Preprint from Proceedings of IFIP Workshop on Logic and Architecture Synthesis, pp.61-70, Grenoble, France, December 1994

# Efficient ROBDD based computation of common decomposition functions of multi-output boolean functions *

Christoph Scholl

Department of Computer Science
Universität des Saarlandes
D 66041 Saarbrücken, FRG

Paul Molitor

Department of Computer Science
Martin-Luther Universität Halle
D-06099 Halle (Saale), FRG

## 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*). The key concept of our method is the exploitation of "equivalences" of the functions $f_1, \ldots, f_m$ which considerably reduces the running time of the tool.*

*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.*

## 1 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 [10, 11, 13, 15] propose an approach different from the one addressed above. This approach to multi-level logic synthesis which originates from Ashenhurst [1], Curtis [8], Hotz [9], and Karp [12] 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 (multi-output) boolean functions $\alpha$, $\beta$, and $g$, such that $f(x_1, \ldots, x_n) = g(\alpha(x_1, \ldots, x_p), \beta(x_{p+1}, \ldots, x_n))$ holds for all $(x_1, \ldots, x_n) \in \{0,1\}^n$ (see Figure 1). If one

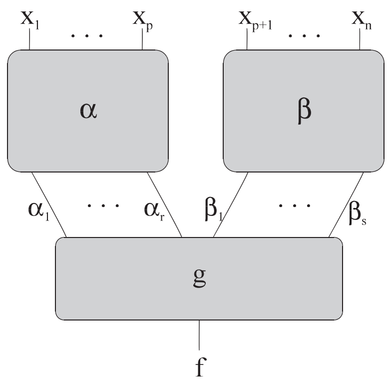


**Figure 1**: Decomposition of a boolean function $f : \{0,1\}^n \to \{0,1\}$

wants to apply functional decomposition recursively to the decomposition functions $\alpha$ and $\beta$, a generalization to multi-output boolean functions is required. Of course, the approaches of the papers above can also be applied to multi-output boolean functions $f = (f_1, \ldots, f_m) : \{0,1\}^n \to \{0,1\}^m$ either by considering multi-output boolean functions as single-output multi-value functions $f' : \{0,1\}^n \to \{0, \ldots, 2^m - 1\}$ defined by $f'(x_1, \ldots, x_n) = \sum_{i=1}^m f_i(x_1, \ldots, x_n) \cdot 2^{i-1}$ or by decomposing each single-output boolean function $f_i$ independently of the other single-output functions $f_j$ ($j \ne i$) and only then testing whether they use some identical decomposition functions by chance. The first method has the drawback that it decomposes each single-output function $f_i$ with respect to the same input partition which can result in poor realizations. It does not take into consideration that there possibly exist single-output boolean functions $f_i$ and $f_j$ such that there does not exist one input partition good for both, $f_i$ and $f_j$. Furthermore, even in case that there is an input partition good for every single-output function of $f$, it is unlikely

*This work was supported in part by DFG grant SFB 124 and the Graduiertenkolleg of the Universität des Saarlandes

that there is a decomposition where function $g$ has much less inputs than $f$ (if $m$ is large enough), i.e., $g$ is not much easier to synthesize than $f$. The drawback of the second method is clear. In the final analysis, it does not use the potential of reusing subcircuits for different outputs of $f$.

In [14] the authors presented a multi-level synthesis method for multi-output boolean functions based on communication complexity which avoids both drawbacks. The method can be 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 $Y_k$. Benchmarking results of circuits taken from 1991 MCNC benchmark set have shown the technique to be effective with respect to layout size and signal delay.

