in the next four columns. In the next six columns, the results for three Look-Ahead Methods, L1, L2, and L3 are listed. The "–" means that the PSBDDs could not be generated using the particular algorithm with the given memory and time limitations.

It can be noted that for most of the tested functions the Look-ahead heuristics, especially L3, give better results than the Greedy ones. However for some functions, as for example *cm162a* and *count*, the results were best for L1 heuristic. In a few cases, *example* 2 (output 58), and *sao* 2 (all tested outputs) the Fixed-Order Method gives the better results than the other methods. Reported results were generated on relatively medium sized functions to allow us to better understand and analyze these algorithms for future research. In addition, there exist functional decomposition methods which can be used as preprocessing, therefore the size of the function is not of great concern at this time.

It can be easily seen that for these real-life functions we have generated PSBDDs with acceptable numbers of nodes and levels. For the majority of the tested functions the number of levels is smaller than two times the number of function variables. Only for function *clip* (both outputs) it is close to three. This function is currently being analyzed to give us hints for further improvements.

In Table II we compare PSBDDs with Reduced Ordered Binary Decision Diagrams (ROBDDs) generated by the algorithm from [3] and OBDDs generated with the algorithm from [5]. Column meanings are the same as in Table I. As was shown in Section 6, mapping OBDDs to a two-dimensional array is not a direct process. It usually requires adding dummy nodes to make routing feasible. Therefore, the final area and delay could be larger than suggested by the number of nodes and the number of variables. In case of ROBDDs, which are not planar, it is necessary to duplicate nodes to make ROBDDs planar and add dummy nodes for routing. Therefore, the initial conclusion can be drawn that, despite the larger number of nodes, PSBDDs are attractive alternatives and should be further investigated. In the same table we also included comparison with PLA for some benchmark functions. Both delay and area for PSBDDs and PLAs are calculated using Eqs. (9) and (10). However, for PSBDD calculations for non-symmetric function in the place of n, the number of function variables, we use the number of levels in PSBDD l, which is always larger or equal to n. As it can be observed, for a number of benchmark functions delays were smaller for PSBDD implementations. The area was also smaller for functions where the number of SOP terms is larger than a number of function variables. We expect the delay and area of PSBDDs to decrease with further improvements in PSBDD generation algorithms.

One must also remember that, to show the power of this approach, only non-decomposed benchmark functions were tested and no symmetry information was explicitly used in the present experiment. To understand better how our heuristics work we compared the variable orders generated by our heuristics with the functions' symmetry sets, as shown in Table III for L3 heuristic. The number of levels and the number of nodes in PSBDDs are given in columns labeled "#le" and "#no", respectively. It is indicated by "y" in column "match" if variables from a symmetry set are together in the variable order and "y" in column "on top" indicates that the symmetry set was placed at the beginning of the variable order. Upper case letters indicate complemented variables. For all but one of the reported functions we found a match and the symmetry set was placed on top of the order, which indicates that our heuristics are really good and generate good orders of variables. Interestingly, for a high percentage of functions that

TABLE III Variable order *versus* symmetry sets

Benchmark functions			L3			Symmetry sets	Comparison	
Name	# in	out #	Order of expansion variables	# le	# no	From [11]	Match	On top
b9	11	0	(aBfJ) ecHIKgd	11	28	aBfJ	y	y
c8	10	13	(heiabcdg) jf	10	29	heiabcdg	y	y
cm162a	10	2	(gBedf)Ch(ih)(g j)	11	24	aBedf, ih, gj	y	y
example2	14	58	(gDc)cEbaf(hijklmn)	15	31	gDc, hijklmn	y	y
example2	14	59	(cdaeh)bf (gi jklmn	14	23	cdaeh, gijklmn	y	y
k2	12	0	(GFD)aBieChK(jl)	12	20	GFD, jl	y	y
ttt2	14	15	(gakjCHEIflmn)	14	30	gakjCHEIflmn	y	y
sao	10	0	(JE)dbhaagGgcEhicAi	18	80	JE	y	y
vg2	15	0	(bdK)JCHEIflmnodb	15	30	dbKfim	partial	y
vg2	25	1	bdegijlmogr	25	91	ck	*n*	*n*
vg2	14	2	(acegil)nkHDjfmb	14	30	acegil	y	y
vg2	18	3	bdfhjlprGnEKOM(ci)ga	18	77	ci	y	y

