Communication Based FPGA Synthesis for Multi-Output Boolean Functions

Christoph Scholl

Department of Computer Science
Universität des Saarlandes
D 66041 Saarbrücken, FRG
Tel: ++49 681 302-2274
Fax: ++49 681 302-4421
e-mail: scholl@cs.uni-sb.de

Paul Molitor

Department of Computer Science
Martin-Luther Universität Halle
D-06099 Halle (Saale), FRG
Tel: ++49 345 622 529
Fax: ++49 345 622 514
e-mail: molitor@informatik.uni-halle.de

Abstract— One of the crucial problems multi-level logic synthesis techniques for multi-output boolean functions $f = (f_1, \ldots, f_m) : \{0,1\}^n \to \{0,1\}^m$ have to deal with is finding sublogic which can be shared by different outputs, i.e., finding boolean functions $\alpha = (\alpha_1, \ldots, \alpha_h) : \{0,1\}^p \to \{0,1\}^h$ which can be used as common sublogic of good realizations of $f_1, \ldots, f_m$.

In this paper we present an efficient ROBDD based implementation of this COMMON DECOMPOSITION FUNCTIONS PROBLEM (CDF).

Formally, CDF is defined as follows: Given m boolean functions $f_1, \ldots, f_m : \{0,1\}^n \to \{0,1\}$, and two natural numbers p and h, find h boolean functions $\alpha_1, \ldots, \alpha_h : \{0,1\}^p \to \{0,1\}$ such that $\forall 1 \le k \le m$ there is a decomposition of f_k of the form

$$f_k(x_1, \ldots, x_n) = g^{(k)}(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_h(x_1, \ldots, x_p), \alpha^{(k)}_{h+1}(x_1, \ldots, x_p), \ldots, \alpha^{(k)}_{r_k}(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n)$$

using a minimal number r_k of single-output boolean decomposition functions.

Experimental results applying the method to FPGA synthesis are promising.

I Introduction

The long term goal for logic synthesis is the automatic transformation from a behavioral description of a boolean function to near-optimal netlists, whether the goal is minimum delay, minimum area, or some combination. Most of the approaches attacking the multi-level logic synthesis problem use gate count as optimization criterion. A survey can be found in [4]. Alternatively, some recent papers [9, 10, 12, 21, 17, 23] propose an approach different from the one addressed above. This approach to multi-level logic synthesis which originates from Ashenhurst [1], Curtis [8], and Karp [11] is based on minimizing communication complexity. The methods used to reduce communication complexity employ functional decomposition, i.e., given a boolean function $f : \{0,1\}^n \to \{0,1\}$ they are looking for boolean functions α and g, such that $f(x_1, \ldots, x_n) = g(\alpha(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n)$ holds for all $(x_1, \ldots, x_n) \in \{0,1\}^n$ (see Figure 1). $\alpha = (\alpha_1, \ldots, \alpha_r) : \{0,1\}^p \to \{0,1\}^r$ is a multi-output boolean function. The goal is to find decompositions where the number r of decomposition functions (i.e. the number of wires between block α and block g) is minimal. (If $r < p$, then the decomposition is called non-trivial.)

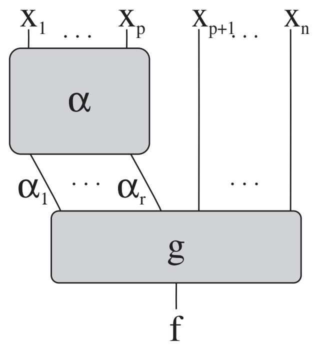

Fig. 1. Decomposition of a boolean function $f : \{0,1\}^n \to \{0,1\}$

This approach to reduce communication complexity is especially suited for FPGA synthesis, where the logic is mapped onto blocks of look-up tables, which can realize arbitrary functions $lut : \{0,1\}^b \to \{0,1\}$ up to a certain number b of inputs* (see [12, 13, 14, 15, 19]).

A fundamental step in logic synthesis is the identification of common sublogic. Methods to identify common sublogic by (algebraic or boolean) division were developed by Brayton et al. and included in the SIS package [22].

In this paper we present a method to identify common sublogic for the case that boolean functions are realized by decomposition. This method comes into play when we have to process multi-output boolean functions $f = (f_1, \ldots, f_m) : \{0,1\}^n \to \{0,1\}^m$. Note that even if the original function f has only 1 output ($m = 1$), in most cases we need a generalization to multi-output boolean functions when we apply functional decomposition recursively to α and g.

Of course, techniques to decompose single-output boolean functions can be generalized to multi-output boolean functions if we consider multi-output boolean functions as single-output multi-value functions $f' : \{0,1\}^n \to \{0, \ldots, 2^m - 1\}$ defined by $f'(x_1, \ldots, x_n) = \sum_{i=1}^m f_i(x_1, \ldots, x_n) \cdot 2^{i-1}$ (see [15]). However, in most cases – unless m is very small – we have the problem, that in such decompositions the function g must have as many inputs as f (or at least nearly as many), such that g is not much easier to synthesize than f.

*For XC3000 device, e.g., b = 5

On the other hand, if we would decompose each single-output boolean function f_i independently of the other single-output functions f_j $(j \neq i)$ and would test after that, whether they use some identical decomposition functions by chance, we wouldn't make use of the potential of reusing subcircuits for different outputs of f.

For these reasons we do decompose all f_i as single-output functions, but we make use of our freedom in the choice of the decomposition functions to compute such decomposition functions which can be used in the decomposition of as many f_i as possible. In the same way as in [17], the method is divided into two steps: In the first step, output partitioning is performed, i.e., $\{f_1, \dots, f_m\}$ is partitioned into disjoint sets $Y_1, \dots, Y_u$. Single-output functions f_i and f_j of the same set Y_k will be decomposed with respect to the same input partition. The partitioning is executed such that for every Y_k there is an input partition which is 'near-optimal' for every $f_i \in Y_k$. In the second step, the decomposition functions of the single-output functions of each class Y_k are constructed giving special attention to generate these functions in such a way that many of them can be used in the decomposition of different elements of Y_k.

Unlike [14] we avoid to compute the set of all possible decomposition functions for all f_i to choose common decomposition functions of the functions f_i from these sets, because the number of possible decomposition functions for such a single-output function f_i is huge: If we need r_i different decomposition functions $\alpha_1^{(i)}, \dots, \alpha_{r_i}^{(i)}$ for the decomposition of f_i, the decomposition functions are chosen from a set of at least $\binom{2^{r_i}}{2^{r_i}-1} = \theta(2^{(2^{r_i} - \frac{r_i}{2})})$ possible decomposition functions. In this paper we present an algorithm which directly computes a maximum number of common decomposition functions of $f_1, \dots, f_m$.

In contrast to [17], which was based on function tables and decomposition charts, we efficiently make use of Reduced Ordered Binary Decision Diagrams (ROBDD) during the computation of common decomposition functions. ROBDD [6] are compact representations for many of the boolean functions encountered in typical applications. In this paper we show that it is possible to carry out all necessary steps based on ROBDD's. This increases the efficiency of the approach in a high degree. In particular, we show that the computation of *common decomposition functions* for the decomposition of several single-output functions can be performed efficiently based on ROBDD's.

