Functional Multiple-Output Decomposition: Theory and an Implicit Algorithm

Bernd Wurth* Klaus Eckl Kurt Antreich
Institute of Electronic Design Automation
Technical University of Munich
80290 Munich, Germany

Abstract – *We present theory and a novel, implicit algorithm for functional disjoint decomposition of multiple-output functions. While a Boolean function usually has a huge number of decomposition functions, we show that not all of them are useful for multiple-output decomposition. We therefore introduce the concept of* preferable *decomposition functions, which are sufficient for optimal multiple-output decomposition. We describe how to implicitly compute all preferable decomposition functions of a single-output, and how to identify all common preferable decomposition functions of a multiple-output function. Due to the implicit computation in all steps, the algorithm is very efficient. Applied to FPGA synthesis, the method combines the typically separated steps of common subfunction extraction and technology mapping. Experimental results show significant reductions in area.*

1 Introduction

The task of logic synthesis is to transform a set of Boolean functions into a netlist of library cells. Due to its complexity, logic synthesis is usually partitioned into two steps. The decomposition of combinational logic is a central problem in both steps.

In the first (technology-independent) step, a multiple-level network is created by identifying and extracting common subfunctions [1]. In the second step of logic synthesis, the functions of the network are bound to a library (technology mapping). This may entail a renewed decomposition of complex Boolean functions. If a network is to be bound to k-input look-up tables (LUTs), which are widely used in FPGAs, classical functional decomposition according to the theory of Ashenhurst [2], Roth and Karp [3,4] can be used [5]. Given a single-output function f and a partition of the input variables into bound set variables $\mathbf{x}$ and free set variables $\mathbf{y}$, functional decomposition determines functions $\mathbf{d}(\mathbf{x})$ and $g(\mathbf{z},\mathbf{y})$ such that $f(\mathbf{x},\mathbf{y}) = g(\mathbf{d}(\mathbf{x}),\mathbf{y})$. If, e.g., the bound set has cardinality k, each decomposition function d_i can be implemented in one LUT. Recently, efficient functional decomposition methods based on Binary Decision Diagrams (BDDs) were proposed [6, 7,8]. It was also shown that an optimized encoding of the $\mathbf{x}$-vertices yields simple g-functions [9] and thus reduces area.

If these approaches are used to decompose a multiple-output function, each output must be decomposed individually. Common subfunctions are not recognized. This is shown for a small circuit in Fig. 1 a).

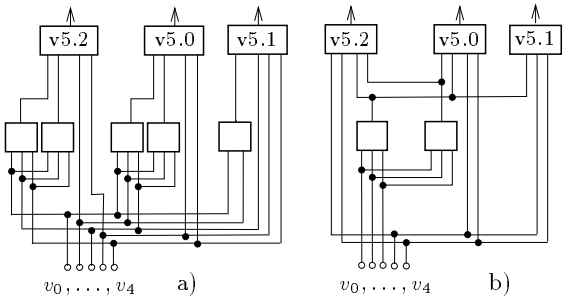

Figure 1: Single-output (a) and multiple-output decomposition (b) of circuit `rd53`, $k = 4$.

Functional decomposition of multiple-output functions has so far not been solved in a satisfactory way. As it combines the previously separated steps of common subfunction extraction and technology mapping, multiple-output decomposition is especially attractive for the synthesis for LUT architectures. Fig. 1 b) shows the example circuit after multiple-output decomposition with our algorithm.

A first approach to multiple-output decomposition was proposed by Karp [4]. It is applicable only for functions with 2 outputs, but it has the advantage that several codes may be assigned to one equivalence class of compatible bound set vertices. If just one code is assigned to each equivalence class (called "strict" decomposition [4]), not all common decomposition functions can be detected. Recent multiple-output decomposition methods [10,11] propose such strict decompositions. A confined non-strict decomposition method, which enforces that all outputs are fed by all decomposition functions, was suggested in [12].

This paper presents new theory and a novel, implicit algorithm for the functional non-strict decomposition of multiple-output functions. Our algorithm is based on the concept of a *preferable decomposition function*, which is a decomposition function for a single output with the potential to be shared by other outputs. We show that the set of bound set vertices can be partitioned into *global classes*, which constitute the elementary blocks needed to construct *all* preferable decomposition functions. Due to the regular structure of most functions appearing in switching applications, the number of global classes is usually much smaller than the number of bound set vertices.

However, the number of preferable decomposition functions can still be huge. Recently, exact solutions to other CAD problems with huge sets of objects have been devised using implicit techniques [13,14]. A set of objects is represented as a minterm using the positional-set notation, and a set of sets of objects is represented by its *characteristic function*. In our method, we use global classes as elementary objects. A set of global classes defines a decomposition function and is represented by

*The first author was supported by an Ernst von Siemens-grant given by Siemens AG.

