VLSI DESIGN
1995, Vol. 3, Nos. 3-4, pp. 315-332
Reprints available directly from the publisher
Photocopying permitted by license only

A New Design Methodology for Two-Dimensional Logic Arrays

NING SONG
Lattice Semiconductor Corporation, 1820 McCarthy Blvd., Milpitas, Ca 95035

MAREK A. PERKOWSKI and MALGORZATA CHRZANOWSKA-JESKE
Portland State University, Department of Electrical Engineering P.O. Box 751, Portland, OR, 97207

ANDISHEH SARABI
Viewlogic Systems, Inc., 47211 Lakeview Blvd., Fremont, CA 94538

This paper introduces a new design approach that combines stages of logic and physical design. The logic function is synthesized and mapped to a two-dimensional array of logic cells. This array generalizes PLAs, XPLAs and cellular Maitra cascades. Each cell can be programmed to a wire, an inverter, or a two-input AND, OR or EXOR gate (with any subset of inputs negated). The gate can take any output of four neighbor cells and four neighbor buses as its inputs, and sends its result back to them. This two-dimensional geometrical model is well suited for both fine-grain FPGA realization and sea-of-gates custom ASIC layout. The comprehensive design method starts from a Boolean function, specified as SOP or ESOP, and produces a rectangularly shaped structure of (mostly) locally connected cells. Two stages: restricted factorization, and column folding, are discussed in more details to illustrate our general methodology.

Key Words: *Cellular FPGAs, Maitra Arrays, Multi-level Representation. Factorization, Folding*

1. INTRODUCTION

Gate arrays and standard cells are currently the most popular technologies used in ASIC design. On the other hand, the two level Sum-of-Products (SOP) structure is widely used in Programmable Logic Devices (PLDs). For two-level logic, there are effective synthesis tools for both SOP minimization [23] and Exclusive-Sum-of-Products (ESOP) minimization [25,28]. While the standard PLA is composed of an AND plane for product terms, and an OR collecting (output) plane, the recently introduced XPLA (Exor PLA—[25]) has an AND plane for product terms and an EXOR collecting plane. Another advantage of the two-level SOP or ESOP implementation is that the difficult placement and routing problems, inherent to gate array and standard cell technologies, are avoided.

The two-level approach, although commonly used in the PLD technology, requires large area and leads to low performance when applied to larger circuits. On the other hand, multiple-level-logic gate arrays and standard cell realizations can have high performance and consume a smaller area. The multiple-level-logic design, however, is much more difficult, both on the logic level and on the physical design level (placement and routing). Using architecture constraints during logic synthesis could decrease complexity of the physical design stage. But until very recently not much has been published on combining the logic and physical design stages.

Therefore, as the result of the above trade-off, there is an increased interest in developing new FPGA architectures that would combine the power and flexibility of multi-level circuits with the regularity and ease of use of logic based on two-level expressions. Two approaches: fine-grain FPGAs and Complex PLDs (CPLDs), have been recently proposed. CPLDs have partitioned PLA/PAL arrays connected by global routing channels. Fine-grain FPGAs have been developed by Concurrent Logic [5,6] (now Atmel [2]), Algotronix [1] (now Xilinx), Pilkington [14], Motorola [16], Plessey, Apple, Toshiba, and National Semiconductor. Although quite different in details, these fine-grain FPGA architectures have some very specific common properties.

Below we will create a *generic model* of a *"two-dimensional logic array"* that includes most of the important properties of these fine-grain FPGA architec-

tures. Although quite simple, the model is also well suited for custom ASIC design in sea-of-gates or similar technologies.

A very practical and interesting research problem related to new programmable architectures is to find some scientific evidence and experimental confirmation with respect to merits of the existing fine-grain architectures: how "good" are they? can they be improved? how? To our knowledge, while designing these architectures ([14] being the only exception), there was no research on selecting the best cells' functionality, their connection patterns, a number and location of buses, etc. The architectures were created purely on the "try and error" principle, with several modifications in next chips' generations and software redesigns. It is then very important to create new general methodologies and related prototyping software to help design new fine-grain architectures. We propose here such a methodology and related software. We will call it the *"Fine-Grain FPGA Designer's Work Bench"*.

Our approach to create optimal fine-grain FPGA architectures is through the *Device and Algorithm Co-Generation*. Conventionally, the devices are designed first. Next, the optimization methods are created to support the synthesis and mapping for these devices. When the design of FPGA architecture is completed, with no consideration of future physical design problems, the software tool design may become unnecessarily complex at the later stages. If the existing algorithms were evaluated on prototype architectures, and the corresponding improved algorithms were created concurrently with the devices, the creation of the high-performance tools would be significantly easier. The tools should be also able to better utilize all the distinct properties of the devices.

The best way to deal with circuits of high complexity is to preserve their regularity as much as possible. Logic synthesis and technology mapping are still performed separately (with a recent exception of combining the technology mapping with placement [7]). However, a good logic synthesis result does not necessarily guarantee the good result of technology and physical mappings, since the physical constraints are not taken into account at the stage of logic synthesis. For instance, algebraic factorization [4] is a popular method to generate multiple-level logic forms from two-level logic expressions. However, without taking certain layout-related constraints into account, such as the limited number and connectivity of buses, a synthesis result having less literals may need more space for routing than another result with more literals.

In the traditional approach where the logic optimization phase is followed by technology mapping and then placement and routing, a large number of logic cells are used for wiring connections or left unused at all. This problem is mainly caused by not preserving local connectivity during the synthesis steps. Therefore, frequently, local buses are used to complete even very short connections, which increases circuit delay. Better solutions that use different logic implementations with a larger number of logic cells but with predominantly local connections are lost during the technology mapping. The traditional technology mapping algorithms optimize area by minimizing the number of logic cells used, and circuit delay by optimizing the number of logic levels. In the "macro block" approach which is currently used in the industry, a technology independent multi-output representation of a Boolean function is covered with a minimum number of small standard subfunctions (macros) which have no uniform shapes, and do not preserve local connectivity between macros. Consequently, the number of cells which need to be used for routing between macros is very large. On average, about 70% of the area occupied by the design in ATMEL 6000 series fine-grain FPGAs [2] is wasted if the traditional synthesis methods are used [6].

Several approaches have been proposed that use various layout constraints during logic synthesis. The first research on applying variable ordering in factorization is reported in [26]. The approach based on trees and decision diagrams (which are Directed Acyclic Graphs—DAGs) [8,15] has been also adapted to fine-grain FPGAs [27,30]. It makes use of the diagrams' regularity and the specific types of logic gates (AND/EXOR, MUX), used in these decision diagrams. These gates are also well-suited to the existing devices from Atmel or Motorola [27,30]. In some cases, however, when the circuit is finally mapped to a rectangular area, the triangular structure of the tree/DAG decomposition may waste a large amount of area for routing.

Therefore, we propose here a totally new approach to combined logic synthesis and physical design. Starting from an observation that the architectures have rectangular arrays of simple, locally connected cells, we create our design method especially for such arrays. The *"generic two-dimensional array"* uses two-input AND, OR and EXOR cells with local connectivity and limited numbers of horizontal and vertical buses. Such generic model includes in itself several simpler, constrained models, each of which can be both a base of logic synthesis/physical design and serve to create a new FPGA architecture with restricted cell functionality and connections. For instance, below we restrict ourselves to the simplified model composed of two distinct planes: the complex (input) plane and the collecting (output) plane. The input variables of the input plane are in vertical buses. The linear sequence (a row of the input plane in the array) of AND, OR and EXOR operators with corresponding literals is called *a Maitra term*. The outputs of the Maitra terms are given to horizontal buses. The Maitra term is

therefore a generalization of the AND term (product term). (AND terms are realized in the AND planes of PLAs realizing the SOPs. The name "Maitra term" comes from "Maitra cascades" [17].) Similar to PLAs and XPLAs, the collecting (output) columns of the two-dimensional array use OR or EXOR gates.

The given above, **particular two-plane specialization** of the "generic two-dimensional logic array model" will be called the *"Complex Maitra Logic Array"* (CMLA). This model allows for simpler logic synthesis methods, and also can be a base for designing new architectures.

The CMLA concept is well suited for both fine-grain Field Programmable Gate Arrays (FPGAs) and ASIC design. CMLA is a powerful generalization of PLAs since the number of Maitra terms for any Boolean function is much larger than both the number of prime implicants in SOP form of this function, and the number of ESOP terms used in ESOP form of this function. Unfortunately, there are no efficient methods in the literature for finding Maitra terms for an arbitrary Boolean function, and particularly for a multi-output function.

In addition, similar to PLAs and gate matrix layout [29], our CMLAs can be folded in many ways. All well-known algorithms for folding and gate-matrix layout can be thus used [7,9,10,11,12,13,29]. However, both the properties of our general array model and the specific properties of particular commercial FPGAs call for new approaches to this folding problem [24].

The comprehensive approach to both the development of new architectures and the creation of software for existing architectures, proposed here, includes two stages:

1. *Logic synthesis* which takes the geometry and layout constraints into account to create a CMLA in which every output function is an OR or EXOR of Maitra terms.
2. *Folding* the CMLA in order to further decrease the area of the layout.

Each of the above stages can be solved in several ways, and this paper attempts to emphasize the general model of the two-dimensional array and the associated design methodology, rather then the details of any particular method to solve the partial problems. Thus, we illustrate the logic synthesis stage with two possible approaches: the *orthogonal canonical expansions*, and the *restricted factorization*. The second approach will be presented in more detail. The result of the logic synthesis stage is a logic structure, which similarly to other multiple-level logic structures, has the advantages of high speed and reduced area. In addition, however, the routing problem involved in our approach is greatly simplified. Although the CMLA structure is more general than PLAs and gate arrays, it still preserves their routing regularity. A Boolean function realized by such a CMLA can be easily mapped to a rectangular area on the chip.

The second, folding, stage can be solved in a "technology independent" way, illustrated here. Or, it can take into account particular cell properties of the given fine-grain FPGA to make the folding even more efficient. One solution to the technology-specific folding, for Atmel 6000 architecture, is presented in [24].