Benchmarking results show the new method to be efficient with respect to layout size, signal delay and running time. Experiments concerning FPGA synthesis using these methods are very promising as well.

We start by giving some basic definitions (section II) and summarize the algorithm for computing common decomposition functions (section III). In section IV we demonstrate how to apply ROBDDs to implement this algorithm. Experimental results close the paper (section V).

II Basic definitions

A multi-output boolean function ϕ with n inputs is represented as a set $\{\phi_1, \dots, \phi_m\}$ of boolean-valued output functions. We denote the set of completely defined boolean functions with n inputs and m outputs by $B_{n,m}$. Let B_n be an abbreviation for $B_{n,1}$. $\phi_{i,\dots,j}$ $(i \le j)$ denotes the tuple $(\phi_i, \dots, \phi_j)$.

Definition 1 *A* decomposition of a multi-output boolean function $f \in B_{n,m}$ *with respect to the input partition* $\{X_1, X_2\}$ $(X_1 = \{x_1, \dots, x_p\}, X_2 = \{x_{p+1}, \dots, x_{p+q}\}$, $p+q=n)$ *is a representation of f of the form*

$$f_k(x_1, \dots, x_n) \quad = \quad g^{(k)}(\alpha_1^{(k)}(X_1), \dots, \alpha_{r'_k}^{(k)}(X_1), X_2)$$

$(\forall k \in \{1, \dots, m\})$, *where* $\alpha_i^{(k)} \in B_p$ $(\forall i)$, *and* $g^{(k)} \in B_{r'_k+q}$. $\alpha_i^{(k)}$ *are called* decomposition functions of f_k. $g^{(k)}$ *is called* composition function of f_k. ◇

With respect to a given input partition $\{X_1, X_2\}$, a single-output function f_k can be represented as a $2^p \times 2^q$ matrix $M(f_k)$, the *decomposition matrix of f_k* or the *chart of f_k* with respect to $\{X_1, X_2\}$. (For illustration see Figure 2.) Each row and column of $M(f_k)$ is associated with a distinct assignment of values to the inputs in X_1 and X_2, respectively, such that $f_k(X_1, X_2) = M(f_k)[X_1, X_2]$ where $M(f_k)[X_1, X_2]$ represents the element of $M(f_k)$ which lies in the row associated with X_1 and the column associated with X_2.

Note that $(\alpha_1^{(k)}, \dots, \alpha_{r'_k}^{(k)})$ of definition 1 encodes the rows of chart $M(f_k)$. Of course, the following property has to hold.

Encoding Property If the row pattern of row $(v_1, \dots, v_p) \in \{0,1\}^p$ differs from the row pattern of row $(v'_1, \dots, v'_p) \in \{0,1\}^p$, then $(\alpha_1^{(k)}, \dots, \alpha_{r'_k}^{(k)})$ has to assign different codes to $(v_1, \dots, v_p)$ and $(v'_1, \dots, v'_p)$. ◇

The minimum number of communication wires required between the subcircuit which encodes the rows of $M(f_k)$ and the composition function $g^{(k)}$ is $\lceil \log dr^{(k)} \rceil$ where $dr^{(k)}$ is the number of distinct row patterns in $M(f_k)$. r_k will denote value $\lceil \log dr^{(k)} \rceil$ in the following.

Definition 2 *A decomposition of* $f_k : \{0,1\}^n \rightarrow \{0,1\}$ *is* optimal (for X_1 with respect to a given input partition $A = \{X_1, X_2\}$) *if it uses only* r_k $(= \lceil \log dr^{(k)} \rceil)$ *decomposition functions* $\alpha_1^{(k)}, \dots, \alpha_{r_k}^{(k)}$ *with domain* X_1. ◇

In the following we will always use such "optimal" decompositions.

To compute decomposition functions (with domain X_1) of a multi-output function f which are used by different single-output functions f_k, we have to consider the following problem which will be denoted by CDF.

Given: Let $f = \{f_1, \dots, f_m\} \in B_{n,m}$ be a multi-output boolean function, $A = \{X_1, X_2\}$ with $X_1 = \{x_1, \dots, x_p\}$ and $X_2 = \{x_{p+1}, \dots, x_n\}$ be an input partition, and h be a natural number with $h \le r_k$ $(= \lceil \log dr^{(k)} \rceil)$ $(\forall k)$.

Find: h single-output boolean functions $\alpha_1, \dots, \alpha_h \in B_p$, which can be used as decomposition functions of every single-output function f_k for $k = 1, \dots, m$ such that there is an optimal decomposition of f_k of the form

$$f_k(x_1, \dots, x_n) \quad = \quad g^{(k)}(\alpha_1(X_1), \dots, \alpha_h(X_1), \alpha_{h+1}^{(k)}(X_1), \dots, \alpha_{r_k}^{(k)}(X_1), X_2).$$

f_1

$x_1x_2x_3$ \ x_4, x_5, …, x_n	00, 00, …, 01	…	1, 1, …, 1
0 0 0	row pattern 1		
0 0 1	row pattern 1		
0 1 0	row pattern 2		
0 1 1	row pattern 2		
1 0 0	row pattern 3		
1 0 1	row pattern 3		
1 1 0	row pattern 3		
1 1 1	row pattern 2		

f_2

$x_1x_2x_3$ \ x_4, x_5, …, x_n	00, 00, …, 01	…	1, 1, …, 1
0 0 0	row pattern 1		
0 0 1	row pattern 2		
0 1 0	row pattern 3		
0 1 1	row pattern 4		
1 0 0	row pattern 2		
1 0 1	row pattern 2		
1 1 0	row pattern 1		
1 1 1	row pattern 4		

Fig. 2. Charts $M(f_1)$ and $M(f_2)$ of the multi-output boolean function $\{f_1, f_2\}$ which will be used to illustrate the ideas of the paper. The input partition is given by $\{\{x_1, x_2, x_3\}, \{x_4, \ldots, x_n\}\}$. Each chart obviously consists of 8 rows. A row pattern is associated to each row. There are three (four) different row patterns in $M(f_1)$ ($M(f_2)$) denoted by the numbers 1, 2, and 3 (1 to 4). Thus, $r_1 = r_2 = 2$ holds.

Of course, such h boolean functions need not to exist. We have proven the problem whether such functions $\alpha_1, \ldots, \alpha_h$ exist to be NP-complete. Nevertheless, we have to solve CDF†. An algorithm which is applicable from the practical point of view (as shown by the benchmarking results) is presented in the next two sections. In particular, experimental results show that the running time needed for computation of common decomposition functions is only a small fraction of the total running time of the tool. The running time is dominated by the computation of good input partitions, not by the computation of common decomposition functions.

III An algorithm for CDF

In the following, let $f = \{f_1, \ldots, f_m\} \in B_{n,m}$ be a multi-output boolean function. Each single-output function f_k has to be decomposed with respect to the same given input partition $A = \{X_1, X_2\}$ (computed during output partitioning (see [17])) which will be fixed in the following.