would be characterized as non-symmetric in previous papers [3], we still find isomorphic nodes and realize these functions in lattices without repeated variables.

The generated results are good in terms of the small number of variable repetitions and the small number of nodes. The small number of repetitions is due to the following reasons: (1) there are many partial symmetries in these functions [11], and our heuristics take those symmetries into account (2) it was shown experimentally that many real-life functions have a lot of single variable symmetries [11], (3) even if there are initially no symmetric variables in a function, they can be created by repeating variables and applying the *Join-Vertex* operations.

Power analysis of the presented approach will not be discussed here, however it should be mentioned that the power dissipation associated with interconnects can be easily determined because in our approach the interconnect length and delays are known directly from the diagrams. That also allows to accurately estimate various circuit/layout parameters before the actual layout is completed. Using these algorithms we have demonstrated that a regular multilevel, two-dimensional representation of a function can lead to practical solutions.

8. CONCLUSIONS

Our experimental results demonstrate that effective heuristics can be developed to minimize the size of PSBBDs by proper variable ordering and very good results can be obtained for practical benchmark functions. Next, we showed that by adding one more expansion type, flipped Shannon, in Look-ahead methods, the numbers of nodes and levels were smaller when compared to pure Shannon PSBDDs. Dynamic generation of the variable order proved to be a good approach in situations where variable exchange-based methods are totally unpractical. At a time when interconnection delay is becoming the major factor in limiting device performance, these diagrams, which offer localized connections and well-defined structure, are one of the solutions to the problem.

Our method is good for completely as well as incompletely specified functions. It can be generalized by allowing more powerful neighborhood geometries (more inputs and outputs from neighbors while maintaining a regular structure) and by mixing control variables in levels, which is an extension to Free Pseudo-Symmetric Diagrams. This concept can also be extended to function representations based on XOR gates as was shown in [10].

In conclusion, there are important advantages to PSBDDs from the point of view of deep submicron technologies, because: (1) connections are short and based only on local cells abutting, (2) delays are equal and predictable, (3) late-arriving variables can be placed closer to the output, (4) logic synthesis can be combined with layout, so that no special stage of placement and routing is necessary or it can be a good starting point for specialized physical design algorithms, and (5) power estimation is simplified as the interconnect contribution can be easily calculated from the length of interconnections. Our Look-ahead methods of variable ordering for PSBDD generation offer significant improvements in reducing sizes and delays of PSBDDs.

References

[1] Akers, S. B., "A Rectangular Logic Array", *IEEE Trans. on Computers*, **C-21**, 848–857, August 1972.

[2] Bryant, R. (1986). "Graph Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Computers*, **C-35**, 677–691.

[3] Panda, S., Somenzi, F. and Plessier, B. F., "Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams", *Proc. of the International Conference on Computer-Aided-Design*, pp. 628–631, 1994.

[4] Sasao, T. (1993) (Ed.). "Logic Synthesis and Optimisation", Kluwer Academic Publishers.

[5] Chrzanowska-Jeske, M., Xu, Y., "Optimized Embedding of Incomplete Binary Tree to Two-Dimensional Array on Programmable Logic Blocks", *Proc. of the 39th Midwest Symposium on Circuits and Systems*, pp. 353–357, Ames, Iowa, August 1996.

[6] Tsai, C. C. and Marek-Sadowska, M. "Generalized Reed-Muller Forms as a Tool to Detect Symmetries", *IEEE Trans. on Computers*, **45**(1), 33–40, 1996.

