A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays

Andisheh Sarabi, Ning Song,

Malgorzata Chrzanowska-Jeske, and Marek A. Perkowski

Portland State University, Department of Electrical Engineering
P.O. Box 751, Portland, OR, 97207, tel. (503) 725-5411

ABSTRACT

This paper introduces a new design approach that combines logic and layout synthesis for Cellular-Architecture (CA) FPGAs. The comprehensive design method starts from a Boolean function, specified as SOP or ESOP, and produces a rectangularly-shaped multi-level structure of (mostly) locally connected cells. This two-dimensional array of logic cells is well suited for CA-type FPGA realization. Two stages: restricted factorization and technology folding are discussed in more details. The architecture constraints and the implementation are presented for ATMEL6000 series architecture.

1. INTRODUCTION

In recent years programmable logic devices are being used more frequently as the alternatives to the traditional ASIC technologies. They can be classified accordingly to the basic Boolean primitives, the granularity of these primitives, the routing domain architecture, and the programming method they use. Intensive studies have been undertaken to develop the technology mapping and layout synthesis methods for the two most popular categories of these devices, namely look-up-table based (LUT-based) and row-based FPGAs. The architecture-specific technology mapping approaches were developed, and the placement and routing techniques were mostly adopted from the semi-custom design styles like standard cells and gate arrays, with some necessary modifications. Other categories of FPGAs have not attracted so far much attention of the research community. This paper focuses on logic and layout synthesis of the other category of FPGAs, mainly Cellular-Architecture (CA) FPGAs. These devices are characterized by the local connectivity between logic blocks placed as a symmetrical array. Logic blocks are usually of the standard-cell type with a limited number of inputs and outputs. These FPGAs are called by some sea-of gate or fine-grain type, however in this paper, we will refer to them as CA-type FPGA, as we believe that this name represents more accurately the main feature that distinguishes this group from the others; local connectivity.

Different approach to the logic and layout synthesis for these type devices is needed to efficiently utilize their potential and we believe that the methods of separate technology mapping, placement and routing, used for other FPGAs, become of little value. In the traditional approach a large number of logic cells is used for wiring connections or left unused at all. This problem is mainly caused by not preserving local connectivity during the synthesis steps. Frequently, local buses need to be 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. In the "macro block" approach which is currently used in the industry [1], 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 local connectivity between macros is not preserved during the subsequent placement. 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 CA-type FPGAs is used for the routing and non-programmed cells [1],[2].

Recently, several logic synthesis approaches applicable to CA-type FPGAs have been presented [13], [15]. The first research on applying variable ordering in factorization is reported in [12]. The approaches based on trees and decision diagrams [6] have also been reported [13], [15]. 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. In a search for more efficient approach, new spectral methods based on orthogonal expansions and Universal XOR Forms [10] were introduced recently. These methods as well as the algebraic restricted factorization method presented in this paper are based on the ideas from the classical cellular arrays. While the spectral methods are more general and usually lead to a better solution, the algebraic one introduced in this paper leads to much more efficient algorithms.

Cellular arrays were studied extensively during sixties and seventies [7], [8], [9]. However, the connectivity patterns of cells were too restricted and the buses were mostly absent. Because of these limitations the number of cells grows rapidly, often exponentially. Therefore, the classical cellular arrays were never commercialized, and the PLD and FPGA technologies were developed with no reference to them. New CA-type FPGAs introduce more powerful logic blocks, more local connections and local and global busses for improved communication over longer distances.

Therefore, we propose here a totally new approach to combined logic synthesis and physical design. 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) 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). The name "Maitra term" comes from "Maitra cascades" [7]. The Maitra terms are collected in output_plane composed of the two-dimensional array with OR or EXOR gates. This two-dimensional logic array will be called the *"Complex Maitra Logic Array"* (CMLA). The CMLA concept, shown in Fig.1, is a powerful generalization of PLAs. In addition, similar to PLAs and gate matrix layout, our CMLAs can be folded in many ways. Unfortunately, our problem is more complex, and all well-known algorithms for PLA folding and gate-matrix problems [3,4,5] can only be used to solve some parts of the technology folding problem. The paper illustrates our general approach on ATMEL6000 Series of FPGAs as the target architecture. The approach, however, can be adopted to other CA-type FPGAs, such as Motorola, Algotronix or Pilkington.