A. *Theoretical background*

We start with a theoretical result working towards a solution to CDF. It gives a condition necessary and sufficient that h single-output functions $\alpha_1, \ldots, \alpha_h \in B_p$ are common decomposition functions of $f_1, \ldots, f_m$. It is a generalization of a lemma shown by Karp [11]. For this, we need the following notations. The rows of chart $M(f_k)$ induce a partition of $\{0,1\}^p$ into equivalence classes $K_1^{(k)}, \ldots, K_{dr^{(k)}}^{(k)}$ such that $v, v' \in \{0,1\}^p$ belong to the same class $K_j^{(k)}$ if and only if the two corresponding row patterns of $M(f_k)$ are identical. We denote the corresponding equivalence relation by $\equiv_k$ and the set of the equivalence classes $\{K_1^{(k)}, \ldots, K_{dr^{(k)}}^{(k)}\}$ by $\{0,1\}^p/_{\equiv_k}$. Let $\theta^{(k)} : \{0,1\}^p \to \{1, \ldots, dr^{(k)}\}$ be the function which maps $v \in \{0,1\}^p$ to the index j of the class $K_j^{(k)}$ to which it belongs.

Continued example (see Figure 2): $\{0,1\}^3/_{\equiv_1}$ consists of 3 elements, namely $K_1^{(1)} = \{000, 001\}$, $K_2^{(1)} = \{010, 011, 111\}$, and $K_3^{(1)} = \{100, 101, 110\}$. It follows that $\theta^{(1)}(000) = \theta^{(1)}(001) = 1$, $\theta^{(1)}(010) = \theta^{(1)}(011) = \theta^{(1)}(111) = 2$, and $\theta^{(1)}(100) = \theta^{(1)}(101) = \theta^{(1)}(110) = 3$. $\{0,1\}^3/_{\equiv_2}$ consists of 4 classes, namely $K_1^{(2)} = \{000, 110\}$, $K_2^{(2)} = \{001, 100, 101\}$, $K_3^{(2)} = \{010\}$, and $K_4^{(2)} = \{011, 111\}$. ◇

Furthermore, for given $\alpha_{1,\ldots,h}$ ‡ and all $a \in \{0,1\}^h$, let $S_a^{(k)}$ be the set $\{\theta^{(k)}(v);\ \alpha_{1,\ldots,h}(v) = a\}$ of those classes which contain a row mapped to a by $\alpha_{1,\ldots,h}$. ($\alpha_{1,\ldots,h}$ is not able to tell these rows apart (see the Encoding Property).) Note that $S_a^{(k)}$ and $S_{a'}^{(k)}$ need not to be disjoint for $a \neq a'$, and that the number $\mid S_a^{(k)} \mid$ of elements of $S_a^{(k)}$ equals the number of *distinct* row patterns of $M(f_k)$ mapped to a by $\alpha_{1,\ldots,h}$. Thus, $\max\{\mid S_a^{(k)} \mid;\ a \in \{0,1\}^h\}$ denotes the *'inability to distinguish' of* $\alpha_{1,\ldots,h}$ *with respect to* f_k. $itd(A, f_k, \alpha_{1,\ldots,h})$ will denote value $\max\{\mid S_a^{(k)} \mid;\ a \in \{0,1\}^h\}$ in the following.

Continued example: Let h be equal 2. Assume that $\forall (v_1, v_2, v_3) \in \{0,1\}^3$ $\alpha_1(v_1, v_2, v_3) = v_2$ and $\alpha_2(v_1, v_2, v_3) = v_3$. It follows that $S_{00}^{(1)} = \{1, 3\}$ holds because $\alpha_{1,2}(000) = \alpha_{1,2}(100) = 00$ and $\theta^{(1)}(000) = 1$, $\theta^{(1)}(100) = 3$. Furthermore the following equalities hold: $S_{01}^{(1)} = \{1, 3\}$, $S_{10}^{(1)} = \{2, 3\}$, and $S_{11}^{(1)} = \{2\}$. Thus the inability to distinguish $itd(A, f_1, \alpha_{1,2})$ of $\alpha_{1,2}$ with respect to f_1 is equal to 2. ◇

Lemma 1 $\alpha_1, \ldots, \alpha_h \in B_p$ *are common decomposition functions of* $f_1, \ldots, f_m$ *with respect to* A *such that there is an optimal decomposition of* f_k *of the form*

$$\begin{aligned} f_k(x_1, \ldots, x_n) &= g^{(k)}(\alpha_1(X_1), \ldots, \alpha_h(X_1), \\ &\qquad \alpha_{h+1}^{(k)}(X_1), \ldots, \alpha_{r_k}^{(k)}(X_1), X_2) \end{aligned}$$

($\forall k \in \{1, \ldots, m\}$) *if and only if the inability to distinguish* $itd(A, f_k, \alpha_{1,\ldots,h})$ *of* $\alpha_{1,\ldots,h}$ *with respect to* f_k *is* $\leq 2^{r_k - h}$ ($\forall k$). ◇

Proof: Since $(\alpha_{1,\ldots,h}, \alpha_{h+1,\ldots,r_k}^{(k)})$ has to assign different values to rows of chart $M(f_k)$ with different row patterns (see the Encoding Property), $\alpha_{h+1,\ldots,r_k}^{(k)}$ has to assign different values to those rows which cannot be told apart by $\alpha_{1,\ldots,h}$. As $\alpha_{h+1,\ldots,r_k}^{(k)}$ can produce at most $2^{r_k - h}$ different values, the statement of the lemma follows. ■

†The (maximal) value of parameter h of CDF is determined by logarithmic search. After that we solve CDF for subsets of $\{f_1, \ldots, f_m\}$, but only for such subsets $\{f_{i_1}, \ldots, f_{i_l}\}$ where all pairs f_{i_j} and f_{i_k} have at least one common decomposition function. Note that this question is *not* NP-complete, but can be decided efficiently by dynamic programming. Also note that the algorithms for the computation of common decomposition functions given in this paper can be generalized in a canonical manner for the case that some of the decomposition functions α_i^k $(i > h)$ are already predetermined. More details of how the CDF algorithm is integrated in the tool can be found in [17].

‡Remember that $\alpha_{1,\ldots,h}$ denotes the tuple $(\alpha_1, \ldots, \alpha_h)$.

B. The basic algorithm

In this section we describe an algorithm which could basically solve CDF. Note that we actually don't use this algorithm in our ROBDD based implementation, but it leads to a better understanding of how the ROBDD based algorithm works.

CDF can be solved (on principle) by computing $\alpha_{1,\ldots,h}$ by a (simplified) branch and bound algorithm. The sets $S_a^{(k)}$, which determine the inability to distinguish $itd(A, f_k, \alpha_{1,\ldots,h})$ with respect to f_k, are constructed step by step. In the initialization phase, $\alpha_{1,\ldots,h}(v')$ is set to *undef* for all $v' \in \{0,1\}^p$, and $S_{a'}^{(k)}$ is set to the empty set for all a' and k. Each time we enter the main loop (step 4 of the algorithm; see Figure 3) there is a $v \in \{0,1\}^p$ and a vector $a \in \{0,1\}^h$ such that $\alpha_{1,\ldots,h}(v')$ is defined for all v' with $int(v') < int(v)$,[§] and there is no extension of the present function table with $\alpha_{1,\ldots,h}(v) = a'$ and $int(a') < int(a)$ which does not violate the condition of lemma 1. In this step we test whether the condition of lemma 1 is violated if $\alpha_{1,\ldots,h}(v)$ is set to a. If the condition is violated, we have to backtrack if $int(a) = 2^h - 1$, i.e., $a = (1, \ldots, 1)$. If $int(a) < 2^h - 1$, we enter the loop once again with a incremented by 1. The sets $S_a^{(k)}$ are updated in each step. (Thus, the numbers $itd(A, f_k, \alpha_{1,\ldots,h})$ of lemma 1 are implicitely updated, too.) For detailed information of the algorithm see the pseudo code shown in Figure 3.