The paper in hand presents a much more efficient version of our algorithms from [14]. During the computation of common decomposition functions we efficiently make use of REDUCED ORDERED BINARY DECISION DIAGRAMS (ROBDD), which are compact representations for many of the boolean functions encountered in typical applications [6]. In this paper we show that it is possible to carry out all necessary steps based on ROBDD's. In particular we show that the computation of *common decomposition functions* for the decomposition of several single-output functions, which is the basis of step 2 of the technique above [14], can be performed efficiently based on ROBDD's. (The crucial point of this method is the exploitation of "equivalences" of the functions $f_1, \dots, f_m$ which results in the computation of connected components (in the graph-theoretical sense)).

Benchmarking results show the new method to be efficient with respect to layout size, signal delay *and running time*.

We start by giving some basic definitions (section 2) and summarize the algorithm for computing common decomposition functions (section 3). In section 4 we demonstrate how to apply ROBDDs to implement this algorithm. Experimental results close the paper (section 5).

## 2 Basic definitions

A multi-output boolean function $\phi$ 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 $B_{n,1}$. $\phi_{i,\dots,j}$ $(i \leq j)$ 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*

$$\begin{aligned} f_k(x_1, \dots, x_n) \quad = \quad & g^{(k)}(\alpha_1^{(k)}(X_1), \dots, \alpha_{r'_k}^{(k)}(X_1), \\ & \beta_1^{(k)}(X_2), \dots, \beta_{s'_k}^{(k)}(X_2)) \end{aligned}$$

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

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 $(v'_1, \dots, v'_p)$. $\diamond$

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 $\lceil \log p_1^{(k)} \rceil$ where $p_1^{(k)}$ is the number of distinct row patterns in $M(f_k)$. $r_k$ will denote value $\lceil \log p_1^{(k)} \rceil$ 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* $r_k$ $(= \lceil \log p_1^{(k)} \rceil)$ *decomposition functions* $\alpha_1^{(k)}, \dots, \alpha_{r_k}^{(k)}$ *with domain* $X_1$. $\diamond$

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. (CDF can be posed for $X_2$ in an analogous manner.)

**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 $h \leq r_k$ $(= \lceil \log p_1^{(k)} \rceil)$ $(\forall k)$.

**Find**: $h$ single-output boolean functions $\alpha_1, \dots, \alpha_h \in B_p$, which can be used as decomposition functions of every

*Likewise, the minimum number $s_k$ of interconnections between $\beta^{(k)}$ and $g^{(k)}$ is $\lceil \log p_2^{(k)} \rceil$ where $p_2^{(k)}$ is the number of distinct column patterns in $M(f_k)$.

$f_1$

| $x_1x_2x_3$ \ $x_4, x_5, \ldots, x_n$ | $x_4$: 00 ... 1, $x_5$: 00 ... 1, $x_n$: 01 ... 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 |

$f_2$

| $x_1x_2x_3$ \ $x_4, x_5, \ldots, x_n$ | $x_4$: 00 ... 1, $x_5$: 00 ... 1, $x_n$: 01 ... 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 |

**Figure 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 $A$ 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.

single-output function $f_k$ for $k = 1, \ldots, m$ such that there is an optimal decomposition of $f_k$ of the form

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

$\diamond$

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.

# 3 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 [14])) which will be fixed in the following.

## 3.1 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 [12]. For this, we need the following notations. The rows of chart $M(f_k)$ induce a partition of $\{0,1\}^p$ into equivalence classes $K_1^{(k)}, \ldots, K_{p_1^{(k)}}^{(k)}$ 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 $\{K_1^{(k)}, \ldots, K_{p_1^{(k)}}^{(k)}\}$ by $\{0,1\}^p/\equiv_k$. Let $\theta^{(k)} : \{0,1\}^p \to \{1, \ldots, p_1^{(k)}\}$ 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 $K_4^{(2)} = \{011, 111\}$. $\diamond$

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. $\diamond$

**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*

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

*($\forall k \in \{1, \ldots, m\}$) 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 $\leq 2^{r_k - h}$ ($\forall k$).* $\diamond$

**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. ■

## 3.2 The basic algorithm

CDF can be solved 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 $v' \in \{0,1\}^p$,

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

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 $S_0^{(2)} = \{1,2\}$, $S_1^{(2)} = \{3,4\}$.§ ◇

# 4 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). Then, the following observations result in an efficient implementation.