The paper is organized as follows. In section 2 we introduce the general model of a two-dimensional cellular array that includes several existing FPGA architectures and technologies, and can be also used to prototype new ones. Section 3 describes the general logic synthesis for CMLA model and introduces briefly two particular methods: the synthesis based on orthogonal uxf-forms and the restricted factorization. Section 4 formally introduces the mathematical apparatus necessary to create the complex terms, the generalization of Maitra terms generated in the restricted factorization. Section 5 gives the complete algorithm to generate the complex terms, and section 6 illustrates the application of this algorithm to a circuit example. Section 7 discusses the column folding problem for our arrays and presents the algorithm and an example. Section 8 discusses the results. Conclusions are presented in section 9. Proofs of theorems and other details are in the Appendix.

2. THE GENERAL MODEL OF A TWO-DIMENSIONAL LOGIC ARRAY

Cellular arrays were studied extensively during sixties and seventies [3,17,18,19,31]. In these studies, however, the connectivity patterns of cells were too restricted and the buses were mostly absent. Because of these limitations, when the number of inputs of a function becomes larger, the number of cells grows rapidly, often exponentially. The classical cellular arrays were then never commercialized, and the PLD and FPGA technologies were developed with no reference to them.

Below we propose a generalized architectural model of several fine-grain devices, that includes also some classical cellular array models. Our "generic" model is a two-dimensional array of identical cells with the following properties:

1. Each cell can be configured into a 2-input 1-output basic logic gate. The basic logic gate can be an AND gate, an OR gate, or an EXOR gate. Programmable inverters at each input are assumed to be available inside the cell. The cell can then realize an arbitrary function of at most 2 inputs.

2. Horizontal buses are connected to all cells in the row and vertical buses are connected to all cells in the column.
3. Each cell has connections to its four adjacent cells. The cells at a border or a corner of an array have three or two adjacent cells, respectively.
4. Each cell can either get its two inputs from any two of its four adjacent cells, or one input from any of its adjacent cells and one input from one of buses connected to it. (Selection of inputs is done by electrically configurable multiplexers). There are no restrictions on which one of the two inputs should be from which adjacent cell, or which input should be from which bus.
5. Each cell can send its output to any bus and/or any adjacent cell. The only restriction is that a cell cannot connect both its input(s) and output to the same adjacent cell, or the same bus.
6. There are some other constraining parameters such as the size of the cells, the number of buses, and the number of storage elements. For ASIC design, these constraining parameters can be modified in software. For FPGA design, these parameters are fixed.

The generic architecture proposed by us is shown in Figure 1. We will call it the *"Generic 2-Dimensional Logic Array"*, or *"2-D Array"*, for short.

The cells which are programmed (electrically configured, personalized) to logic gates will be called *logic cells*. A *routing cell* is a cell which passes a signal (wire) only. An *empty cell* is a cell unused in a mapping.

The 2-D Array approach provides a compromise between the two main mapping approaches to fine-grain FPGAs; i.e. "module block" and "cellular array" approach. Each of these approaches provides advantages as well as disadvantages for the mapping problem. In the module block approach, the general function is decomposed into smaller subfunctions which would not have uniform shapes but can be optimized locally. On the other hand, the modules in the module block approach would not have uniformity of local communications and with the routing restrictions of fine-grain FPGAs lead to wasteful routing (high percent of empty and routing cells). In the case of cellular array, the whole function is mapped into regular structures which can grow significantly with a large number of input variables. However, the routing is local and therefore best fits the fine-grain routing resources.

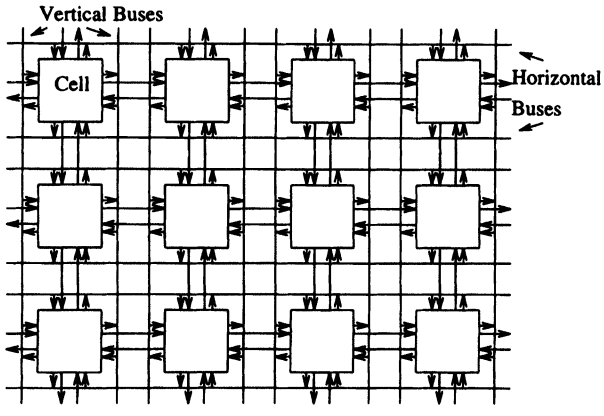

FIGURE 1 Generic Architecture of a Two-Dimensional Logic Array.

It is our opinion that the design of the next generation of fine-grain FPGA devices should be based not only on the design experience but also on experimenting with software tools for "generic" fine grain devices, for instance as the one proposed here. The device architect should experiment with these tools by assigning values to various constraining parameters, such as: cell's personalizations, number of inputs, cells' connectivity, number and location of buses (vertical, horizontal, oblique), types of buses (local, global, intermediate), hierarchy, and possible others. Therefore, when used with some particular set of constraints, our methodology and "generic algorithms" produce an efficient tool for respective fine-grain FPGA technology. When used without any constraints, the proposed approach produces the tools for custom ASIC logic/layout co-design.

The CMLA model is created from the "generic 2-D Array" by separating the array into two planes: "complex term plane" and "collecting plane" and restricting correspondingly the connectivity and reconfigurability of cells in each plane. For instance, by using only Maitra terms in the complex plane, each cell there can have only one input from a neighbor and one input from a vertical bus, and send its output to only one neighbor. Similarly, the cells in the collecting plane can be programmable only to OR and EXOR. All such restrictions simplify greatly the cell and its connectivity pattern. This decreases additionally the total area and speeds-up the circuit. Similarly, other new models can be created by imposing certain constraints on our generic 2-D Array model.

3. LOGIC SYNTHESIS APPROACHES FOR THE CMLA MODEL

The following methods from the literature can be adapted to generate terms for CMLAs:

1. Classical cellular array methods [3,17,18,19,31].
2. Methods based on orthogonal expansions [21] and Universal XOR Forms [22].
3. The constrained factorization [24].

The classical methods seem to be too restricted for both the generic and CMLA models, but some of the algebraic ideas introduced by them seem still worthy of further investigations, and can be used to improve the efficiency of the methods proposed here. In the remaining of section 3 we will introduce two new methods: one is based on Universal XOR Forms [22] (section 3.1), and the other is based on restricted factorization to complex (Maitra) terms (section 3.2). While the first (Boolean/spectral) method is more general and usually leads to better solutions because of extremely large space of solutions it searches, the second (algebraic) method in our current implementation leads to much faster programs.

3.1. Synthesis Based on UXF Forms

3.1.1. Universal XOR forms In the vector space Ψ over *GF(2)* formed by the set of n-variable switching functions under addition mod-2, every switching function can be represented uniquely as a linear combination of the *basis functions* [22]. The task of the identification of all canonical forms of the switching functions in this field thus entails the identification of all possible bases of the 2^n-dimensional vector space Ψ. A *Universal XOR form (UXF)* is a basis vector in vector space Ψ. Each term in the *UXF* is a *basis function.*

If the basis functions are realized as products of literals, the basis functions will be called *monoterms*. For instance, the set of all *UXF* forms includes all possible AND/EXOR canonical forms including all known (Reed-Muller, Fixed-Polarity Reed-Muller, Generalized Reed-Muller, Kronecker-Reed-Muller), and lesser-known AND/EXOR forms [8]. Some *UXF* forms also include terms which require gates other than AND and NOT for their realization. They include various AND/OR/EXOR canonical forms [21,22].

One well known XOR canonical form is that of the *Reed-Muller Canonical (RMC)* form. The standard canonical sum-of-minterms form can also be considered an *UXF*. As an example, the monoterms of the *RMC* (the coefficient of the Reed-Muller expansion) and the minterms are related by the following nonsingular matrix for the case of functions of two variables:

$$\begin{bmatrix} 1 \\ a \\ b \\ ab \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{a}\,\bar{b} \\ a\,\bar{b} \\ \bar{a}\,b \\ a\,b \end{bmatrix}$$

The Reed-Muller expansion is a particular example of an orthogonal expansion and the RMC is a particular *UXF* form. In general, the coefficients of the orthogonal expansion for a Boolean function are obtained by multiplying the matrix of this expansion by the vector of minterms of this function. Matrix of expansion is an inverse to the matrix of basis functions [21]. By repeating this procedure for the expansion matrices corresponding to all the bases from some family **F** of bases, and selecting the base for which the minimum number of coefficients are non-zero, one obtains the exact minimal form in this family **F** of bases.

The total number of *UXF* forms was shown to be

$$\frac{2^{(2^n-1)(2^n-1)}}{2^n\,!} \prod_{i=1}^{2^n} (2^i - 1)$$

where n is the number of variables in the function [22].

Among all these forms, there are those families of forms which have easy circuit realization for a given fine-grain FPGA architecture.

3.1.2. CMLA synthesis using universal XOR forms *UXF* forms can be used in the CMLA approach. In such case uxf-terms are realized as rows of the complex plane (called the *orthogonal plane*, since all terms realized by it are orthogonal functions). The output plane includes only EXOR gates. The general scheme of such restricted CMLA is shown in Figure 2. The CMLA array is comprised in this case of the orthogonal (or basis) plane realizing the terms, and an EXOR plane collecting them. Each level (row) in the orthogonal plane realizes an

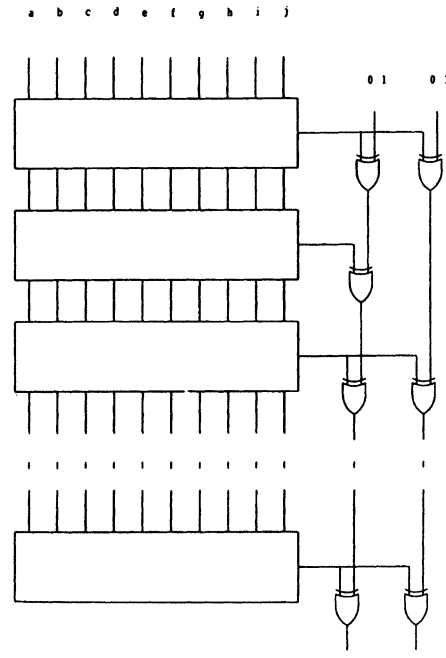

FIGURE 2 CMLA realization of the UXF form.

uxf-term of the function. These terms are then EXORed together in the EXOR plane.