Continued example: Assume $h = 1$, and remember that $r_1 = r_2 = 2$ holds. The CDF algorithm above computes $\alpha_1 : \{0,1\}^3 \to \{0,1\}$ defined by $\alpha_1(v) = 1 \iff v \in \{010, 011, 111\}$ as common decomposition function of f_1 and f_2. The inability to distinguish with respect to f_k ($k = 1, 2$) is equal to 2 as $S_0^{(1)} = \{1,3\}$, $S_1^{(1)} = \{2\}$, and $S_0^{(2)} = \{1,2\}$, $S_1^{(2)} = \{3,4\}$.[¶] ◇

IV A ROBDD BASED IMPLEMENTATION

Assume the function $f = \{f_1, \ldots, f_m\} \in B_{n,m}$ we want to decompose is given by m ROBDDs $bdd_1, \ldots, bdd_m$ and that the ordering of the variables is given by $(x_1, \ldots, x_n)$ (for illustration see Figure 4).

A. BDD based decomposition

The following observations result in an efficient implementation of functional decomposition:

The first observation, already made in [7, 12], is that for all $(v_1, \ldots, v_p) \in \{0,1\}^p$ the row pattern belonging to row $(v_1, \ldots, v_p)$ of $M(f_k)$ equals the function table of the cofactor $(f_k)_{x_1^{v_1} \cdot \ldots \cdot x_p^{v_p}}$ (with $x_i^0 = \overline{x_i}$ and $x_i^1 = x_i$). Thus the problem of determining the number $dr^{(k)}$ of different row patterns of $M(f_k)$ is equivalent to the problem of computing the number of different cofactors $(f_k)_{x_1^{v_1} \cdot \ldots \cdot x_p^{v_p}}$. The ROBDD of the cofactor $(f_k)_{x_1^{v_1} \cdot \ldots \cdot x_p^{v_p}}$ is given by the sub-bdd of bdd_k whose root is reached by starting at the root of bdd_k and then following the path corresponding to $(v_1, \ldots, v_p)$. The roots of these cofactors are called *linking nodes* (the shaded nodes in Figure 4). Since f_k is given by a ROBDD, the number of different linking nodes of bdd_k obviously equals the number of different cofactors. The computational complexity of determining the number of different linking nodes is at most linear in the size of bdd_k since it can be determined by traversing bdd_k in a depth first search manner.

The second observation is that encoding the linking nodes of bdd_k with a code of length r_k (which is the logarithm of the number of linking nodes of bdd_k) results in an optimal decomposition of f_k. (Of course this simple approach does not lead to common decomposition functions.) For $1 \le i \le r_k$ the corresponding decomposition function $\alpha_i^{(k)}$ is given by substituting the linking nodes of bdd_k by the ith bits of the codewords belonging to the linking nodes. The composition function $g^{(k)}$ is given by substituting the part of bdd_k, which corresponds to the variables $x_1, \ldots, x_p$, by the corresponding code-tree. For illustration see Figure 5.

A decomposition constructed in this way leads to decomposition functions $\alpha_j^{(k)}$ of f_k which assign the same value to all those $(v_1, \ldots, v_p) \in \{0,1\}^p$ for which the corresponding row patterns of $M(f_k)$ are identical. (Note that till now the decomposition functions $\alpha_1^{(k)}, \ldots, \alpha_{r_k}^{(k)}$ of a single-output function f_k are allowed to assign different values to rows with identical row patterns in the case that the total number of different row patterns is less than 2^{r_k}.)

B. BDD based solution of CDF

In the last section we have made the observation, that an encoding of the linking nodes leads to a special subclass of decomposition functions, which we will call "equivalence preserving decomposition functions":

Definition 3 *A decomposition function $\alpha_i \in B_p$ of a boolean function $f_k \in B_n$ is said to* preserve equivalences *if $\alpha_i(v) = \alpha_i(v')$ holds for every $v, v' \in \{0,1\}^p$ with $v \equiv_k v'$.* ◇

In practical applications functions f_k often have some desirable properties like symmetry in some variables or independence of some variables. The following lemma shows that equivalence preserving decomposition functions "preserve such properties".

Lemma 2 *If f_k is decomposed with respect to the input partition $\{\{x_1, \ldots, x_p\}, \{x_{p+1}, \ldots, x_n\}\}$ and f_k is symmetric in x_i and x_j ($i, j \in \{1, \ldots, p\}$) (i.e. $f_k(\ldots, x_i, \ldots, x_j, \ldots) = f_k(\ldots, x_j, \ldots, x_i, \ldots)$), then all equivalence preserving decomposition functions of f_k are symmetric in x_i and x_j too. An analogous statement holds for independence of x_i, $i \in \{1, \ldots, p\}$.* ◇

That's why we restrict our search for common decomposition functions to equivalence preserving decomposition

§ $int(y)$ denotes the natural number represented by the boolean vector y

¶ Because of $itd(A, f_k, \alpha_1) = 2$, there obviously exists a decomposition function $\alpha_2^{(k)} : \{0,1\}^3 \to \{0,1\}$ such that $\forall v, v' \in \{0,1\}^3$ $(\alpha_1(v), \alpha_2^{(k)}(v)) \neq (\alpha_1(v'), \alpha_2^{(k)}(v'))$ if the row patterns of chart $M(f_k)$ coresponding to v and v' are different ($k = 1, 2$).

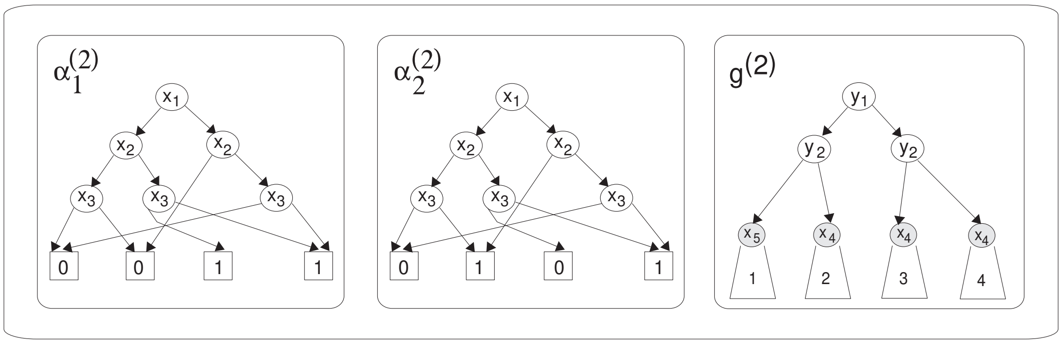

Fig. 5. *Continued example:* Optimal decomposition of f_2 by encoding the first linking node by 00, the second by 01, the third by 10 and the fourth by 11. (To make things clear the OBDDs shown are not reduced.)