The paper is organized as follows. In Section 2 we present our general approach and define Maitra terms. Section 3 formally introduces the mathematical apparatus necessary to create the complex terms and gives the description of the algorithm. The solution method to the technology folding problem is discussed in Section 4. In Section 5 the detailed example is presented. Results and conclusions are given in Section 6.

2. OUR APPROACH

The comprehensive approach to the logic and layout synthesis for CA-type FPGAs includes two stages:

1. *Logic optimization* 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. *Technology-folding* which maps CMLA representation of the function to the target architecture, such that the area of the layout is minimized.

Each of the above stages can be solved in several ways. In this paper, we illustrate the logic synthesis stage with the *restricted factorization* method, which is a general approach not associated with any specific CA-type FPGA. The second, technology folding stage consists of two parts: folding of the complex_plane (input), and folding of the collecting_plane (output). Technology dependent constraints of ATMEL FPGA architecture are used to illustrate the complex_plane folding. Output_plane folding is more general and does not depend that much on the particular chip architecture as only OR and EXOR gates are used, and will not be discussed here.

A generic model of the CA-type FPGA consists of the array of logic blocks each connected with its *s* neighbours through input and output connections. Each logic block can use up to *n* inputs from the neighbours and produce up to *m* outputs to its neighbours. In addition, *k* inputs from local buses, vertical and horizontal, can be used by each logic block. Outputs can be also assigned to *l* local buses, vertical and horizontal. The logic block can realize some set of logic functions. Since this method can be really effective for small granularity blocks, logic blocks should be kept rather small. For the purpose of simplicity we assume that they are limited to a finite set of simple logic gates, which includes an EXOR gate. A number of inputs and outputs should also be small. The schematic picture of such architecture is shown in Fig.2.

2.1. Maitra terms and Complex terms

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.*

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.

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

Each of the following expressions represents a forward Maitra term: $(a\ \bar{b}) + c$, $(a + b)\,c$, $(a \oplus b) + \bar{c}$, $((c\ \bar{b}) + a) \oplus d$. And, each of the following expressions represents a reverse Maitra term: $c + (a\ \bar{b})$, $c\ (a + b)$. The expression $((a\ b) + \bar{b})\,c$ is not a Maitra term because the literal b appears twice. Similarly, $a + (b\ \bar{c}) + d$ 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.

M1 ⊕ M2 = {(ab) + c} ⊕ {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.

These examples show 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. 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 example of CMLA is shown in Fig.3.

3. RESTRICTED FACTORIZATION THEORY

The new method called *restricted factorization* based on *cube calculus operations* [11], [14] is described. 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.*

3.1. 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 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 1. 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 variables a and c have 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* [14] of terms T_1 and T_2 is defined by the following formula:

$$T_1 \otimes T_2 = \bigoplus \{ \hat{x_1} \cdots \hat{x_{i-1}} (\hat{x_i} \oplus \hat{y_i}) \hat{y_{i+1}} \cdots \hat{y_n} \mid \text{for such } i = 1, ..., n, \text{ that } \hat{x_i} \neq \hat{y_i} \}$$

Example 2. Given two product terms $a\ b\ e$ and $a\ \bar{b}\ c\ d\ e$. The exorlink of these two terms is

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

An exorlink operation generates a set of resultant product terms. It is explained in [14]. 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 *directly combinable,* if these two product terms are in one of the following forms,

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

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

In equation (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 3. $a\ b\ d\ e \oplus c\ d\ e = (a\ b \oplus c)\ d\ e$.

In equation (2), the two product terms can be combined to

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

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

Example 4. $a\ \bar{b}\ c\ d\ e \oplus d\ e = (a\ \bar{b}\ c \oplus 1)\ d\ e = (\bar{a} + b + \bar{c})\ d\ e$, the two product terms are directly combinable.

3.2. Checking if two terms are combinable.

The criteria for combining product terms are based on calculating the difference, the distance 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.