In the orthogonal plane, it is assumed that the primary inputs are carried across the levels through buses. The uxf-terms are then constructed through allowable gates in the level. As an example, the product *ac* can be produced by getting a signal from the bus, passing it through the "*b*-cell" via a wire (a connection cell) and then ANDing *a* and *c* in the "*c*-cell". In similar way, various terms composed from connection cell ("wire"), AND, OR, and EXOR cells can be realized in the orthogonal plane. An example is shown in Figure 3.

While the number of all *UXF* forms is enormously large, the constraints of the technology limit the number of basis vectors that can be utilized in a given architecture. As the rows of arrays realize the basis elements of a given basis which have a coefficient of 1, it may not be possible to realize every possible basis element in a single row. As an example, let us assume that the array is comprised of only AND gates. Furthermore, let us assume that one of the basis elements is $a + b$, where a and b are two primary inputs. In this architecture, the OR-type basis elements can not be realized. Therefore, basis elements have to be chosen based on the target architecture. Or, vice versa, the new target architecture may result from a particularly powerful family of bases.

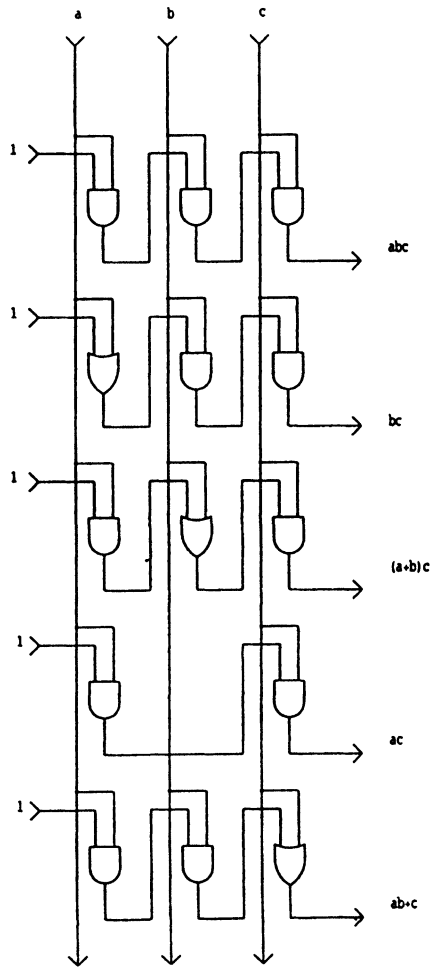

FIGURE 3 An example of an orthogonal plane.

Obviously, for every family of forms there exists the best form, the one which has the least number of non-zero coefficients. Such coefficients correspond to the uxf-terms—those basis functions that actually appear as rows in the layout realization of the function. For any type of cells and their connectivity pattern, one creates a family of basis functions, and next finds the corresponding expansion matrices and minimal forms. In [22] some narrower families of bases are presented, The bases that have all basis functions composed of connection cells and two-input AND gates create Fixed Polarity Reed Muller forms. The bases that have all basis functions composed of two-input AND and OR gates create AND/OR canonical forms [22]. Bases with functions composed of two-input AND, OR and EXOR gates can also be identified. The expansion of a given Boolean function in a base is done by multiplying the matrix of the expansion by the vector of minterms of this function. The procedure is repeated through all expansion matrices of the set of bases. The best form is found for which the given function has the minimum number of uxf-terms [22].

This method is very general and can be applied to any cells and connectivity patterns, potentially it can produce results of very small area. It can also lead to the development of new fine-grain architectures. However, its current realization is not very efficient numerically, since it takes much space and time to calculate all expansion matrices and next to multiply them by the vector of minterms. Therefore, another method is also presented below.

3.2. Synthesis Based on Constrained Factorization

3.2.1. Maitra terms and Complex terms In this section, new concepts, *Maitra term* and *complex Maitra terms* will be first introduced. Then our method of constrained factorization will be discussed. An example will be given to help present the principle of our method.

Definition 1A: A **forward Maitra term** is defined recursively as follows:

1. a literal is a forward Maitra term.
2. if M is a forward Maitra term then

$M\,a$, $M\,\bar{a}$, $M \oplus a$, $M \oplus \bar{a}$, $M + a$, and $M + \bar{a}$

are also forward Maitra terms if no literal or its complement appears in the string more than once.

Definition 1B: A **reverse Maitra term** is defined recursively as follows:

1. a literal is a reverse Maitra term.
2. if M is a reverse Maitra term then

a M, $\bar{a}$ M, $a \oplus$ M, $\bar{a} \oplus$ M, a + M, and $\bar{a}$ + M

are also reverse Maitra terms if no literal appears in the string more than once.

Forward and reverse Maitra terms are called *simple Maitra terms.*

Example 1: Each of the following expressions represents a forward Maitra term:

$$(a\bar{b})+c, \qquad (a+b)c,$$

$$(a \oplus b)+\bar{c}), \qquad ((c\bar{b})+a) \oplus d$$

Each of the following expressions represents a reverse Maitra term:

$$c+(a\bar{b}), \qquad c(\text{a+b})$$

Example 2: ((a b) + $\bar{b}$)c is not a Maitra term because the literal b appears twice.

Example 3: $a + (b\ \bar{c}) + d$ is not a forward Maitra term because it cannot be generated from the forward Maitra term definition (analyzing the expression from right to left, $a + (b\ \bar{c})$ is not a forward Maitra term). However, if the order of variables is changed to b, c, a, d, then $(b\ \bar{c}) + a + d$ becomes a forward Maitra term.

This example shows that whether a given logic expression is a Maitra term or not, depends on the order of variables in this expression. Some expressions which are not Maitra terms can become Maitra terms by changing the order of variables in them.

For every order of input variables, a Boolean function can be decomposed to an OR or EXOR of Maitra terms. This is always possible, since the AND terms (used in SOPs and ESOPs) are particular cases of the Maitra terms.

The following two figures explain the reason for introducing the concept of the Maitra term. Figure 4 realizes a function $f_r = (a + b)\ (c + d)$. This is a factorized expression implemented by an array of three cells. Note that a routing wire is needed. Function f_r is not a Maitra term and it cannot be changed to Maitra term by changing the order of literals and operators. Therefore, a wire will always be needed for above function f_r to implement the three operators in a row. In a fine-grain FPGA implementation, this routing wire can be realized by a bus or a row of cells. When a Boolean function becomes more complicated, the number of routing wires needed increases. This increases the routing complexity

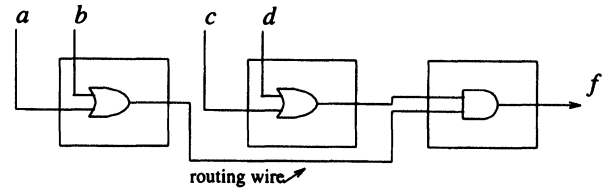

FIGURE 4 Realization of factored term that is not a Maitra term.

but in a custom ASIC implementation, but one can add wires freely to the routing channels. However, in an FPGA implementation, the number of buses is very limited, thus the routing must be done by programming the logic cells to wires. This is a big waste of resources, and this is why some factorized forms are much more useful than some others.

Figure 5 shows a realization of a forward Maitra term $f = ((a + b)c + d)\,e$ that resulted from the restricted factorization. Obviously, there is no routing wire needed, assuming order of variables a, b, c, d, e. Lack of routing wires is convenient in both ASIC and FPGA implementation. However, for some other orders of variables, such as d, c, a, b, e, several additional wires would be required. Therefore, the order of variables in the realization must reflect the order in the Maitra term. By flipping vertically the schematic of a forward Maitra term from Figure 5 one would obtain the schematic of a reverse Maitra term.

Definition 1C: A bidirectional Maitra term has the form

M1 *operator* M2

where *operator* is a Boolean function of two arguments, M1 is a forward Maitra term, and M2 is an reverse Maitra term, such that M1 and M2 have different sets of variables and do not exhaust together all input variables of the function.

For instance, M1 $\oplus$ M2 = {(ab) + c} $\oplus$ {e (f + g)} is a bidirectional term of function $f(a, b, c, d, e, f, g)$ since M1 is a forward term on variables $\{a, b, c\}$, M2 is a reverse term on variables $\{e, f, g\}$, sets $\{a, b, c\}$ and $\{e, f, g\}$ are non-overlapping, and variable d is not used in any of these sets.

Fig. 6 illustrates the realization of the bidirectional term.

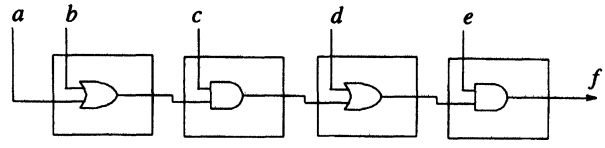

FIGURE 5 Realization of a forward Maitra term in a row of a CMLA.

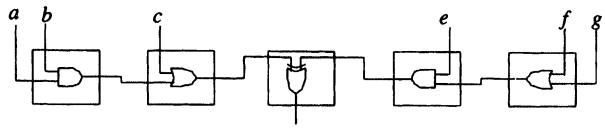

FIGURE 6 Bidirectional Maitra term realized by a row of a CMLA.

Definition 1D: A **complex Maitra term** (**complex term** for short) is a forward Maitra term, a reverse Maitra term, or a bidirectional Maitra term.

After the product terms have been factorized to complex terms, the next stage is to perform the output column folding. In this stage, the number of complex terms is known. Each complex term is connected to one or more output functions. To minimize the area, a proper order of complex terms is found such that the number of overlapping nets is minimized (net is a list of terms and associated output functions). The nets that do not overlap are next put to the same column. This is similar to the gate matrix problem in which non-overlapping nets can be put to the same track.