## 4.1 Some observations

The first observation, already made in [7, 13], 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 $(f_k)_{x_1^{v_1}\ldots x_p^{v_p}}$ (with $x_i^0 = \overline{x_i}$ and $x_i^1 = x_i$). Thus the problem of determining the number $p_1^{(k)}$ of different row patterns of $M(f_k)$ is equivalent to the problem of computing the number of different cofactors $(f_k)_{x_1^{v_1}\ldots x_p^{v_p}}$. The ROBDD of the cofactor $(f_k)_{x_1^{v_1}\ldots x_p^{v_p}}$ 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.

‡ $int(y)$ 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 $(k = 1,2)$.

1. Let $S_{a'}^{(k)} = \emptyset \quad (\forall 1 \le k \le m,\ a' \in \{0,1\}^h)$,
   $\alpha_{1,\ldots,h}(v') = undef \quad (\forall v' \in \{0,1\}^p)$,
   $v = (0,\ldots,0) \in \{0,1\}^p$, 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** $(\forall k)\ \mid S_a^{(k)} \cup \{\theta^{(k)}(v)\} \mid \le 2^{r_k - h}$
   /* 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 $\alpha_{1,\ldots,h}(v) = undef$.
   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 $\alpha_{1,\ldots,h}(v) = undef$.
   **fi**
5. **If** $(\forall v \in \{0,1\}^p)\ \alpha_{1,\ldots,h}(v) \neq undef$
   **then** return $\alpha_1,\ldots,\alpha_h$.
   **else** **if** $v = (0,\ldots,0)$
   **then** return "There is no solution"
   **else** goto 4 **fi**
   **fi**

**Figure 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 $v' \neq v$ with $int(v') < int(v)$, $\alpha_{1,\ldots,h}(v') = \alpha_{1,\ldots,h}(v)$, and $\theta^{(k)}(v') = \theta^{(k)}(v)$.

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 $i$th bits of the codewords belonging to the linking nodes. The composition function $g^{(k)}$ is given by

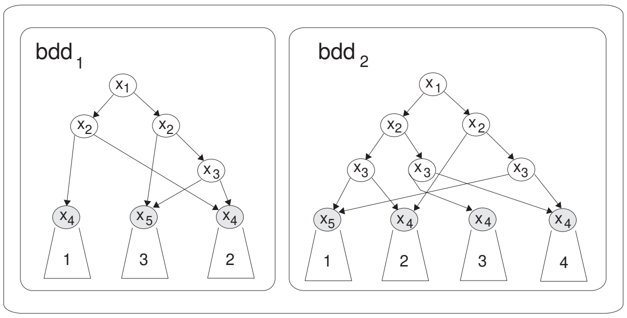


**Figure 4**: *Continued example:* The ROBDDs of $f_1$ and $f_2$. The left (right) outgoing edge of node $x_i$ corresponds to the case $x_i = 0$ ($x_i = 1$).

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 $2^{r_k}$.)

In order to use this simple method of constructing the decomposition functions and composition functions, the branch and bound algorithm above is modified so that only equivalence preserving decomposition functions are considered. We will name the modified algorithm "ROBDD based branch and bound algorithm".

**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* $v \equiv_k v'$. $\diamond$

Thus, 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

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

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 $\{0,1\}^p/_\sim$.