[7] Kohavi, Z. (1978). "Switching and Finite Automata Theory", McGraw-Hill Inc.

[8] Sarabi, A., Song, N., Chrzanowska-Jeske, M. and Perkowski, M. A., "Comprehensive Logic and Layout Synthesis for Cellular FPGAs", *Proc. of the ACM/IEEE Design Automation Conference DAC'94*, pp. 321–326, June 1994.

[9] Chrzanowska-Jeske, M., Wang, Z. and Xu, Y., "A Regular Representation for Mapping to Fine-Grain, Locally-Connected FPGAs", *International Symposium on Circuits and Systems ISCAS'97*, pp. 2749–2752, June 1997.

[10] Perkowski, M., Chrzanowska-Jeske, M. and Xu, Y., "Reed-Muller Lattice Diagrams", *Proc. of IFIP W. G. Workshop on Application of the Reed-Muller Expansion in Circuit Design, Oxford, U.K.*, pp. 85–102, September, 1997.

[11] Wang, W. and Chrzanowska-Jeske, M., "Optimizing Pseudo-Symmetric Binary Decision Diagrams using Multiple Symmetries", *Proc. of the International Workshop on Logic Synthesis IWLS'98*, pp. 334–340, June 1998.

[12] Edwards, C. R. and Hurst, S. L., "Digital Synthesis Procedure under Function Symmetries and Mapping Methods", *IEEE Trans. on Computers*, **27**(11), 985–997, Nov. 1978.

[13] Rudell, R., "Dynamic Variable Ordering for Ordered Binary Decision Diagrams", *Proc. of the International Conference on Computer-Aided-Design*, pp. 42–47, Nov. 1993.

[14] Somenzi, F. BDD package, University of Colorado at Boulder.

[15] Arnold, R. F. and Harrison, M. A., "Algebraic Properties of Symmetric and Partially Symmetric Boolean functions", *IEEE Trans. Electron. Comput.*, **EC-12**, 244–251, June 1963.

[16] Lee, D. T. and Hong, S. J., "An Algorithm for Transformation of an Arbitrary Function to a Completely Symmetric Function", *IEEE Trans. on Computers*, **C-25**(11), 1117–1123, November 1976.

Authors' Biographies

Malgorzata Chrzanowsa-Jeske has served on the faculty of the Technical University of Warsaw, Poland; Design Automation Specialist at the Research and Production Center of Semiconductor Devices, Warsaw. Currently, she is an Associate Professor of Electrical and Computer Engineering at Portland State University in Portland, Oregon. Her research interests include logic and layout synthesis of VLSI circuits and systems, FPGA synthesis and architecture and design automation for deep submicron technology. She is a senior member of the IEEE, and a member of Eta Kappa Nu.

Yang Xu obtained his BS in East China Institute of Metallurgy in 1985, his MS in Beijing University of Science and Technology in Metallurgy in 1991. He got his second MS in Portland State University in Electrical and Computer Engineering in 1998. Currently, he is a Ph.D. student in Portland State University in Electrical and Computer Engineering. He is also a Tester Engineer in Lattice Semiconductor Corporation in Hillsboro, Oregon. His research interests include logic synthesis, logic decomposition and software development.

Marek A. Perkowski has served on faculty at the Institute of Automatic Control, Technological University of Warsaw, Warsaw, Poland; and the

faculty of Electrical Engineering at University of Minnesota, Minneapolis, MN. Currently he is a Professor of Electrical and Computer Engineering at Portland State University, Portland, Oregon. His current research interests include logic synthesis, finite state machines, multi-valued logic, FPGA computing, Data Mining, robotics, Artificial Intelligence, Testing and Design for Test, high-level synthesis, Formal Verification, and Image Processing. He is the Chair-elect of ISMVL-2000. His Internet address is: http://www.ee.pdx.edu/~mperkows/