Example 4: Given a SOP expression for a three-input two-output function:

$$f_1 = (a + b)\, c + d + \bar{a}\,\bar{b}\,\bar{c}$$

$$f_2 = \bar{a}\,\bar{c}\,\bar{d} + a\, b\, c\, d$$

The first function has two complex terms, $(a + b)c + d$ and $\bar{a}\,\bar{b}\,\bar{c}$. The second function has also two complex terms. A realization shown in Figure 7 needs two output columns. The rows are now permuted to avoid overlap of nets connected to each column. Then the two output columns can be combined into one column, as shown in Figure 8, and the total number of output columns is reduced.

Based on the above discussion, the outline of the combined factorization/folding approach is the following.

1) Start from a minimized SOP expression, a minimized ESOP expression, or a minimized mixed SOP/ESOP expression. Use one column for each input variable and one column for each output variable. Use one row for each product term. For an expression with n inputs, m outputs, and p product terms, this function is mapped into $n + m$ columns and p rows. This is the initial solution. It does not take into account factorization and folding, and is thus the worst case solution, to which our solutions will be compared.
2) Factorize product terms to complex terms, such that each complex term can then be put in one row. After the factorization, the number of complex terms is not greater than the number of product terms.
3) To reduce the number of rows as much as possible, perform step 2) iteratively and reshape the product terms. Repeat until some cost improvement criteria are satisfied.
4) Permute the rows with complex terms in order to minimize the number of overlapping nets.
5) Minimize the number of columns by folding, merging the nonoverlapping nets into the same columns.

3.2.2. A complete example of factorization and folding

The following example of a two-bit adder illustrates the above procedure. This function has 4 inputs and 3 outputs and has been minimized as an ESOP of 8 product terms:

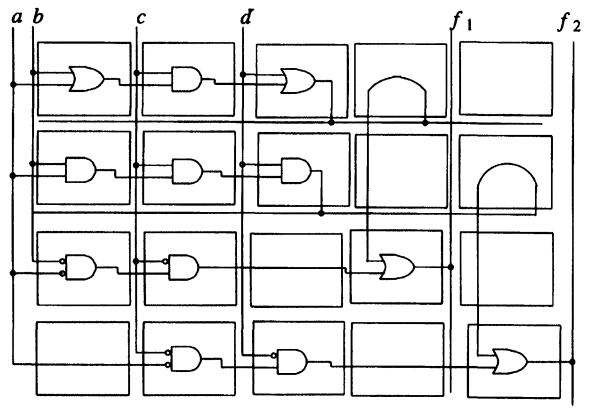

FIGURE 7 Initial CMLA for example 4.

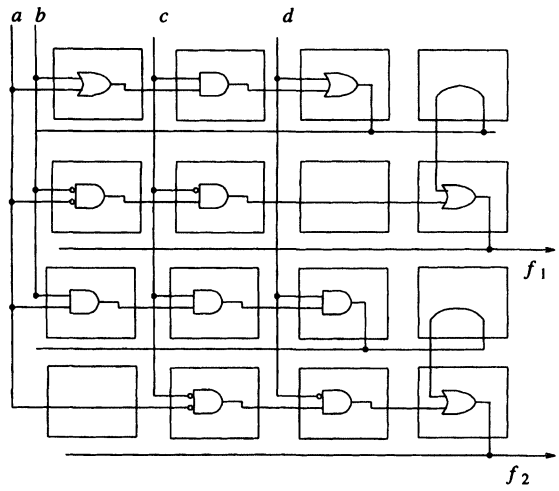

FIGURE 8 Folded CMLA for example 4.

$$f_0 = a\,c \oplus \bar{a}\,b\,d \oplus b\,\bar{c}\,d$$

$$f_1 = b \oplus d$$

$$f_2 = c \oplus b\,d \oplus a$$

Thus the initial solution requires 8 rows and 7 columns. Each product term is mapped into one row. There are 4 columns for inputs and 3 columns for outputs. By setting the order of the input variables as *(b, d, a, c)* the three product terms in f_2 can be combined into a complex term. Three product terms in f_0 can be factorized to two complex terms as shown in Figure 9.

A cube (product term) B and cube C in Figure 9(a) can be reshaped to B' and C' in Figure 9(b). Since the true minterm 1111 is covered by three cubes A, B' and C', the operators between these cubes can be either EXOR or OR:

$$f_0 = a\,c \oplus \bar{a}\,b\,d \oplus b\,\bar{c}\,d = a\,c \oplus b\,c\,d \oplus a\,b\,d$$
$$= a\,c + b\,c\,d + a\,b\,d = (b\,d + a)\,c + b\,d\,a$$

The result of the factorization is:

$f_0 = (b\,d + a)c + b\,d\,a$
which has two complex terms;
$f_1 = b \oplus d$
which has one complex term;
$f_2 = b\,d \oplus a \oplus c$
which has one complex term.

After the output folding, the final result is shown in Figure 10.

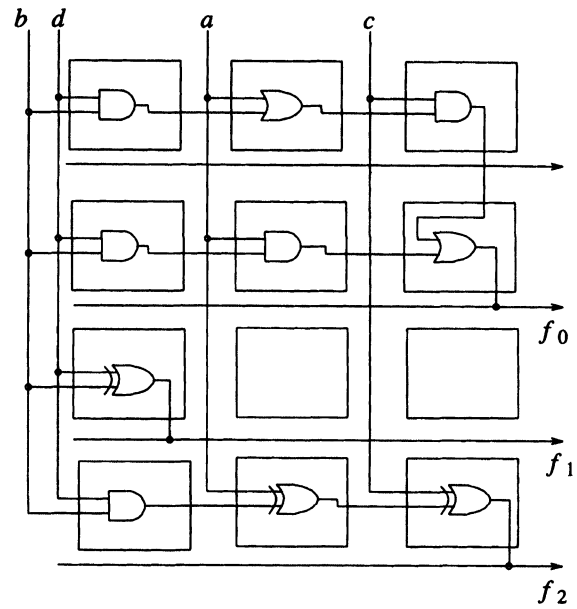

FIGURE 10 The final CMLA of the two-bit adder after folding.

4. RESTRICTED FACTORIZATION THEORY

Since the outlined above factorization problem involves more constraints than the standard factorization, and since the conventional algebraic division method [4] does not take these constraints into account, we have developed a new factorization method for this specific problem. We call this *restricted factorization*. The new method is based on *cube calculus operations* [23,25,28]. In this section, the concepts of *distance* and *difference* of two product terms, and a cube operation—*exorlink* are first introduced. Then the method to generate complex terms from product terms is discussed.

The algorithm to combine product terms to complex terms is based on calculating the *difference* and the *distance* of the cubes for every pair of cubes representing product terms. This is used to decide whether two product terms can be combined to a complex term. It also determines the cases when the cubes need to be reshaped in order to increase the possibility of re-combining them. This reshaping is done using the *exorlink operation.*

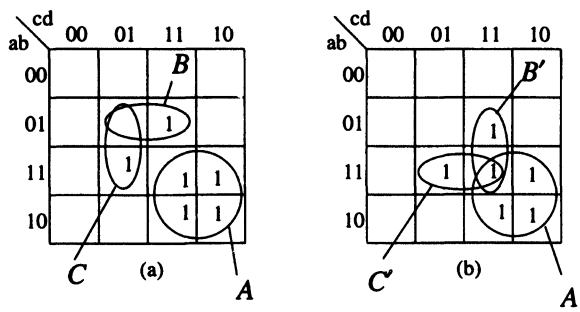

FIGURE 9 Example of Reshaping Cubes before Factorization to Complex Terms.

4.1. Definitions

In positional cube notation, a literal with a *positive polarity* (a variable with no negation) is coded as 10, a literal with a negative polarity (a variable with negation) is coded as 01, and a missing literal is coded as Figure 11.

Definition 2: The **distance of two terms** is the number of variables for which the corresponding literals of these terms have different polarities.

Definition 3: The **difference of two terms** is the number of variables for which the corresponding literals of these terms have different values.

Here "different values" means different codings, and "different polarities" means disjoint codings. For instance, 11 and 10 are different values, 10 and 01 are also different values. For binary logic, the only case of different polarities are 10 and 01. The difference of two product terms T_i and T_j is indicated by *difference* $(T_i, T_j) = d'$. Similarly, the distance of T_i and T_j is indicated by *distance* (T_i, T_j), $= d''$.

Example 5: Given are three terms $T_1 = a\ c$, $T_2 = \bar{a}\ b\ d$, and $T_3 = b\ \bar{c}\ d$. The difference of T_1 and T_2 is 4, because all four pairs of literals are different. The distance of T_1 and T_2 is 1, because the literals of variable a have different polarities. The difference of T_2 and T_3 is 2, because for variables a and c there are different literals. The distance of T_2 and T_3 is 0, because no literal has different polarities.

Let $T_1 = \hat{x}_1 \ldots \hat{x}_n$ and $T_2 = \hat{y}_1 \ldots \hat{y}_n$ be two terms. The *exorlink* [28] of terms T_1 and T_2 is defined by the following formula:

$$T_1 \otimes T_2 = \oplus \{\hat{x}_1 \ldots \hat{x}_{i-1}\ (\hat{x}_i \oplus \hat{y}_i)\hat{y}_{i+1} \ldots \hat{y}_n \mid \text{for such } i = 1, \ldots, n, \text{ that } \hat{x}_i \neq \hat{y}_i\}$$

Example 6: Given two product terms $a\ b\ e$ and $\bar{a}\ \bar{b}\ c\ d\ e$. The exorlink of these two terms is shown in Figure 11.

In Figure 11a, three arrows indicate the three pairs of literals with different values. Since the difference of the two terms is three, three resultant cubes are generated. Figure 11b shows the generation of the first resultant cube. The first literal in the resultant cube is copied from the first term. The second literal is the result of EXOR operation of the corresponding literals from the first and the second terms (remember, EXOR is performed on the positional cube notation, therefore, $1 \oplus - = [01] \oplus [11] = [10] = 0$). The remaining three literals in the resultant cube are copied from the second term. The second and the third resultant cubes are generated in a similar way by performing EXOR operation on the third and fourth literals, respectively. The final result is an ESOP of three terms:

$$a\ b\ e \oplus a\ \bar{b}\ c\ d\ e = a\ c\ d\ e \oplus a\ b\ \bar{c}\ d\ e \oplus a\ b\ \bar{d}\ e$$

Given two product terms, an exorlink operation generates a set of resultant product terms. The number of resultant product terms is equal to the difference of the two given product terms.