*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 4.2.2.) 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. $\diamond$

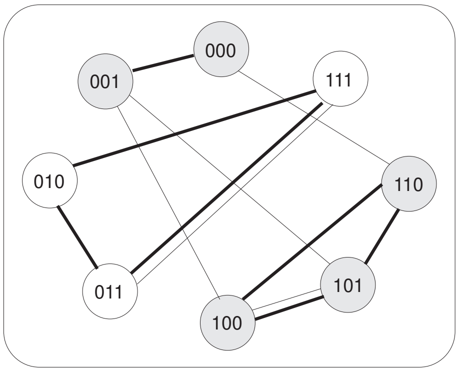


**Figure 6**: *Continued example:* Illustration of the definition of the equivalence classes $E_i$.

Thus, the ROBDD based branch and bound algorithm assigns values to the subsets $E_1, \ldots, E_l$. 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 3.2). (The benchmarking results will also show that this reduction of the running time can be achieved without reducing the quality of the circuits constructed.) 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 $SET_2^{(2)} = \{3,4\}$. $\diamond$

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

## 4.2 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.

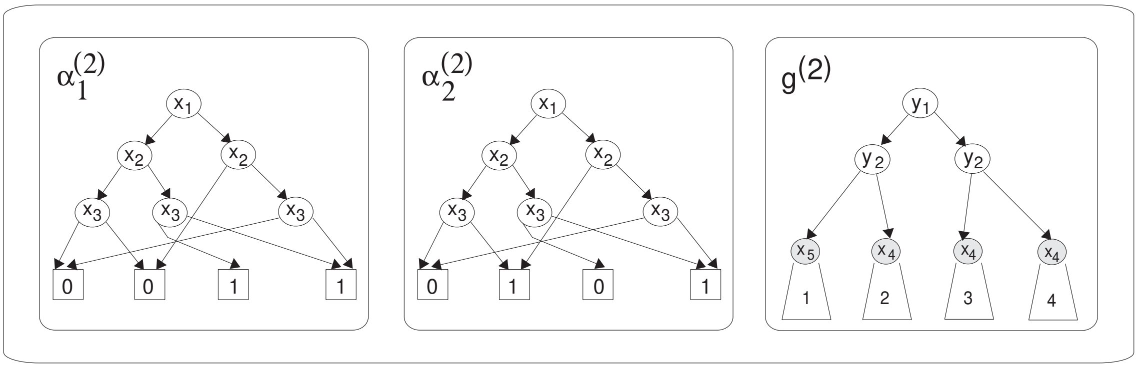


**Figure 5**: *Continued example:* Optimal decomposition of $f_2$ by encoding the first linking node by 00, the second by 01, the third by 10 and the fourth by 11. (To make things clear the OBDDs shown are not reduced.)

Note that all these computation steps have to be done without constructing the charts $M(f_k)$. Since we have already explained in section 4.1 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.

### 4.2.1 Computation of the equivalence classes with respect to $\equiv_k$

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.

### 4.2.2 Computation 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 their ON-sets are not disjoint (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 $bdd_3^{(2)} \vee bdd_4^{(2)}$. ◇

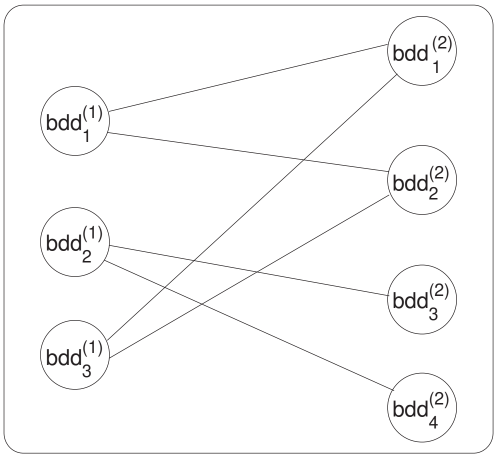

**Figure 8**: *Continued example:* Computation of the equivalence classes $\{0,1\}^p/_\sim$. The connected components of graph $G$ represent the classes $E_i$.