functions. We modify the branch and bound algorithm above such that only equivalence preserving decomposition functions are considered and we name the modified branch and bound algorithm "ROBDD based branch and bound algorithm". Moreover the restriction to equivalence preserving decomposition functions has the additional effect that we receive an algorithm which is much more efficient. In the rest of this section we give a sketch of how this algorithm works:

Common equivalence preserving decomposition functions $\alpha_1, \ldots, \alpha_h$ of $f_1, \ldots, f_m$ have to assign the same value to v and $v' \in \{0,1\}^p$ whenever there is a $k \in \{1, \ldots, m\}$ such that the rows of $M(f_k)$ corresponding to v and v' have identical row patterns. More formally, let

$$v \sim v' \overset{\text{def}}{\Longleftrightarrow} (\exists 1 \leq k \leq m)\ v \equiv_k v',$$

then the corresponding equivalence relation partitions the rows, i.e. $\{0,1\}^p$, into equivalence classes $E_1, \ldots, E_l$ such that common equivalence preserving decomposition functions have to assign the same value to each $v \in E_i$. We will denote the set of these equivalence classes by $\{0,1\}^p/_\sim$. (Thus, for all $k \in \{1, \ldots, m\}$ the classes E_i are unions of certain classes $K_j^{(k)}$.)

Continued example: Figure 6 illustrates the definition of this equivalence relation. Consider a graph whose vertices are the 8 rows $000, \ldots, 111$. (Note that this graph will not be constructed by the algorithm presented in subsection C.2.) There is an edge between two rows v and v' if the corresponding row patterns in $M(f_1)$ or $M(f_2)$ are identical, i.e., if $v \equiv_1 v'$ or $v \equiv_2 v'$. (Edges resulting from f_1 are drawn in bold.) This results in a graph whose connected components determine the equivalence classes E_i. There are two classes, namely $E_1 = \{000, 001, 100, 101, 110\}$ and $E_2 = \{010, 011, 111\}$. E_1 is marked by shaded nodes. ◇

Thus, the ROBDD based branch and bound algorithm assigns values not to single elements of $\{0,1\}^p$ but to whole classes E_i. Because l mostly is much smaller than 2^p, this approach considerably reduces the running time compared to the original branch and bound algorithm (see subsection B.). (The benchmarking results will also show that this reduction of the running time can be achieved without reducing the quality of the circuits constructed. One reason for this fact is given by lemma 2.) During the ROBDD based branch and bound algorithm, every time a value $a \in \{0,1\}^h$ is assigned to an equivalence class E_i by $\alpha_{1,\ldots,h}$, the sets $S_a^{(k)}$ $(\forall k)$, which contain the different row patterns of $M(f_k)$ mapped to a by $\alpha_{1,\ldots,h}$, have to be updated by $S_a^{(k)} = S_a^{(k)} \cup SET_i^{(k)}$. $SET_i^{(k)}$ denotes the set $\{j;\ K_j^{(k)} \subseteq E_i\}$, i.e., the set which consists of the indices (with respect to $\theta^{(k)}$) of the different row patterns of $M(f_k)$ belonging to E_i. Note that the sets $SET_i^{(k)}$ and $SET_j^{(k)}$ are disjoint if $i \neq j$.

Continued example: $SET_1^{(1)} = \{1,3\}$, $SET_2^{(1)} = \{2\}$, $SET_1^{(2)} = \{1,2\}$, and $SET_2^{(2)} = \{3,4\}$. ◇

The ROBDD based branch and bound algorithm requires a preprocessing and postprocessing phase described in the following.

C. Preprocessing steps

In the first preprocessing step we have to efficiently determine the minimum number r_k of decomposition functions required for a decomposition of f_k $(\forall k)$. In the second preprocessing step we construct ROBDDs representing the equivalence classes $K_j^{(k)} \in \{0,1\}^p/_{\equiv_k}$. Then, we determine the ROBDDs of the equivalence classes $\{0,1\}^p/_\sim$ corresponding to $\{f_1, \ldots, f_m\}$ and thus also the corresponding sets $SET_i^{(k)}$ the algorithm requires.

Note that all these computation steps have to be done based on the ROBDD's without constructing the charts $M(f_k)$. Since we have already explained in section A. how to efficiently determine the minimum number r_k of decomposition functions required for a decomposition of a function f_k, it remains to show how to efficiently compute the equivalence classes.

C.1 Computation of the equivalence classes with respect to $\equiv_k$

The ROBDDs $bdd_1^{(k)}, \ldots, bdd_{dr^{(k)}}^{(k)}$ for the characteristic functions of the equivalence classes $K_1^{(k)}, \ldots, K_{dr^{(k)}}^{(k)}$ with respect to $\equiv_k$ can be easily computed from the ROBDDs of the f_k: As already mentioned, identical row patterns of $M(f_k)$ correspond to the same linking node of bdd_k. Thus every equivalence class $K_j^{(k)}$ is associated to exactly one linking node $n_j^{(k)}$ and vice versa. $K_j^{(k)}$ is given by the set of the paths from the root of bdd_k to linking node $n_j^{(k)}$. Thus, substituting the sub-bdd of bdd_k with root $n_j^{(k)}$ by constant 1 and connecting every edge which leaves the cone of $n_j^{(k)}$ to constant 0 results in a ROBDD, whose ON-set is given by $K_j^{(k)}$. We call this ROBDD $bdd_j^{(k)}$. The cone of node $n_j^{(k)}$ is defined to be the set of those nodes x of bdd_k such that there is a path from x to $n_j^{(k)}$. For illustration see Figure 7.

C.2 Computation of the equivalence classes with respect to $\sim$

To compute the characteristic functions of the equivalence classes with respect to $\sim$, we implicitely construct a graph $G = (V, E)$ where the set V of vertices is given by the ROBDDS $bdd_j^{(k)}$ representing the equivalence classes $K_j^{(k)}$. At the end, there is an undirected edge $\{bdd_{j_1}^{(k_1)}, bdd_{j_2}^{(k_2)}\}$ if and only if $bdd_{j_1}^{(k_1)} \wedge bdd_{j_2}^{(k_2)} \neq \emptyset$, i.e., iff $K_{j_1}^{(k_1)} \cap K_{j_2}^{(k_2)} \neq \emptyset$ (see Figure 8). Obviously, there is a one-to-one relation between the set of the connected components (in the graph-theoretical sense) of G and the set of the equivalence classes $\{0,1\}^p/_\sim$. For every class E_i, there is a connected component CC_i of G such that the logical-or of the ROBDDS $bdd_j^{(k)}$ (for any fixed k) corresponding to vertices of CC_i results in a representation of E_i and vice versa.