Definition 4: Two product terms T_1 and T_2 are referred to as **directly combinable**, *if these* two product *terms are in one of the following forms,*

$$T_1 = \dot{x}_1 \dot{x}_2 \cdots \dot{x}_{i-1}\ \dot{x}_{i+1} \cdots \dot{x}_n \qquad (4.1)$$
$$T_2 = \quad \dot{y}_i \dot{y}_{i+1} \cdots \dot{y}_n$$
$$\dot{x}_j = \dot{y}_j \text{ for } j \geq i + 1$$

$$T_1 = \dot{x}_1 \dot{x}_2 \cdots \dot{x}_{i-1}\ \dot{x}_i \dot{x}_{i+1} \cdots \dot{x}_n \qquad (4.2)$$
$$T_2 = \quad \dot{y}_{i+1} \cdots \dot{y}_n$$
$$\dot{x}_j = \dot{y}_j \text{ for } j \geq i + 1$$

In equation (4.1), the two product terms can be combined to

$$(\dot{x}_1 \dot{x}_2 \cdots \dot{x}_{i-1} \oplus \dot{x}_i)\ \dot{x}_{i+1} \cdots \dot{x}_n$$

Example 7: $a\ b\ d\ e \oplus c\ d\ e = (a\ b \oplus c)\ d\ e$

In equation (4.2), the two product terms can be combined to

$$(\dot{x}_1 \dot{x}_2 \ldots \dot{x}_{i-1} \dot{x}_i \oplus 1)\ \dot{x}_{i+1} \ldots \dot{x}_n =$$
$$(\tilde{x}_1 + \dot{x}_2 + \cdots + \tilde{x}_{i-1} + \tilde{x}_i)\ \dot{x}_{i+1} \cdots \dot{x}_n$$

here $\tilde{x}_i$ indicates the negation of $\dot{x}_i$.

Example 8: $a\ \bar{b}\ c\ d\ e \oplus d\ e = (a\ \bar{b}\ c \oplus 1)\ d\ e = (\bar{a} + b + \bar{c})\ d\ e$, the two product terms are directly combinable.

4.2. Checking if Two Terms are Combinable

In the following the criteria for combining product terms are discussed. The method is based on calculating the distance, the difference and other properties of the two terms. Let us observe that in case of ESOP minimization, two product terms can be combined only if their difference ≤ 1. However, in case of restricted factorization there are more opportunities to create complex terms, since two product terms of any difference may be combinable, however in different ways for various values of the difference.

Example 9: Given are two product terms $a\ \bar{b}\ e$ and $a\ b\ \bar{c}\ d\ e$. The difference of these two terms is 3. So, these two terms can not be combined into a product term. They

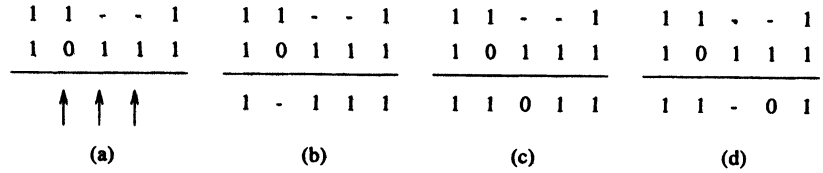

FIGURE 11 The method of calculating the exorlink of product terms ***abe*** and ***ab'cde.***

can, however, be combined into complex terms as follows: $a\ \bar{b}\ e \oplus a\ b\ \bar{c}\ d\ e = a\ \bar{b}\ e + a\ \bar{c}\ d\ e = a\ (\bar{b} + \bar{c}\ d)\ e = (\bar{c}\ d + \bar{b})\ a\ e$.

For convenience, two given product terms in the forms $T_1 = \hat{x}_1\hat{x}_2\ldots\hat{x}_n$ and $T_2 = \hat{x}_1\hat{x}_2\ldots\hat{x}_n$ are assumed. Without loss of generality, it is assumed that the pairs of literals which have different values appear at the left side in the terms. In other words, if the difference of the terms is 1, then $\hat{x}_1$ is different in the two product terms. If the difference of the terms is 2, then $\hat{x}_1$ and $\hat{x}_2$ are different in the two product terms.

4.2.1. Difference $(T_1, T_2) \leq 1$

4.2.1.1. Difference $(T_1, T_2) = 0$

Difference = 0 means these two terms are identical. In case of an ESOP, since $A \oplus A = 0$, these two product terms can be removed. In case of a SOP, since $A + A = A$, one of the terms can be removed.

4.2.1.2. Difference $(T_1, T_2) = 1$

(1) If distance $(T_1, T_2) = 0$, then $\hat{x}_1$ only in one term. Since $1 \oplus a = \bar{a}$, these two product terms are directly combinable.

(2) If distance $(T_1, T_2) = 1$, then $\hat{x}_1$ appears in both terms, but in different polarities. Since $a \oplus \bar{a} = 1$, these two product terms are also directly combinable.

Theorem 1: If the difference of two product terms is greater than 1, then these two product terms are directly combinable if and only if their distance is 0 and from all the literals that do not appear concurrently in both terms only one literal can appear in a term.

Proof: See in the Appendix.

If two product terms are not directly combinable, after reshaping, they may or may not be combinable.

Example 10: $\bar{a}\ c \oplus a\ b\ c$ are not directly combinable. By reshaping these two terms to $c \oplus a\ \bar{b}\ c$, they become combinable, since $c \oplus a\ \bar{b}\ c = (1 \oplus a\ \bar{b})\ c = (\bar{a} + b)\ c$.

Example 11: $a\ b \oplus \bar{a}\ c$ are not combinable. Since the difference of these two terms is 3, performing exorlink operation on these two terms will generate three resultant product terms. These three resultant terms can not be combined to a complex term. Further exorlink operations can be applied to any two of the three resultant terms so that they are reshaped to other product terms. By trying all the possibilities one can prove that these product terms can not be combined to one complex term no matter how to reshape them.

Definition 5: Two product terms are referred as **combinable** either when these two product terms are directly combinable or if they can become directly combinable by reshaping them.

Theorem 2: If difference $(T_1, T_2) \leq 2$, terms T_1 and T_2 are combinable.

Proof: See in the Appendix.

Other cases of combinability of product terms for various values of difference and distance are discussed in the Appendix. The given combinability criteria are used in the algorithm to generate complex terms.

During the transformation from product terms to complex terms, some SOPs may be created from initial ESOPs, and vice versa. The SOP transformations similar to the above ESOP transformations have been formulated. They are not discussed here because of the lack of space.

5. THE ALGORITHM TO GENERATE COMPLEX TERMS

Based on the above discussion, the algorithm to create complex terms from product terms has been created, the pseudo-code of which is given in Figure 12.

Input: A minimized ESOPs with pt product terms.

(1) /* record the initial result as the best result */

best_result = initial result

(2) /* Search all the candidate complex terms, use rules for combinebility of product terms, record their desired orders */

For each pair of product terms T_j and T_k {
if the terms of the pair are combinable {
record the desired order of this pair.
} endif
} endfor

(3) /* Combine the desired orders which do not conflict with each other, count the number of product terms which correspond to these desired orders. */

For each pair of desired orders {
if the two desired orders are the same {
combine them to one desired order,
record the number of product terms corresponding to this desired order. }
else if there are no conflicts between the two desired orders {
combine them to one desired order which satisfies both the desired orders,
record the number of product terms corresponding to this desired order.

```
        } endif
    } endfor
```

(4) /* select the desired order which satisfies the maximum number of product terms */
best order = the first desired order in the list.

```
For each desired order {
    if (the number of product terms
    corresponding to this desired order
      >the number of product terms corre-
      sponding to the best desired order) {
    assign this order as the value of the best
    desired order.
    } endif
} endfor
```

(5) /* generate complex terms */

```
For each candidate complex term {
    according to the best order,
    convert when possible the candidate
    complex terms to the complex terms
} endfor

number_of_loops = 0
```

(6) /* record the current result which contains all the complex terms generated plus the remaining product terms. Next reshape the remaining product terms and repeat the above procedure. If a better result is obtained, take it as the best_result. */

```
iterate until number_of_loops > pt
{
    current_result = complex terms plus
    remaining product terms
    reshape current_result
    if the current_result is better than the
    best_result { number_of_loops = 0 }
    else { number_of_loops = number_
    of_loops + 1 }
    best_result = current_result
}
```

(7) /* reshape the remaining product terms */

```
For each pair of product terms in best_result {
    if the difference of the two terms ≤ 3 {
      perform exorlink operation on them,
      perform steps (2), (5) and (6).
    } endif
} endfor
```

(8) *output the results and stop the program*

FIGURE 12 Pseudo-code of the algorithm to generate complex terms

6. A DETAILED EXAMPLE

In this section, an MCNC benchmark function SQUAR5 will be used as an example to show how the product terms are factorized to complex terms. SQUAR5 has 5 input variables and 8 output variables. By first using EXORCISM-MV-2 [28] the original function is reduced to 19 product terms as shown in Table I.

On the left of the table each row corresponds to a product term, each "1" indicates a variable, each "0" indicates a negated variable, and each "—" indicates a missing variable. On the right of the table, each column presents an output function, a "1" indicates that the corresponding row is connected to this function. For instance, in column f_1, two 1's indicate that two rows, 10111 and 11---, are connected to f_1, which means

$$f_1 = a\,\bar{b}\,c\,d\,e \oplus a\,b$$

To find an optimum order of input variables, at first, each pair of product terms is checked for combinability. If they are combinable, the desired order is recorded. For instance, the first row and the fourth row are a pair of candidates, since $a\ c$ and $a\ b\ d$ can be factorized as $(b\ d \oplus c)a$. The desired order is $(b\ d\ c\ a)$. Creating desired orders is repeated until all the pairs of product terms are checked. All the desired orders are recorded. According to the algorithm from Fig. 12, the best order selected is: $(d\ b\ c\ a\ e)$.