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 non-

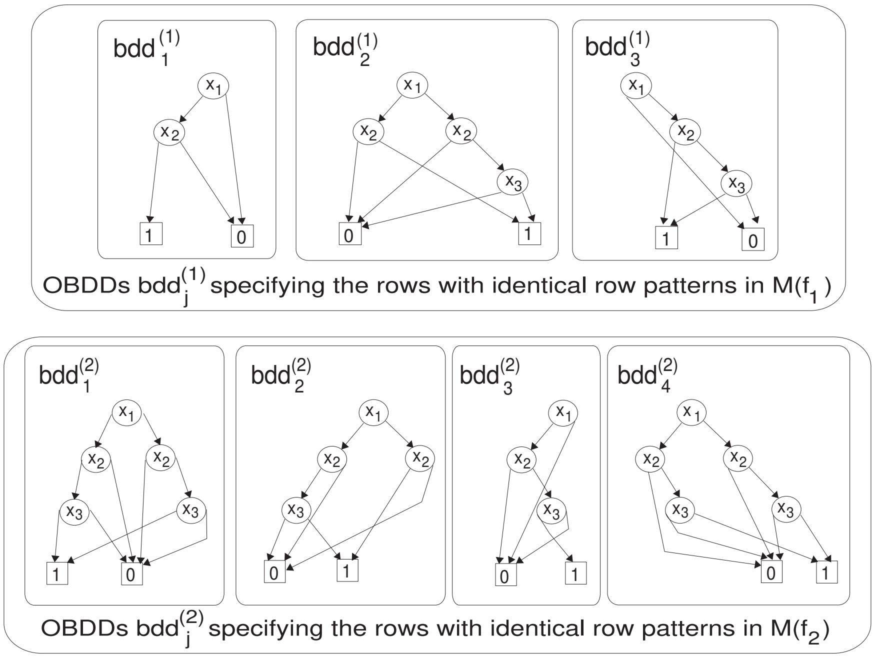


**Figure 7**: *Continued example:* Illustration of how to efficiently compute the equivalence classes $\{0,1\}^p/_{\equiv_k}$. (Note that the OBDDs shown still have to be reduced.)

```
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;
```

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

empty 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, so 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 $m-1 \le degree(b)$ holds. The running time of the remaining operations is linear in the size of the relevant ROBDD.

## 4.3 Postprocessing steps

After the execution of the preprocessing steps, the ROBDD based branch and bound algorithm encodes the equivalence classes as already described.

### 4.3.1 Computation of the ROBDDs of the decomposition functions

Assume that a single-output function $\alpha_i$ 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 $\alpha_i$ in order to obtain the ROBDD representing $\alpha_i$.

*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 $\alpha_1$ is given by the ROBDD specifying $E_2$, and equals function $\alpha_1^{(2)}$ of Figure 5. ◇

### 4.3.2 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 4.1 (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].

## 5 Benchmarking results

We have synthesized several examples of the 1991 MCNC multi-level logic benchmark set in order to compare *factor*, *factorII* [10, 11], which are communication based multi-level synthesis tools developed at PennState University, and *misII* [5], *sis 1.1* [16] to our tool which uses the CDF algorithm described above as basis.* We will call the ROBDD based implementation of our tool *mulopII*. The former implementation working on charts will be called *mulop*.

As running time considerations motivate this paper, we compared the running times of our ROBDD based implementation *mulopII* to those of our former version *mulop*. Table 1 shows gate counts† and running times for some of the MCNC multi-level logic benchmark circuits. The running times are measured on a SPARCstation 10/30 (64 MByte RAM). The experiments show that the ROBDD based implementation is much faster than the former version. 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. In particular, these experiments prove our synthesis tool to be applicable in terms of running time. Although the running times of *mulopII* are much smaller than those of our former version *mulop*, in almost all cases the numbers of gates of the computed circuits are not larger.