Example 5. 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 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 $T_1 = \hat{x}_1 \hat{x}_2 \cdots \hat{x}_n$ and $T_2 = \hat{x}_1 \hat{x}_2 \cdots \hat{x}_n$ 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.

Case when 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.

Case when difference(T_1, T_2) = 1.

(1) If distance(T_1, T_2) = 0, then $\hat{x_1}$ appears 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.

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) ≤ 2, terms T_1 and T_2 are combinable.

Other cases of combinability of product terms for various values of difference and distance are taken into account in the algorithm to generate complex terms which are not discussed here. 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.

The basic steps of the algorithm to create complex terms from product terms are given below. Combinability graph is a compatibility graph, with the compatibility relation substituted with combinability relations.

1. For each pair of product terms T_i and T_j if the terms are combinable, record all possible variable orderings for the pair;
2. Build the adjacency matrix for the combinability graph;
3. Create a priority list of complex terms in the decreasing order of the number of adjacents, and adjacents of their adjacents in the combinability graph.
4. The order of the variables in the maximum clique which does not violate all possible combinabilities in the clique is chosen as the ordering of input variables.
5. For each group of product terms in the priority list, with the chosen order of input variables, generate the complex terms
6. Add all terms not included in the complex terms generated in step 5.

4. TECHNOLOGY FOLDING

Once an optimized set of complex terms has been identified for a given multi-output function f, folding techniques are used to even more economically utilize the FPGA cells. To minimize the area, a proper matching of complex terms is found such that the number of rows occupied by complex terms is minimized. These folding techniques depend on the specific architecture of the FPGA which could allow different compatible terms to be placed on the same row. This problem may look similar to the well known problem of multiple column folding [3], however, the basic difference is that we do not change the order of variables, which is fixed after complex term generation, but we choose the best combination of complex terms, which can be assigned to the same row due to architecture constraints.

4.1. Folding for ATMEL6000 Series

The possible gates that can be utilized in ATMEL6000 are an inverter, an AND, OR, EXOR, NAND gate, a wire and their combinations [1]. A number of available inputs is limited to three and outputs to two. Only one input can be taken from the local bus, and a number of local busses is limited to four. Each cell has connections to four neighbors. Based on the above architecture limitation the number of complex terms which can be folded into one row is six. The complex terms can be accessed from the left-most cell, the right-most cell, and the two local busses. Because each cell has two outputs, denoted by A and B, which are available for all neighbours the total number of complex terms which can be accessed in one row is six. We assume the following signal assignments. The vertical local busses in the complex_plane carry both input variables and

their negations for each column. The cell personalized to the OR gate uses only output of type A.

AB Parallelism

The conditions for placing two complex terms in one row of ATMEL using the "A" and "B" inputs and outputs going horizontally through the row are defined. The input variables are assigned to the vertical busses in a given order. We assume that the order increases from left to right. Therefore, the left-most input variable will be in lowest position. Moreover, the instance of a literal in a complex term with the lowest order will be referred to as the *initial* literal and will be denoted by C^i where C is some term. The literal with the highest order will be referred to as the *last* literal and will be denoted by C^l. The set of all literals appearing in a term j will be referred to as the *literal_set* and will be denoted by SL_j.

A complex term can be a product, a sum, an exclusive-sum or a combination of any of the above. A monoterm, which is a product of literals, will be referred to as a *product_line* and will be denoted by P. A monoterm which is only comprised of summation of literals will be referred to as a *sum_line* and will be denoted by S. A monoterm which is only comprised of exclusive summation of literals will be referred to as a *exsum_line* and will be denoted by X. A term which is comprised of sums and products of literals will be referred to as a *sum_product_line* and will be denoted by C.

The *literal_distance*, denoted by d, between two literals is the difference between their indices. As an example the distance between b and c, if placed in alphabetical order is 1. A *continous term* (denoted by *Q)* is one that has a continuous literal set, i.e. for any literal besides the initial and final, there exists two literals with distance 1 in the literal set (for initial and final literal there exists one literal with this property).

Two terms C_i and C_j, with $SL_i \cap SL_j \neq 0$, are AB-Parallel if they can be placed in the same row of the CA-type FPGA, and will be denoted by $<C_i, C_j>$ where C_i and C_j are the two terms. Possible matching types are given below. In the examples we have assumed that the variables are assigned in the alphabetical order.