Based on this order, the complex terms are generated (Table II).

For instance, row 1, $a\ c$ and row 4, $a\ b\ d$ can be factorized as t2 = $(d\ b \oplus c)a$. Let us observe that the complex terms t3, t4, and t5 are reverse Maitra terms, and all other complex terms generated are forward Maitra terms. There are no bidirectional terms in this example.

In this example, 19 product terms are factorized to 15 complex terms, t1, ..., t15. The intermediate result at this point has 15 rows and 13 columns. Five columns are needed for inputs and eight columns for outputs.

7. OUTPUT COLUMN FOLDING

In this section we present a new algorithm for the multiple column folding problem. This problem is similar to the gate matrix optimization problem, but with additional minimization objectives. Our input is a list of terms and associated output functions, called nets. The netlist obtained from the logic synthesis stage is already organized as a two-dimensional rectangular array, which preserves local connectivity.

TABLE I
SQUAR5 Benchmark Function

#	a	b	c	d	e	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
1	1	–	1	–	–	0	1	1	0	0	0	0	0
2	1	1	–	1	0	0	0	0	1	0	0	0	0
3	–	–	1	–	1	0	0	0	0	0	1	1	0
4	1	1	–	1	–	0	0	1	0	0	0	0	0
5	0	0	–	1	–	0	0	0	1	0	0	0	0
6	–	–	–	1	–	0	0	0	1	0	0	1	0
7	–	1	0	1	–	0	0	0	1	1	0	0	0
8	1	–	–	–	1	0	0	0	0	1	0	0	0
9	1	0	1	1	1	1	1	1	1	0	0	0	0
10	1	1	–	–	–	1	0	0	0	0	0	0	0
11	–	1	0	–	0	0	0	0	1	0	0	0	0
12	–	1	–	–	1	0	0	0	1	0	1	0	0
13	–	1	1	–	–	0	0	1	0	0	0	0	0
14	–	1	1	–	1	0	0	0	1	1	0	0	0
15	1	1	1	0	1	0	0	0	1	0	0	0	0
16	1	0	0	–	–	0	1	0	0	0	0	0	0
17	–	–	–	1	0	0	0	0	0	0	0	1	1
18	–	0	1	1	–	0	0	0	0	1	0	0	0
19	–	–	1	0	0	0	0	0	0	0	1	0	0

Each term is mapped into one row of a CMLA. Then, each term is connected to one or more output columns. The main difference between our problem formulation and traditional Gate Matrix (GM) problem formulation is, that in addition to minimizing the number of columns used for function realization we minimize also the number of logic cells used for routing. We call this problem the *GM-RCM (Gate Matrix with Routing Cell Minimization)*. Minimization of the number of columns reduces the area occupied, and minimizing the number of routing cells reduces the circuit delay.

7.1 Previous Work

Various methods have been published in the literature dealing with the GM layout problem. An exact

TABLE II
The Results of the Method

#	Row	Complex Term	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
t1	$1 \oplus 16$	$a\,c \oplus a\,\bar{b}\,\bar{c} = (\bar{b} + c)\,a$	0	1	0	0	0	0	0	0
t2	$1 \oplus 4$	$a\,c \oplus a\,b\,d = (d\,b \oplus c)\,a$	0	0	1	0	0	0	0	0
t3	$2 \oplus 6 \oplus 7$	$a\,b\,d\,\bar{e} \oplus d \oplus b\,\bar{c}\,d = (\bar{e}\,a \oplus c +)\,d$	0	0	0	1	0	0	0	0
t4	$11 \oplus 12$	$b\,\bar{c}\,\bar{e} \oplus b\,e = (e + \bar{c})b$	0	0	0	1	0	0	0	0
t5	$7 \oplus 18$	$b\,\bar{c}\,d \oplus \bar{b}\,c\,d = (c \oplus b)\,d$	0	0	0	0	1	0	0	0
t6	$8 \oplus 14$	$a\,e \oplus b\,c\,e = (b\,c \oplus a)\,e$	0	0	0	0	1	0	0	0
t7	$3 \oplus 12$	$c\,e \oplus b\,e = (b \oplus c)\,e$	0	0	0	0	0	1	0	0
t8	$3 \oplus 6 \oplus 17$	$c\,e \oplus d \oplus d\,\bar{e} = (d \oplus c)\,e$	0	0	0	0	0	0	1	0
t9	9	$d\,\bar{b}\,c\,a\,e$	1	1	1	0	0	0	0	0
t10	10	$b\,a$	1	0	0	0	0	0	0	0
t11	13	$b\,c$	0	0	1	0	0	0	0	0
t12	14	$b\,c\,e$	0	0	0	1	0	0	0	0
t13	15	$\bar{d}\,b\,c\,a\,e$	0	0	0	1	0	0	0	0
t14	19	$\bar{d}\,c\,\bar{e}$	0	0	0	0	0	1	0	0
t15	17	$d\,\bar{e}$	0	0	0	0	0	0	0	1

algorithm, for this NP-hard [29] problem, which uses dynamic programming is presented in [9]. Both, the space and time complexity of this method are exponential (space: $O(m\,2^m)$, time: $O(m^2\,2^m)$, where m is the number of gates. Therefore, a number of heuristic methods have been invented. In a paper written by Ohtsuki, et al [20], a graph-theoretical approach based on interval graphs is used. The literature published later generally uses the same problem formulation, however, solves the problem in two different ways. Ohtsuki, et al. [20] have solved this problem by generating an initial solution and than improving this solution iteratively. Wing et al. [29] generate many interval graphs heuristically, and than select the best one. Huang, et al [11] followed the same direction but in addition they considered also a layout aspect ratio. Greedy approach of assigning gates to rows one at a time was first suggested by Deo, et al. in [9]. Later, Huang, et al. [12] gave an algorithm that first selects nets and than selects and assigns gates according to the previously selected nets. In the paper written by Hu, et al. [10] this method is combined with ideas from Artificial Intelligence. The concept of *most constraint* (MC) and *least impact* (LI) are used as criteria to select nets and gates. Although various methods have been published, no method can quickly generate high quality results.

We present the algorithm which solves the column folding problem assuming the number of terms is fixed. This is analogous to the GM problem in which the number of gates is fixed. To minimize the area (a number of columns used), we try to find an optimum assignment of terms to rows of the CMLA, such that the number of overlapping nets is minimized. The different nets can be put in the same column, if they do not overlap. In addition, to decrease circuit delay, we minimize a number of logic cells used for routing, the GM-RCM problem. We choose to solve this problem by solving two subproblems separately. First we find an optimal ordering of terms that minimizes the number of cells used for routing. Next, we find an assignment of nets to columns such that no nets overlap and the number of columns is minimized.

7.2. Definitions

Definition 6: A *multi-net term* is a term connected to more than one net.
Definition 7: A *single-net term* is a term connected to only one net.
Definition 8: A *top cell* is a logic cell which is in the highest vertical position among the logic cells in the same net.
Definition 9: A *bottom cell* is a logic cell which is in the lowest vertical position among the logic cells in the same net.
Definition 10: A *used cell* is either a logic cell or a routing cell.
Definition 11: The *max-net-number* is the largest number of nets a term is connected to.
Such terms are referred to as *max-net-terms*.
Definition 12: A *row-length* is a number of used cells in a row.
Definition 13: A *max-row-length* is the row-length of the row with the largest number of cells.
Definition 14: A *column-length* is a number of cells between the top cell and the bottom cell.
Definition 15: A *max-column-length* is the largest column-length.
Definition 16: A *cost function* is a sum of column-lengths of all columns used for function implementation. In other words, the cost function is a number of cells (logic cells and routing cells) used for mapping.

7.3. Our Approach

The key idea of our algorithm for column folding is to use a global but simple approach. In previous work [11,12,29], max-net-number was used as a guide for heuristic moves. However, this number gives only a lower bound on the solution, but no information on how and if the lower bound can be achieved. Moreover, if there is a loop in the input file, and at least one term that belongs to that loop is a max-net-term, the lower bound solution is impossible. The problem of finding these loops is also quite complicated. In addition, finding all these loops can only help to estimate the lower bound, but cannot help to find the minimum solution. When the rows are permuted, the column-length of each row, as well as the max-column-length change. Therefore, in our method the max-net-number is used as a lower bound, but we use column-lengths, and especially the max-column-length as the guide for the heuristic moves.

Two important ideas are introduced in this work. The first one, which allows us to achieve very good results without exhaustive row permutations is presented in step three and the second one, which finds an optimum net assignment very efficiently is presented in step five of the algorithm description. The main steps of our algorithm are presented below.

1. *Separate the single-net terms from the multi-net terms. Minimize the multi-net terms first and add the single-net terms later.*
   It is not difficult to prove that the max-column number should not be increased if the single-net term is added to the right place into a set of minimized multi-net terms.

2. *Generate initial solution by assigning terms to rows in the descendant order of the number of nets.*
3. *Move the top rows down and the bottom rows up.* While every two rows can be permuted, some permutations increase the cost function, some decrease, and others have no effect. Our experimental results show that moving the top and bottom rows is an efficient way to find good solution while avoiding permuting all the rows exhaustively.
4. *Add the single-net terms after the multi-net terms are minimized.* If only multi-net terms are minimized, the cost function represents the cost associated with multi-net terms only. Therefore, adding the single-net terms will increase the cost function. We insert the single-net terms in each available position, one term at a time, and then select a solution with the minimum cost increase.
5. *Minimize the number of columns by combining the nonoverlapping nets.* Once all the single-net terms are added, a column combination is performed. If the bottom cell of column A is higher than the top cell of column B, or the top cell of column A is lower than the bottom cell of column B, then the columns A and B can be combined. We use the number of cells between the bottom cell at the highest position and the top cell at the lowest position as the cost indicator. If we combine the column with the minimum cost indicator, then the total number of columns is equal to the max-row-length.