Continued example: E_1 is represented by the ROBDD $bdd_1^{(1)} \vee bdd_3^{(1)}$ which is the same ROBDD as $bdd_1^{(2)} \vee bdd_2^{(2)}$. E_2 is represented by $bdd_2^{(1)}$ which is the same ROBDD as $bdd_3^{(2)} \vee bdd_4^{(2)}$. $\diamond$

Note that the algorithm described below does not have to test each pair of ROBDDS $bdd_{j_1}^{(k_1)}$, $bdd_{j_2}^{(k_2)}$ whether their ON-sets are disjoint. Virtually, it performs depth first search on graph G. In each step we compute a connected component which contains a node $bdd_z^{(1)}$ not yet touched by calling procedure $search(bdd_z^{(1)}, 1)$. This procedure recursively constructs ROBDDS $cc^{(1)}, \ldots, cc^{(m)}$. At each moment $cc^{(k)}$ equals the ROBDD representing the logical-or of all the nodes $bdd_j^{(k)}$ of the present connected component which have already been touched. At the end of procedure call $search(bdd_z^{(1)}, 1)$, the equation $cc^{(1)} = \ldots = cc^{(m)}$ holds, and $cc^{(1)}$ represents the connected component computed. The exact implementation of *search* is shown in Figure 9. Note that, if the ROBDD *notcovered* is nonempty during a step, there is a row v belonging to the present connected component which is not in the ON-set of $cc^{(j)}$ yet, such that the ROBDD $bdd_u^{(j)}$ which describes the set of the rows which have identical row patterns as v in $M(f_j)$ has to be joined to $cc^{(j)}$.

Procedure *search* is called exactly once for every node of G. During the execution of the body of procedure $search(b, k)$ (without the recursive calls) there are one apply operation (see [2, 6]) performing the logical-or of two ROBDDS and at most $m - 1 + degree(b)$ apply operations performing logical-and of two ROBDDS, where $degree(b)$ denotes the degree of vertex b with respect to G. This results in a number of apply operations which is linear in the size of G as $m - 1 \leq degree(b)$ holds. The running time of the remaining operations is linear in the size of the relevant ROBDD.

D. Postprocessing steps

After the execution of the preprocessing steps, the ROBDD based branch and bound algorithm encodes the equivalence classes as already described. We still have to compute ROBDDS for decomposition and composition functions, such that we can call the algorithm recursively for these functions.

D.1 Computation of the ROBDDs of the common decomposition functions

Assume that a single-output function α_i has been found by the ROBDD based algorithm which can be used as decomposition function of f_k. Note that the algorithm does not explicitly assign values to every $v \in \{0,1\}^p$ but only to the equivalence classes $\{0,1\}^p/_\sim$. As the ROBDD of these equivalence classes are known, we only have to connect by logical-or those ROBDDS whose corresponding equivalence classes are mapped to value 1 by α_i in order to obtain the ROBDD representing α_i.

Continued example: Assume $h = 1$. The ROBDD based branch and bound algorithm constructs $\alpha_1 : \{0,1\}^3 \to \{0,1\}$ defined by $\alpha_1(E_1) = 0$ and $\alpha_1(E_2) = 1$ as common decomposition function of f_1 and f_2. Thus the ROBDD of α_1 is given by the ROBDD specifying E_2, and equals function $\alpha_1^{(2)}$ of Figure 5. $\diamond$

D.2 Computation of the ROBDDs of the composition functions

Once that r_k boolean valued functions $\alpha_1^{(k)}, \ldots, \alpha_{r_k}^{(k)}$ which can be used to decompose function f_k are determined, we have to compute the ROBDD of the corresponding composition function $g^{(k)}$. This is done as (informally) described in section A. (second observation). For illustration see Figure 5 once again. The ROBDD of $g^{(k)}$ can be constructed using the linking nodes $n_j^{(k)}$ and combining these cofactors through the codetree with *if-then-else*-operations of the ROBDD-package [2]. This step doesn't differ from the computation of the composition functions for the case that decomposition is done by a simple encoding of the linking nodes without computation of common decomposition functions.

V EXPERIMENTAL RESULTS

We applied our tool, which uses the CDF algorithm described above as basis, to a number of benchmarks of the 1991 MCNC multi-level logic benchmark set. We will call the ROBDD based implementation of our tool *mulopII*.

TABLE I

Comparison between the ROBDD based implementation of our synthesis tool *mulopII* and the former version *mulop* working on decomposition charts. The technology file consists of the 2-input gates from *stdcell2_2.genlib* available in *octtools*.

Circuit	No. of gates *mulop*	No. of gates *mulopII*	ratio	Running time *mulop*	Running time *mulopII*	ratio	Fraction of running time for CDF (*mulopII*)
9symml	40	45	0.89	1.40 sec	1.23 sec	1.14	0.69%
C17	6	7	0.86	0.32 sec	0.15 sec	2.13	0.01%
cm138a	20	18	1.11	1.01 sec	0.18 sec	5.61	0.89%
cm151a	48	41	1.17	4.16 sec	1.09 sec	3.82	0.13%
cm152a	34	27	1.26	2.15 sec	0.50 sec	4.30	0.36%
cm162a	46	44	1.05	350.65 sec	3.32 sec	105.62	0.26%
cm163a	38	34	1.12	2923.31 sec	2.35 sec	1243.96	0.08%
cm82a	13	13	1.00	0.38 sec	0.21 sec	1.81	0.01%
cm85a	42	42	1.00	7.46 sec	3.73 sec	2.00	0.27%
cmb	24	29	0.83	1836.13 sec	2.52 sec	728.62	0.05%
decod	31	28	1.11	26.15 sec	2.56 sec	10.21	1.93%
f51m	64	56	1.14	3.14 sec	1.83 sec	1.71	0.28%
majority	9	9	1.00	0.44 sec	0.08 sec	5.50	0.01%
parity	15	15	1.00	111.06 sec	1.37 sec	81.07	0.00%
z4m1	20	20	1.00	0.66 sec	0.76 sec	0.87	0.16%

Our former implementation working on charts [17] will be called *mulop*.

Table I shows gate counts* and running times (in CPU seconds, measured on a SPARCstation 10/30 (64 MByte RAM)) of our ROBDD based algorithm *mulopII* compared to those of our former version *mulop*. For these examples *mulop* has running times up to about 50 CPU minutes while the running time of *mulopII* is at most a few seconds. The experiments prove our ROBDD based version to be much more efficient than the former version. Nevertheless in almost all cases the numbers of gates of the computed circuits are not larger.

The last column of Table I shows the fraction of running time which is used in the computation of common decomposition functions compared to the total running time of the tool. It shows that only a very small fraction of the total running time is used for the computation of common decomposition functions. The running time is dominated by the computation of good input partitions, not by the computation of common decomposition functions. This confirms our approach to compute common decomposition functions rather than to encode linking nodes in a straightforward manner (as described in section A.).

Table II shows a comparison between *mulopII* and *sis 1.1* with respect to layout size.† For almost two thirds of the benchmark set, our approach dominates (or is as good as) that of *sis* with respect to layout size. Nevertheless, the signal delays of our realizations for more than two thirds of the circuits considered are better (or equal) than those of the realizations synthesized by *sis*.

Since functional decomposition techniques are especially suited for FPGA synthesis, we also made experiments with *mulopII* on a number of benchmarks for XC3000 device. As proposed by Lai, Pan, Pedram, Vrudhula [13, 14], we applied our algorithm to multi-output functions which result from a node clustering in a boolean network which is received by running the rugged script

*The library consists of the 2-input gates from *stdcell2_2.genlib* available in *octtools*.

†The technology library used consists of the set of the 2-input gates from *stdcell2_2.genlib* available in *octtools*. Placement and routing was done by *TimberWolf* integrated in *octtools*.

TABLE II

Comparison between *mulopII* and *sis1.1* with respect to layout size, and signal delay.