TYPE_1. $<P_1, P_2>$ such that $SL_1 \bigcup SL_2 = SL \in Q$ and $SL_1 \bigcap SL_2 = P_1^i$.
e.g. $<P1, P2> = <a, abcde>$; $<ae, abcd>$; $<ade, abc>$; where $SL = \{a, b, c, d, e\}$.

TYPE_2. $<S, P>$ such that $SL_P \in SL_S$, $SL_P \in Q$, and $S^l \in SL_P$.
e.g. $<S, P> = <a+b+c+d, abcd>$; $<a+b+c+d, bcd>$; $<a+c+d, cd>$; $<a+b+c+d, d>$.

TYPE_3. $<P, S>$ such that $SL_S \in SL_P$, and $SL_P \in Q$.
e.g. $<S, P> = <abcd, a+b+c+d>$; $<abcd, b+c+d>$; $<abcd, b+d>$; $<abcd, a+c>$.

TYPE_4. $<C, S>$ such that $d(C_S{}^l, C_P{}^i) \geq 1$, $C_S \in Q$, $C_P \in Q$ and $SL_S \in SL_{C_P}$.
e.g. $<C, S> = <(a+b+c)defg, d+f>$.

TYPE_5. $<C_1, C_2>$ such that $SL_{S1} = SL_{S2}$, $SL_{P1} \bigcup SL_{P2} \in Q$, $SL_{P1} \bigcap SL_{P2} = SL_P{}^i$ and $d(SL_{S1}{}^l, SL_{P1}{}^i) \geq 2$. Where SL_{P1} denotes the product part of the first term. (Sum_line can be substituted with exsum_line.)
e.g. $<C_1, C_2> = <(a+b+c)d, (a+b+c)dfg>$.
e.g. $<C_1, C_2> = <(a \oplus b \oplus c)d, (a \oplus b \oplus c)dfg>$.

TYPE_6. $<C_1, C_2>$ such that $SL_{S1} = SL_{S2}$, $SL_{P1} \bigcup SL_{P2} \in Q$ and $C_S{}^l < (C_{P1}{}^i - 1)$ and $C_S{}^l < C_{P2}{}^i$. (Sum_line can be substituted with exsum_line.)
e.g. $<C1, C2> = <(a+b+c)ef, (a+b+c)gh>$.
e.g. $<C1, C2> = <(a \oplus b \oplus c)ef, (a \oplus b \oplus c)gh>$.

L-Placeability

L-Placeability refers to the terms that can have their outputs put on a local bus L. The following are the possible cases of L-Placeability:

1. The AB-Parallel terms of TYPE_1.
2. The sum_line part of AB-Parallel terms of TYPE_2.
3. A single primary input.

The terms that can be placed in the same row will be referred to as being row compatible or R-compatible. Some additional restrictions are introduced if multioutput functions are considered.

Row Compatibility

1. Two AB-Parallel terms are R-compatible. If the terms belong to more than one output, they have to be L-placeable in order to be R-compatible.
2. An L-placeable set of TYPE_1 and another AB-Parallel set are R-compatible if the distance between the last literal of the first set and the first literal of the second set is at least two.
e.g. $<ad, abc>$ and $<f+g+h+i, fghi>$.
3. An L-placeable term of TYPE_2 and AB-Parallel set of TYPE_5 are R-compatible if the sum_lines (exsum) of all three terms are identical.
e.g. $<a+b+c>$ and $<(a+b+c)e, (a+b+c)efg>$.
4. An L-placeable term of TYPE_2 and a sum_line (exsum) which includes the L-placeable term as lower order literals and its higher order literals start with a distance of two from last literal of the L-placeable term and are continuous, are R-compatible. $<S_1, S_2>$ such that $SL_1 \in SL_2$, $d(SL_1{}^l, SL\ (S_2 - S_1)^i) \geq 2$, $S_2 - S_1 \in Q$. Where "–" denotes the difference of the literal sets; i.e. all literals which are in the first set and not in the other.
e.g. $<a+c+d>$ and $<a+c+d+f+g+h>$.
5. An L-placeable term of TYPE_2 and an AB-Parallel set of terms of TYPE_1 are R-compatible if the distance between the last literal of the sum-line (exsum) and the first literal of the AB-Parallel set of terms is at least two.
e.g. $<a+b+c>$ and $<e, eg>$.
6. An L-placeable term of TYPE_2 and a sum-product-line which includes the L-placeable term as summation part and its product literals start with a distance of two (≥ 1 for exsummation) from the last literal of the summation part and are continuous, are R-compatible. $<S, C>$ such that $C_S = S$, $d(SL_S{}^l, SL_P{}^i) \geq 2$, $C_P \in Q$.
e.g. $<a+c+d>$ and $<(a+c+d)fgh>$.
7. All R-compatibilities involving L-placeable term of TYPE_2 are applicable to L-placeable terms of TYPE_3. The difference is that the sum_lines in TYPE_2 terms are replaced by a single variable.