We compared *mulopII* to *factor*, *factorII*. We ran the experiments with the technology mapping used in [11]. Since the quality of the layouts synthesized by *factorII* approximately equals the quality of the layouts synthesized by *factor* (see [11]), Table 2 only reports the results of the comparison of *mulopII* and *factorII*. (The results concerning *factorII* are those which are published in [11].) Compared to *factorII*, our approach generates realizations with a smaller (or equal) number of gates for almost all circuits. As layout considerations motivate the approach of minimizing communication complexity, we compared layout sizes in the following. A comparison to *factorII* with respect to layout size was not possible because *factorII* and

| | Number of | | No. of gates | | |
|---|---|---|---|---|---|
| Circuit | inputs | outputs | *factorII* | *mulopII* | ratio |
| 9symm1 | 9 | 1 | 75 | 52 | 1.44 |
| cm138a | 6 | 8 | 21 | 21 | 1.00 |
| cm151a | 12 | 2 | 37 | 40 | 0.93 |
| cm162a | 14 | 5 | 80 | 41 | 1.95 |
| cm163a | 16 | 5 | 47 | 36 | 1.31 |
| cm82a | 5 | 3 | 18 | 14 | 1.29 |
| cmb | 16 | 4 | 33 | 34 | 0.97 |
| decod | 5 | 16 | 31 | 32 | 0.97 |
| f51m | 8 | 8 | 107 | 54 | 1.98 |
| x2 | 10 | 7 | 65 | 51 | 1.27 |
| z4m1 | 7 | 4 | 25 | 21 | 1.19 |

**Table 2**: Comparison between our tool *mulopII* and *factorII* with respect to the number of gates used. The technology file used is taken from the paper of Hwang et al. Note that it is different from the technology file in the other comparisons.

the layout tool used in the paper of Hwang et al. was not at our disposal. We used the standard cell place and route package *wolfe* which is integrated in *octtools*. Table 3 shows the comparison between *mulopII*, *misII* and *sis 1.1* with respect to layout size. The technology library used consists of the set of the 2-input gates.‡ For almost two thirds of the benchmark set, our approach dominates (or is as good as) that of *misII* and *sis* with respect to layout size. 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 *misII* and *sis*. However, on the other hand the results confirm the observation already made in [10, 11] that some circuits, e.g., $cm151a$, are not suited for being decomposed with respect to disjoint input partitions. The most dramatic improvement has been obtained for circuit $9symm1$ which is a symmetric function. This confirms the approach of searching equivalence preserving decomposition functions. §

## 6 Conclusion

We have presented a ROBDD based technique of 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 [14] 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.

*The (maximal) value of parameter $h$ of CDF is determined by logarithmic search. More details of how the CDF algorithm is integrated in the tool can be found in [14].

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

‡For the technology file itself see *stdcell2_2.genlib* available in *octtools*.

§Let $f : \{0,1\}^m \to \{0,1\}^n$ be a boolean function which is symmetric in some variables. Then, each equivalence preserving decomposition function of $f$ is symmetric in these variables, too.

| Circuit | No. of gates | | | Running time | | |
|---|---|---|---|---|---|---|
| | *mulop* | *mulopII* | ratio | *mulop* | *mulopII* | ratio |
| 9symm1 | 40 | 45 | 0.89 | 1.40 sec | 1.23 sec | 1.14 |
| C17 | 6 | 7 | 0.86 | 0.32 sec | 0.15 sec | 2.13 |
| cm138a | 20 | 18 | 1.11 | 1.01 sec | 0.18 sec | 5.61 |
| cm151a | 48 | 41 | 1.17 | 4.16 sec | 1.09 sec | 3.82 |
| cm152a | 34 | 27 | 1.26 | 2.15 sec | 0.50 sec | 4.30 |
| cm162a | 46 | 44 | 1.05 | 350.65 sec | 3.32 sec | 105.62 |
| cm163a | 38 | 34 | 1.12 | 2923.31 sec | 2.35 sec | 1243.96 |
| cm82a | 13 | 13 | 1.00 | 0.38 sec | 0.21 sec | 1.81 |
| cm85a | 42 | 42 | 1.00 | 7.46 sec | 3.73 sec | 2.00 |
| cmb | 24 | 29 | 0.83 | 1836.13 sec | 2.52 sec | 728.62 |
| decod | 31 | 28 | 1.11 | 26.15 sec | 2.56 sec | 10.21 |
| f51m | 64 | 56 | 1.14 | 3.14 sec | 1.83 sec | 1.71 |
| majority | 9 | 9 | 1.00 | 0.44 sec | 0.08 sec | 5.50 |
| parity | 15 | 15 | 1.00 | 111.06 sec | 1.37 sec | 81.07 |
| z4m1 | 20 | 20 | 1.00 | 0.66 sec | 0.76 sec | 0.87 |