Circuit	Layout size *sis*	Layout size *mulopII*	ratio	Signal delay *sis*	Signal delay *mulopII*	ratio
9symml	1194336	201400	5.93	27.6	13.6	2.03
C17	28800	31744	0.91	4.2	4.2	1.00
cm138a	103896	87480	1.19	5.8	6.8	0.85
cm151a	95312	177712	0.54	12.6	16.4	0.77
cm152a	85536	106704	0.80	10.0	13.2	0.76
cm162a	131976	192000	0.69	12.0	13.2	0.91
cm163a	144008	164416	0.88	13.0	10.4	1.25
cm82a	74784	61600	1.21	7.2	7.0	1.03
cm85a	165456	180000	0.92	10.2	11.0	0.93
cmb	204792	123496	1.66	9.4	6.8	1.38
decod	140448	119496	1.18	6.2	5.0	1.24
f51m	561184	251392	2.23	51.0	18.4	2.77
majority	42200	39168	1.00	7.8	6.6	1.18
parity	99408	96976	1.03	5.0	5.0	1.00
z4m1	156288	103896	1.50	16.2	9.8	1.65
$\sum$	3228K	1937K	1.67	198.2	147.4	1.34

TABLE III

Experimental results for XC3000 device

Circuit	in	out	Number of CLB FGSyn	Number of CLB FGMap	Number of CLB mis-pga(new)	Number of CLB mulopII
5xp1	7	10	9	15	13	9
9sym	9	1	7	7	7	7
alu2	10	6	52	53	96	51
apex7	49	37	45	47	43	45
b9	41	21	29	27	32	30
bw	5	28	27	27	27	27
C499	41	32	54	49	66	60
C880	60	26	88	74	72	87
clip	9	5	18	20	23	14
count	35	16	24	24	30	26
duke2	22	29	94	178	94	114
e64	65	65	56	55	56	55
f51m	8	8	9	11	15	8
misex1	8	7	9	8	9	9
misex2	25	18	22	21	25	24
rd73	7	3	5	7	5	5
rd84	8	4	8	12	9	8
rot	135	107	144	194	143	146
sao2	10	4	20	27	28	20
vg2	25	8	20	23	18	18
z4ml	7	4	4	5	4	4

[20] on the benchmark circuit. (The multi-output boolean function defined by such a cluster of nodes constitutes an input to the decomposition algorithm.) In Table III we compare our results with FGSyn [14], FGMap [13] and mis-pga(new) [16, 18]. Our first results are promising. Moreover we made the experience that an exploitation of don't cares ‡ (which is not included in this paper) will lead to further improvements of the results.

VI Conclusion

We have presented a ROBDD based technique for computing common decomposition functions of multi-output boolean functions. This algorithm has been integrated in our multi-level synthesis tool which has been presented in [17] where more details of how the CDF algorithm is integrated can be found. The benchmarking results show that most of the circuits constructed by our synthesis tool are very efficient. They also prove it to be applicable in terms of running time. Results for FPGA mapping are promising.

References

[1] R.L. Ashenhurst. The decomposition of switching functions. In *Proceedings on an International Symposium on the Theory of Switching* held at Comp. Lab. of Harvard University, pages 74–116, 1959.

[2] K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Proc. of 27th Design Automation Conference DAC90*, pages 40–45, 1990.

[3] R.K. Brayton, C.T. McMullen G.D. Hachtel, and A.L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1984.

[4] R.K. Brayton, G.D. Hachtel, and A. L. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proceedings of the IEEE*, 78(2):264–300, February 1990.

[5] R.K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A.R. Wang. MIS: A multiple-level logic optimization system. *IEEE Trans. on CAD*, CAD-6(11), November 1987.

[6] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, August 1986.

[7] S. Chang and M. Marek-Sadowska. BDD representation of incompletely specified functions. In *Notes of the International Workshop on Logic Synthesis held in Tahoe City, California*, May 1993.

[8] H.A. Curtis. A generalized tree circuit. *J. Assoc. Comput. Mach.*, 8:484–496, 1961.

[9] T. Hwang, R.M. Owens, and M.J. Irwin. Exploiting communicaton complexity for multilevel logic synthesis. *IEEE Trans. on CAD*, CAD-9(10):1017–1027, October 1990.

[10] T. Hwang, R.M. Owens, and M.J. Irwin. Efficient computing communication complexity for multilevel logic synthesis. *IEEE Trans. on CAD*, CAD-11(5):545–554, May 1992.

[11] R.M. Karp. Functional decomposition and switching circuit design. *Journal of Society of Industrial Applied Mathematics*, 11(2):291–335, June 1963.

‡Even if the benchmark circuit doesn't contain any don't cares, we will receive don't cares when we apply decomposition recursively to the composition functions.

[12] Y. Lai, M. Pedram, and S. Vrudhula. BDD based decomposition of logic functions with application to FPGA synthesis. In *Proc. of 30th Design Automation Conference DAC93*, pages 642–647, 1993.

[13] Y. Lai, K. Pan, M. Pedram, and S. Sastry. FGMAP: A Technology Mapping Algorithm for Look-Up Table Type FPGAs Based on Function Graphs. In *Workshop Notes IWLS*, pages 9b1–9b4, May 1993.

[14] Y. Lai, K. Pan, and M. Pedram. FPGA Synthesis using Function Decomposition. In *Proceedings of ICCD94*, pages 30–35, 1994.

[15] Y. Lai, M. Pedram, and S. Vrudhula. EVBDD-Based Algorithms for Integer Linear Programming, Spectral Transformation, and Function Decomposition. *IEEE Trans. on CAD*, CAD-13(8), August 1994.

[16] R. Murgai, N. Shenoy, R. K. Brayton, A. Sangiovanni-Vincentelli. Improved Logic Synthesis Algorithms for Table Look Up Architectures. In *Proc. Int'l Conf. on Computer Aided Design*, pages 564-567, 1991.

[17] P. Molitor, C. Scholl. Communication based multilevel synthesis for multioutput boolean functions. In *Proceedings of the 4th Great Lakes Symposium on VLSI, Notre Dame, Indiana*, March 1994.

[18] A. Sangiovanni-Vincentelli, A. E. Gamal, J. Rose. Synthesis Methods for Field Programmable Gate Arrays. In *Proceedings of the IEEE*, vol. 81, pages 1057–1083, July 1993.

[19] T. Sasao. FPGA design by generalized functional decomposition. In *Logic Synthesis and Optimization*, Sasao ed., Kluwer Academic Publisher, pages 233–258, 1993.

[20] H. Savoj, H. Y. Wang. Improved Scripts in MIS-II for Logic Minimization of Combinational Circuits. In *Proc. of Int'l Workshop on Logic Synthesis*, May 1991.

[21] U. Schlichtmann. Boolean Matching and Disjoint Decomposition for FPGA Technology Mapping. In *Proceedings of the IFIP Workshop on Logic and Architecture Synthesis*, pages 83–102, 1993.

[22] E. Sentovich et al. SIS: a system for sequential circuit synthesis. Department of EE and CS, UC Berkeley, May 1992.