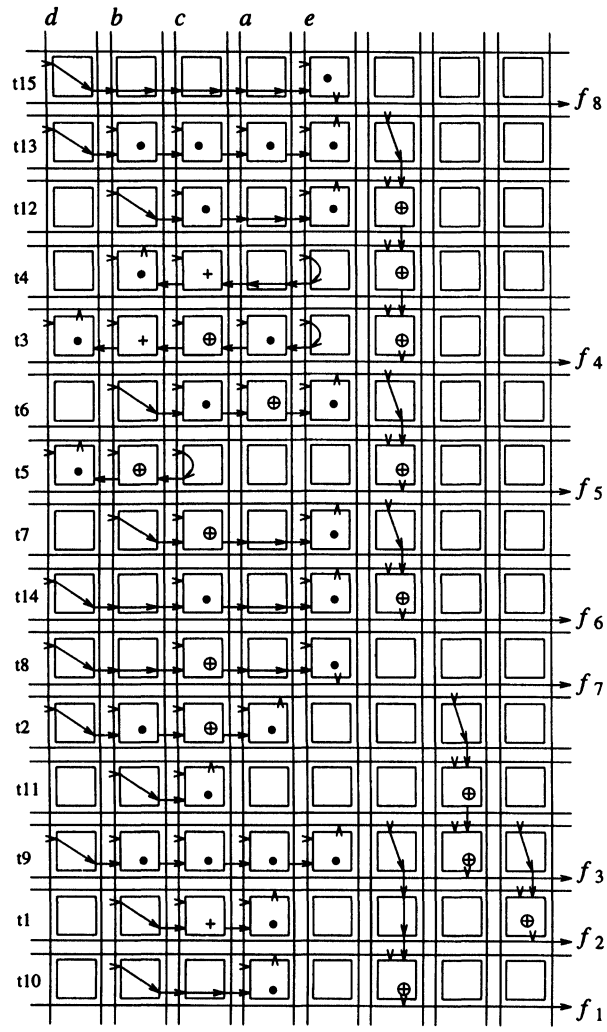

FIGURE 13 The final CMLA for the example from section 6.

7.4. The Example Continued

At the beginning of the folding stage, the problem from section 6 is represented by a "term-net" table as shown in Table III.

This table shows the same example as Table II. Here t1 is the first complex term in Table II, which is $(\bar{b} + c)\ a$.

TABLE III
The term-net table for SQUAR5 Function

Term	Net	Term	Net
t1	2	t9	1 2 3
t2	3	t10	1
t3	4	t11	3
t4	4	t12	4
t5	5	t13	4
t6	5	t14	6
t7	6	t15	8
t8	7		

The net 2 in Table III shows the relation between the complex terms and the output function f_2: since terms t1 and t9 occur in net 2, f_2 = t1 $\oplus$ t9.

From the term-net netlist presented in Table III the output functions can be reconstructed. For example, since terms t1 and t9 are the only terms that belong to net 2, f_2 = t1 $\oplus$ t9.

The final FPGA implementation of the example function is shown in Fig. 13. In the figure, symbol · indicates an AND gate, symbol + indicates an OR gate, and symbol $\oplus$ indicates an EXOR gate. Inverters are assumed available inside each cell.

8. RESULTS AND DISCUSSION

The results of the factorization method and the folding method specific to Atmel 6000 architecture are presented in [24], so here we will concentrate only on the general

TABLE IV
The Results of MINCOL

benchmark	#Tms	In.S.	Mn.S.	#L.C	#R.C
b12	28	9	5	40	16
cu	16	11	2	23	1
squar5	19	8	4	28	17
misex2	27	18	4	38	4
vg2	184	8	3	198	102
exam1	21	18	4	43	14

output column folding algorithm presented in this paper. MCNC benchmarks, in two-level PLA format, have been used to test the MINCOL (Column Minimizer) program. These benchmarks are first minimized using the EXOR minimization program EXORCISM-MV-2 [28]. The results of the MINCOL are presented in Table IV.

In the above table, #Tms and In.S. indicate the number of terms and the number of the function outputs in the input file, respectively. The number of the function outputs is equal to the number of columns in the initial solution. The minimized solution (Mn.S.) indicates the number of columns in the final solution. The number of logic cells and the number of routing cells in the final solutions are also shown in the fourth and fifth columns, respectively. For all examples the number of routing cells is smaller than the number of terms and the number of logic cells. On average, each term uses less than one cell for routing. This demonstrates one of the advantages of our approach for mapping to fine-grain FPGAs as compared to the approach currently used in the industry. For the last example, exam1 from [12], our final solution has 4 columns (4 tracks in gate matrix problem) and is the known minimum solution. However, the number of cells used for routing in our solution is 16 and is smaller than 19 cells used for routing in the best published result [12].

The decomposition of the problem into two subproblems does not influence the quality of the overall solution. The minimum solution to the first subproblem defines the lower bound for the solution of the second subproblem. And as it was mentioned previously, the solution with a number of columns equal to the lower bound of the second subproblem defined by the max-row-length can always be found with our algorithm.

The MINCOL algorithm is written in C and implemented on a SPARC workstation. The preliminary results are very encouraging for solving output column folding for the fine-grain FPGA mapping problem, GM-RCM, as well as for the general Gate Matrix problem.

9. CONCLUSIONS AND CURRENT RESEARCH

The main technical contribution of this paper is the proposition of a comprehensive design methodology for both ASIC design and (fine grain) FPGA realization. This methodology has several important advantages. It merges the stages of logic synthesis and physical design into a single stage, making use of the regularity of structure. The structure of the mapping solution is a regular array, which is good for several existing technologies. Our approach takes also advantage of the fact that two-input AND, OR or EXOR gates with subsets of negated inputs can be mixed in rows and columns of the array, creating thus the (complex) Maitra terms and the collecting columns.

Our methodology is based on a hierarchy of models. For instance here we created two design models: the "generic array" model, and the CMLA model included in it. The creation of physical design tools becomes significantly easier for the CMLA model. Since it was observed that industrial tools for fine-grain FPGAs waste about 70% of cells on empty cells and routing cells (for combinational and FSM benchmarks), the proposed approach gives very competitive results in terms of the area. It gives also especially good results on data path circuits.

The methodology proposed by us is totally new and must be thus tested on many more practical examples, together with the pre- and post-processing algorithms. Currently the most severe limitation of the method is the size of circuits that we can deal with. Especially, a fast algorithm to generate all UXF forms must be created. However, the method can be applied to parts of a circuit which was first partitioned or decomposed using general methods. It can be thus treated as a generator of large custom macro-blocks. The area of research that needs further investigation is also the comparison of the speed of the synthesized and device-mapped circuits to the solutions obtained by standard logic synthesis tools and mapped to fine-grain FPGAs using respective commercial mappers. Thanks to a recent generous donation of tools by Atmel, this research becomes now possible to us.

Although our method is particularly tuned to Atmel and Motorola architectures, we believe that the results of this paper can be also used to **create new architectures** and high-performance methods for other fine-grain FPGAs. Such architectures would be located between fine-grain FPGAs and CPLDs. For instance, the circuits obtained as above for the constrained CMLA model can be more efficiently realized in a restricted layout than in the layout corresponding directly to the generic model. Another variant of our algorithm, which applies only forward Maitra terms instead of the complex terms, would use only left local input to a cell, making thus the complex plane more compact. One can imagine also a new device architecture composed of a complex plane and an OR/EXOR plane (possible, with several such pairs of planes placed on a chip and connected by routing

channels as in CPLDs). Concluding, by imposing additional constraints on the algorithms one can easily tune them to new restricted device architectures allowing for more compact layouts.

References

[1] Algotronix Ltd., "Configurable Array Logic User Manual," Edinburgh, UK, 1991.

[2] ATMEL Corporation CMOS Integrated Circuit Data Book, 1993, 1994. 2125 O'Nel Drive, San Jose, CA, 95131.

[3] S. Bandyopadhyay, A. Pal, A. K. Choudhury, "Characterization of Unate Cascade Realizability Using Parameters," *IEEE Trans. on Comput.*, Vol. 24, No. 2, pp. 218–219, February 1975.

[4] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. R. Wang, "MIS: Multi-Level Interactive Logic Optimization System," *IEEE Trans. on CAD*, Vol. 6, no. 6, pp. 1062–1082, November 1989.

[5] Concurrent Logic, Inc., "CLi6000 Series Field Programmable Gate Array," *Rev. 1.3*, May 1992.

[6] Concurrent Logic, Inc., Seminar at PSU, November 17th, 1992.

[7] A.K. Dasari, N. Song, M. Chrzanowska-Jeske, "Layout-Driven Factorization and Fitting for Cellular-Architecture FPGAs," *Proc. Northcon '93*, pp. 106–111, Portland, October 1993.

[8] M. Davio, J. P. Deschamps, A. Thayse, "Discrete and Switching Functions," *McGraw-Hill*, 1978.

[9] N. Deo, M.S. Krishnamoorthy, and M.A. Langston, "Exact and Approximate Solutions for the Gate Matrix Layout Problem," *IEEE Trans. on CAD*, Vol. CAD-6, No. 1, pp. 79–84, January 1987.

[10] Y.H. Hu and S.J. Chen, "GM_Plan: A Gate Matrix Layout Algorithm Based on Artificial Intelligence Planning Techniques," *IEEE Trans. on CAD*, Vol. 9, No. 8, pp. 836–845, August 1990.

[11] S. Huang and O. Wing, "Gate Matrix Partitioning," *IEEE Trans. on CAD*, Vol 8., No. 7, pp. 756–767, July 1989.

[12] S. Huang, O. Wing, "Improved Gate Matrix Layout," *IEEE Trans. on CAD*, Vol. 8, No. 8, pp. 875–889, August 1989.

[13] D. K. Hwang, W. K. Fuchs, S. M. Kang, "An Efficient Approach to Gate Matrix Layout," *IEEE Trans. on CAD*, Vol. CAD-6, No. 5, pp. 802–809, September 1987.