**Table 1**: Comparison between the (prototype) 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 | Layout size | | | ratio | | Signal delay | | | ratio | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *misII* | *sis* | *mulopII* | *misII* | *sis* | *misII* | *sis* | *mulopII* | *misII* | *sis* |
| 9symm1 | 917928 | 1194336 | 201400 | 4.56 | 5.93 | 26.0 | 27.6 | 13.6 | 1.91 | 2.03 |
| C17 | 28800 | 28800 | 31744 | 0.91 | 0.91 | 4.2 | 4.2 | 4.2 | 1.00 | 1.00 |
| cm138a | 101528 | 103896 | 87480 | 1.16 | 1.19 | 5.8 | 5.8 | 6.8 | 0.85 | 0.85 |
| cm151a | 102528 | 95312 | 177712 | 0.58 | 0.54 | 12.6 | 12.6 | 16.4 | 0.77 | 0.77 |
| cm152a | 90360 | 85536 | 106704 | 0.85 | 0.80 | 10.0 | 10.0 | 13.2 | 0.76 | 0.76 |
| cm162a | 149736 | 131976 | 192000 | 0.78 | 0.69 | 13.8 | 12.0 | 13.2 | 1.05 | 0.91 |
| cm163a | 153272 | 144008 | 164416 | 0.93 | 0.88 | 11.0 | 13.0 | 10.4 | 1.06 | 1.25 |
| cm82a | 83104 | 74784 | 61600 | 1.35 | 1.21 | 8.2 | 7.2 | 7.0 | 1.17 | 1.03 |
| cm85a | 171584 | 165456 | 180000 | 0.95 | 0.92 | 10.2 | 10.2 | 11.0 | 0.93 | 0.93 |
| cmb | 198616 | 204792 | 123496 | 1.61 | 1.66 | 14.8 | 9.4 | 6.8 | 2.18 | 1.38 |
| decod | 133496 | 140448 | 119496 | 1.12 | 1.18 | 6.2 | 6.2 | 5.0 | 1.24 | 1.24 |
| f51m | 536016 | 561184 | 251392 | 2.13 | 2.23 | 51.0 | 51.0 | 18.4 | 2.77 | 2.77 |
| majority | 42200 | 42200 | 39168 | 1.08 | 1.00 | 7.8 | 7.8 | 6.6 | 1.18 | 1.18 |
| parity | 96976 | 99408 | 96976 | 1.00 | 1.03 | 6.2 | 5.0 | 5.0 | 1.24 | 1.00 |
| z4m1 | 176160 | 156288 | 103896 | 1.70 | 1.50 | 18.0 | 16.2 | 9.8 | 1.74 | 1.65 |
| $\sum$ | 2982K | 3228K | 1937K | 1.54 | 1.67 | 205.8 | 198.2 | 147.4 | 1.40 | 1.34 |

**Table 3**: Comparison between *mulopII*, *misII*, and *sis1.1* with respect to layout size, and signal delay.

## 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 *IEEE/ACM 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] G. Hotz. Zur Reduktionstheorie der booleschen Algebra. In *Colloquium über Schaltkreis- und Schaltwerk-Theorie*, 1960.

[10] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

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