[23] B. Wurth, K. Eckl and K. Antreich. Functional Multiple-Output Decomposition: Theory and an Implicit Algorithm. to appear in *Proc. of 32nd Design Automation Conference DAC95.*

1. Let $S_{a'}^{(k)} = \emptyset \quad (\forall 1 \le k \le m,\ a' \in \{0,1\}^h)$,
   $\alpha_{1,\ldots,h}(v') = \mathit{undef} \quad (\forall v' \in \{0,1\}^p)$,
   $v = (0,\ldots,0) \in \{0,1\}^p$, and
   $a = (0,\ldots,0) \in \{0,1\}^h$.

2. Let $\alpha_{1,\ldots,h}(v) = a$, /* $\alpha_{1,\ldots,h}(0,\ldots,0) = (0,\ldots,0)$ */
   $S_a^{(k)} = \{\theta^{(k)}(v)\} \quad (\forall k)$.

3. Increment v.

4. Let $\alpha_{1,\ldots,h}(v) = a$.
   If $(\forall k) \quad |S_a^{(k)} \cup \{\theta^{(k)}(v)\}| \le 2^{r_k - h}$
   /* test whether the condition of lemma 1 is
   not violated */
   then let $S_a^{(k)} = S_a^{(k)} \cup \{\theta^{(k)}(v)\}$ $(\forall k)$.
   Increment v.
   Let $a = (0,\ldots,0) \in \{0,1\}^h$.
   else **while** $\alpha_{1,\ldots,h}(v) == (1,\ldots,1)$
   do let $\alpha_{1,\ldots,h}(v) = \mathit{undef}$.
   Decrement v;
   "$S_{\alpha_{1,\ldots,h}(v)}^{(k)} = S_{\alpha_{1,\ldots,h}(v)}^{(k)} \setminus \{\theta^{(k)}(v)\}$" $\forall k$.
   od
   $a = \alpha_{1,\ldots,h}(v)$.
   Increment a.
   Let $\alpha_{1,\ldots,h}(v) = \mathit{undef}$.
   fi

5. **If** $(\forall v \in \{0,1\}^p)$ $\alpha_{1,\ldots,h}(v) \ne \mathit{undef}$
   then return $\alpha_1,\ldots,\alpha_h$.
   else **if** $v = (0,\ldots,0)$
   then return "There is no solution"
   else goto 4 **fi**
   fi

Fig. 3. Pseudo code of the algorithm solving CDF. In step 2 of the algorithm, we can set $\alpha_{1,\ldots,h}(0,\ldots,0) = (0,\ldots,0)$ without loss of generality because if there exist h common decomposition functions then there also exist h common decomposition functions with $\alpha_{1,\ldots,h}(0,\ldots,0) = (0,\ldots,0)$. Furthermore, the operation $S_{\alpha_{1,\ldots,h}(v)}^{(k)} = S_{\alpha_{1,\ldots,h}(v)}^{(k)} \setminus \{\theta^{(k)}(v)\}$ in step 4 is somewhat more complex than common set difference: the index $\theta^{(k)}(v)$ is only removed from $S_{\alpha_{1,\ldots,h}(v)}^{(k)}$ if there is no $v' \ne v$ with $int(v') < int(v)$, $\alpha_{1,\ldots,h}(v') = \alpha_{1,\ldots,h}(v)$, and $\theta^{(k)}(v') = \theta^{(k)}(v)$.

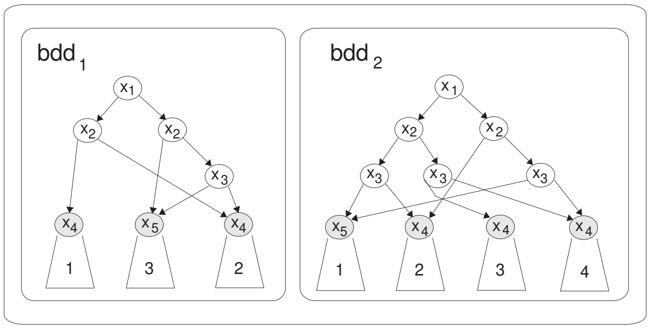

Fig. 4. *Continued example:* The ROBDDs of f_1 and f_2. The left (right) outgoing edge of node x_i corresponds to the case $x_i = 0$ ($x_i = 1$).

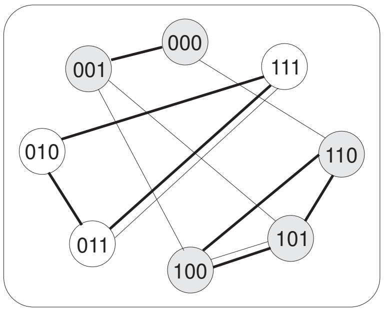

Fig. 6. *Continued example:* Illustration of the definition of the equivalence classes E_i.

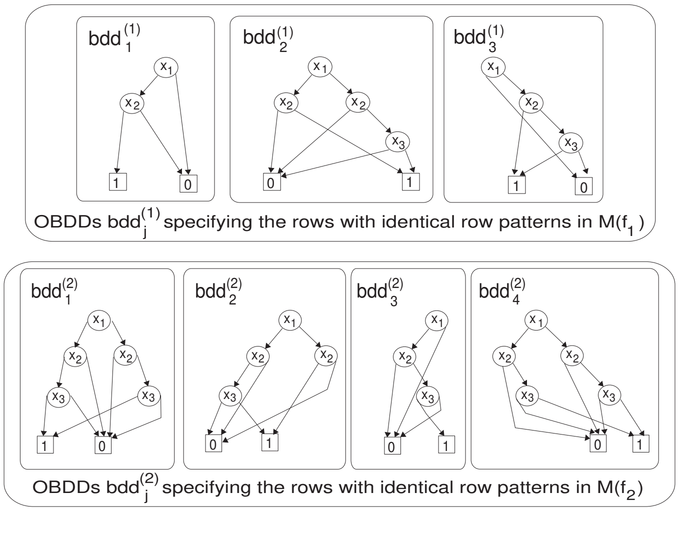

Fig. 7. *Continued example:* Illustration of how to efficiently compute the equivalence classes $\{0,1\}^p/\equiv_k$. (Note that the OBDDs shown still have to be reduced.)

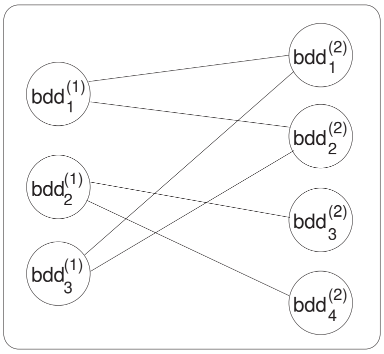

Fig. 8. *Continued example:* Computation of the equivalence classes $\{0,1\}^p/_\sim$. The connected components of graph G represent the classes E_i.

```
procedure search (bdd b, int k)

mark b as touched;
cc^(k) = cc^(k) ∨ b;   /* logical-or of two ROBDDs */
for j = 1 to m do
    if j ≠ k then
       notcovered = b ∧ ¬cc^(j);   /* logical-and */
       while notcovered ≠ ∅ do
             let v be element of ON(notcovered).
             let bdd_u^(j) be the ROBDD
                          with v ∈ ON(bdd_u^(j)).
             call search(bdd_u^(j), j);
             notcovered = b ∧ ¬cc^(j);   /* logical-and */
       od;
    fi;
od;
```

Fig. 9. Pseudo code of the algorithm computing the equivalence classes $\{0,1\}^p/_\sim$