[14] G.J. Jones, D.M. Wedgwood, "An Effective Hardware/Software Solution for Fine Grained Architectures," *Session 2, FPGA'94, 1994 ACM Second Intern. Workshop on FPGAs*, Febr. 13–15, 1994, Berkeley.

[15] U. Kebschull, E. Schubert, W. Rosenstiel, "Multilevel Logic Synthesis Based on Functional Decision Diagrams," *Proc. IEEE European Design Automation Conference*, pp. 43–47, 1992.

[16] Motorola MPA10xx Data Sheet, 1994.

[17] K. K. Maitra, "Cascaded Switching Networks of Two-Input Flexible Cells," *IRE Trans. Electron. Comput.*, Vol. EC-11, pp. 136–143, 1962.

[18] A. Mukhopadhyay, "Unate Cellular Logic," *IEEE Trans. on Comput.*, Vol. 18, No. 2, pp. 114–121, February 1969.

[19] A. Mukhopadhyay, "Cellular Logic," in "*Recent Developments in Switching Theory*," Ed. Mukhopadhyay, A., pp. 281–285, 1971, *Academic Press*.

[20] T. Ohtsuki, H. Mori, E.S. Kuh, T. Kashiwabara, and T. Fujisawa, "One-Dimensional Logic Gate Assignment and Interval Graphs," *IEEE Trans. on Circuits and Systems*, Vol. CAS-26, no. 9, pp. 675–684, September 1979.

[21] M. A. Perkowski, "A Fundamental Theorem for EXOR Circuits," *Proc. of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, September 1993, Hamburg, Germany, pp. 52–60.

[22] M. A. Perkowski, A. Sarabi, and F. R. Beyl, "XOR Canonical Forms of Switching Functions," *Proc. of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, September 1993 Hamburg, Germany, pp. 27–32.

[23] R. Rudell, A. Sangiovanni-Vincentelli, "ESPRESSO-MV: Algorithms for Multiple-valued Logic Minimization," *Proc. IEEE Custom Integrated Circuits Conf.*, 1985.

[24] A. Sarabi, N. Song, M. Chrzanowska-Jeske, and M. Perkowski "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Array", *Proc. of Design Automation Conference*, San Diego, June 1994.

[25] T. Sasao, "EXMIN2: A Simplification Algorithm for Exclusive-OR-Sum-of-Products Expressions for Multiple-Valued-Input Two-Valued-Output Functions," *IEEE Trans. on CAD*, Vol. 12, No. 5., pp. 621–632, May 1993.

[26] G. Saucier, J. Fron, and P. Abouzeid, "Lexicographical Expressions of Boolean Functions with Application to Multilevel Synthesis," *IEEE Trans. on CAD*, November 1993, pp. 1642–1654.

[27] I. Schäfer, M.A. Perkowski, H. Wu, "Multilevel Logic Synthesis for Cellular FPGAs Based on Orthogonal Expansions," *Proc. IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Sept. 1993, Hamburg, Germany, pp. 42–51.

[28] N. Song, M. A. Perkowski, "EXORCISM-MV-2: Minimization of Exclusive Sum of Products Expressions for Multiple-Valued Input Incompletely Specified Functions," *Proc. ISMVL*, pp. 132–137, Sacramento, May 24–27, 1993.

[29] O. Wing, S. Huang, R. Wang, "Gate Matrix Layout," *IEEE Trans. on CAD*, Vol. CAD-4, No. 3, pp. 220–231, July 1985.

[30] L.-F. Wu, M. A. Perkowski, "Minimization of Permuted Reed-Muller Trees for Cellular Logic Programmable Gate Arrays," In H. Gruenbacher and R, Hartenstein (eds.), "*Lecture Notes in Computer Science*", No. 705, Springer Verlag, pp. 78–87, Berlin/Heidelberg, 1993.

[31] M. Yoeli, "A Group-Theoretic Approach to Two-Rail Cascades," *IRE Trans. Electron. Comput.*, Vol. EC-14, pp. 815–822, 1965.

APPENDIX

Proof of Theorem 1

Necessity: If the distance of the two product terms is not 0, then there exists at least one literal that appears in both terms with different polarities. According to equation (4.1) from section 4.1, none of the literals can appear in both terms with different polarities. So, the distance of the two terms must be zero. If both product terms have two or more literals which appear in only one term, these two terms can be factorized to the form

$$(\dot{x}_1 \cdots \dot{x}_i) \oplus (\dot{x}_{i+1} \cdots \dot{x}_j) \dot{x}_{j+1} \cdots \dot{x}_n.$$

By definition, this is not a Maitra term.

Sufficiency: Since the order of variables is not fixed, it can be arranged like this: If a literal appears in both

terms, the corresponding variable is shifted to the right side of the term. If a literal appears only in one term, the corresponding variable is shifted to the left side of the term. If a literal is missing in both terms, the positions of their corresponding variables are arbitrary. After the arrangement, the two product terms will be either in the form of equation (4.1) or in the form of equation (4.2).•

Proof of Theorem 2

Difference (T_1, T_2) = 2

1. If distance (T_1, T_2) = 0, there are two cases.
   (a) $\hat{x}_1$ appears in one term and $\hat{x}_2$ appears in another term. In this case they are directly combinable.
   (b) $\hat{x}_1$ and $\hat{x}_2$ appear in one term and are missing in another term. In this case, since $1 \oplus a\ b = \bar{a} + \bar{b}$ these two product terms are also directly combinable.
2. If distance (T_1, T_2) = 1, assume $\hat{x}_1$ appears in both terms in different polarities and $\hat{x}_2$ appears in T_1 and is missing in T_2. Performing exorlink on T_1 and T_2, $a\ b \oplus \bar{a}$ can be reshaped to $1 \oplus a\ \bar{b}$, which corresponds to case (b) of distance 0 above. So, these two terms can be reshaped, and become directly combinable.
3. If distance (T_1, T_2) = 2, then $\hat{x}_1$ and $\hat{x}_2$ appear in both T_1 and T_2 and in different polarities. Performing exorlink on T_1 and T_2, $a\ \bar{b} \oplus \bar{a}$ can be reshaped to $a \oplus b$, which corresponds to case (a) of distance 0 above. So, these two terms can also be reshaped, and become directly combinable.

Thus, from above discussed case of Difference (T_1, T_2) = 2, and cases presented in section 4.2 it results that if difference (T_1, T_2) $\leq$ 2, these two terms are always combinable.

Other cases of Difference and Distance used in the algorithm from Fig. 12 are presented below.

Difference (T_1, T_2) = 3

Distance (T_1, T_2) = 0

There are two cases: in the first case all three literals $\hat{x}_1$, $\hat{x}_2$, and $\hat{x}_3$ appear in one term and are missing in another term. In the second case two literals appear in one term and one literal appears in another term. For instance:

1. Since $1 \oplus a\ b\ c = \bar{a} + \bar{b} + \bar{c}$, terms 1 and $a\ b\ c$ are combinable.
2. Two terms $a\ b \oplus c$ are directly combinable.

Concluding, if difference (T_1, T_2) = 3, and distance (T_1, T_2) = 0, terms T_1 and T_2 are directly combinable.

Distance (T_1, T_2) = 1

Assume $\hat{x}_3$ appears in both terms in different polarities.

1. $\hat{x}_1$ and $\hat{x}_2$ appear in one term and are missing in another term. Since these two terms are disjoint, the operator $\oplus$ can be changed to +, then a consensus operation can be performed: $a\ b\ c \oplus \bar{c} = a\ b\ c + \bar{c} = a\ b + \bar{c}$. In this case, these two terms are combinable.
2. $\hat{x}_1$ appears in one term and $\hat{x}_2$ appears in another term. In this case, trying all the possibilities shows that these two terms are not combinable.

Distance (T_1, T_2) $\geq$ 2

In this case, trying all the possibilities shows that these two product term are not combinable.

Difference (T_1, T_2) > 3

(3) If distance (T_1, T_2) = 0, and the two terms can be arranged to the form of equation (4.2), these two terms are combinable.

(4) If distance (T_1, T_2) = 1, and the two terms can be arranged to the form of equation (4.1), these two terms are combinable.

Biographies

NING SONG received the M.S. degree in management science and computer science from Shanghai Jiaotong University, China in 1983 and the M.S. degree in electrical engineering from Portland State University, Portland, Oregon in 1992.

He is currently with Lattice Semiconductor Corporation, Milpitas, CA. He is also working towards his Ph.D. degree in electrical engineering at Portland State University. His research interests are in the logic synthesis, technology mapping and exclusive- or minimization.

MAREK A. PERKOWSKI received the M.S. degree in electronics in 1970 and the Ph.D. degree in automatics (digital systems) in 1980 from Technical University of Warsaw. He was an Assistant Professor at Technical University of Warsaw from 1980 to 1981, a Visiting Assistant Professor at the University of Minnesota from 1981 to 1983, and since 1983 he has been at Portland State University where he is a Professor of Electrical and Computer Engineering. He is the co-author of three books, seven book chapters, and over 120 technical articles in design automation, computer architecture, artificial intelligence, image processing and robotics.

MALGORZATA CHRZANOWSKA-JESKE received the M.Sc. degree in electronic engineering from the Technical University of Warsaw in 1972, and the Ph.D. degree in electrical engineering from Auburn University in 1988. In 1972, she joined the Electronic Engineering Faculty at the Technical University of Warsaw. In 1977, she became a Research Staff Member of the Research and Production Center of Semiconductor Devices, Warsaw, Poland. Since 1989, she has been an Assistant Professor in the Department of Electrical Engineering at Portland State University. She has published more than 30 technical papers and articles. Her research interests are in computer-aided-design of integrated circuits, device simulation and low-temperature electronics.

ANDISHEH SARABI received the M.S. degree in Mathematical Sciences in 1991 and the Ph.D. degree in electrical and computer engineering in 1994 both from Portland State University. He has been with the CAD group at Portland State University since 1989. His research interests are in logic synthesis, cellular FPGAs, XOR logic, and testing.