For multi-output functions, it is assumed that the complex terms which need to be EXORed in the collecting_plane are all on horizontal local busses. The possible row folding is shown in Fig.4.

The basic steps of our approach to the technology folding are presented below.

1. Identify terms which are R-compatible and these which are non R-compatible.
2. Construct the compatibility graph CG(V,E).
3. Find an optimum clique covering, with clique sizes smaller or equal to the maximum number of terms which can be placed in the same row (here 6).
4. Start with vertices with the smallest vertex degree, which belong to the cliques of the largest permitted sizes. Assign them to one row.
5. Assign all remaining vertices, which are connected with at least one edge to the graph.
6. Assign all remaining unconnected nodes to different rows.

As it can be observed, the method is general while the type of R-compatibilities and the maximum size of the clique varies with different architectures.

5. A DETAILED EXAMPLE

An MCNC benchmark function SQUAR5 is used 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 [14] the original function is reduced to 19 product terms as shown in Table I. 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. 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, ..., t 15. The intermediate result at this point has 15 rows and 13 columns. Five columns are needed for inputs and eight columns for outputs.

6. RESULTS AND CONCLUSIONS

The main technical contribution of this paper is the proposition of a comprehensive design methodology for CA-type FPGAs. This methodology has several important advantages. It merges the stages of logic synthesis and layout synthesis 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.

The results of the evaluation of our approach to technology folding on a set of benchmark functions are presented in Table III. The number of input variables is shown in column 2, and the number of outputs in column 3. In column 4 a number of column_terms, which are present in the ESOP function representation before folding, and in column 5 after folding are shown, respectively. The average improvement on this set of benchmarks is 17%. It can be noticed that for some functions there was no improvement and for some the improvement was marginal. But for majority of the tested examples the improvement was significant and as high as 39%. These results can still be improved once some of the new ideas will be incorporated.

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. 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 CMLA concept is well suited for both fine-grain Field Programmable Gate Arrays (FPGAs) and ASIC design. Although the algorithm presented in this paper is particularly tuned to Atmel architecture, the results of this paper can be also used for other CA-type FPGAs, for example the Motorola chip.

benchmark	#in	#out	non-fd	folded
5xp1	7	10	33	25
add6	12	7	128	122
adr2	4	3	8	4
cadr4	8	5	32	25
cmlp4	8	8	62	50
cnrm4	8	5	70	62
f51m	8	8	32	28
mlp3	6	6	19	13
rd53	5	3	15	12
rd73	4	3	39	34
sao2	10	4	29	27
sqr6	6	12	35	19
squar5	5	8	19	12
t481	16	1	14	10
Total			535	443

TABLE III. Benchmark Examples

REFERENCES

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

2. Concurrent Logic, Inc., Seminar at PSU, Nov. 17th, 1992.

3. D.F. Wong, H.W. Leong, C.L. Liu, "Multiple PLA folding using the method of simulated annealing," *Proc. Custom Integrated Circuit Conf. May 1986.*

4. 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.

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

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

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

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

9. A. Mukhopadhyay, "Cellular Logic," in *"Recent Developments in Switching Theory,"* Ed. Mukhopadhyay, A., pp. 281-285, 1971, *Academic Press.*

10. 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.

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

12. 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.

13. 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.

14. 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.

15. 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.