32nd ACM/IEEE Design Automation Conference ®

a positional minterm. The set of all preferable decomposition functions of an individual function is represented by a Boolean formula. Using BDDs, we are thus able to represent sets of preferable decomposition functions as Boolean formulas in a very compact way.

In this paper, we furthermore describe how to implicitly compute all preferable decomposition functions, and how to identify the common preferable decomposition functions of a multiple-output function. The BDD-based, implicit decomposition algorithm is very efficient. We apply it to the synthesis of Xilinx XC3000 circuits. Experimental results show significant reductions in area.

The remainder of this paper is organized as follows. After preliminaries on Boolean functions and partitions in Section 2, Section 3 summarizes the classical theory of single-output decomposition. In Section 4, we discuss the partition of bound-set variable vertices into global classes. We introduce preferable decomposition functions in Section 5. The theory developed so far is applied in Section 6, where our implicit algorithm for computing all preferable decomposition functions is described. Experimental results on various benchmarks are reported in Section 7.

2 Preliminaries

A *single-output Boolean function* is given by $f : \{0,1\}^n \rightarrow \{0,1\}$. A *multiple-output Boolean function* is a vector of single-output Boolean functions and is denoted by a bold letter, $\mathbf{f} = (f_1, \ldots, f_m)$. Vectors of *Boolean variables* x_i are also printed bold, $\mathbf{x} = (x_1, \ldots, x_n)$.

A *partition* Π of the set $\mathcal{X}$ of all bound set vertices $\mathbf{x}$ divides the set into disjoint *blocks* or *classes*. Let R be an equivalence relation on $\mathcal{X}$. Then, the set of equivalence classes under R is a partition of $\mathcal{X}$, denoted by $\mathcal{X}/R$. Let $\Pi_1 = \mathcal{X}/R_1$ and $\Pi_2 = \mathcal{X}/R_2$ be partitions of $\mathcal{X}$. Then Π_2 *refines* Π_1 if every block of Π_2 is contained in a block of Π_1. Equivalently, Π_2 *refines* Π_1 iff $R_2 \subseteq R_1$. Let Π_1 and Π_2 be partitions of $\mathcal{X}$. The *product* partition Π of Π_1 and Π_2, denoted $\Pi = \Pi_1 \cdot \Pi_2$, is the partition of $\mathcal{X}$ which has the smallest number of blocks and refines both Π_1 and Π_2. The product Π of c partititons Π_i is

$$\Pi = \prod_{i=1}^{c} \Pi_i.$$

3 Single-Output Decomposition

We summarize the classical decomposition theory of Ashenhurst [2], Roth and Karp [3,4]. Only disjoint decomposition of completely specified functions will be considered.

Given a function $f(\mathbf{x}, \mathbf{y})$ and a partition of its n input variables into the bound set $BS = \{x_1, \ldots, x_b\}$ and the free set $FS = \{y_1, \ldots, y_{n-b}\}$, functional decomposition determines *decomposition functions* $d_1, \ldots, d_c$ and the *composition function* g such that $f(x_1, \ldots, x_b, y_1, \ldots, y_{n-b}) = g(d_1(x_1, \ldots, x_b), \ldots, d_c(x_1, \ldots, x_b), y_1, \ldots, y_{n-b})$. We are only interested in non-trivial decompositions, i.e., $c < b$.

Let $\mathcal{X} = \{0,1\}^b$ denote the set of all BS-vertices. The existence of a decomposition can be determined using a relation of compatibility between elements of $\mathcal{X}$.

Definition 1 *Two bound set vertices* $\mathbf{x}_\alpha \in \mathcal{X}$ *and* $\mathbf{x}_\beta \in \mathcal{X}$ *are* **compatible**, *denoted* $\mathbf{x}_\alpha R_f \mathbf{x}_\beta$, *iff*

$$\forall \mathbf{y} \in \{0,1\}^{n-b}: \quad f(\mathbf{x}_\alpha, \mathbf{y}) = f(\mathbf{x}_\beta, \mathbf{y}).$$

For completely specified functions, compatibility is an equivalence relation. The relation R_f induces a *local compatibility partition* $\Pi_f = \mathcal{X}/R_f = \{L_1, \ldots, L_\ell\}$ of the BS-vertices $\mathbf{x}$ into ℓ equivalence classes. These classes, which are associated with a single-output function, will be called *local classes* later on.

The *decomposition chart* visualizes compatibility of BS-vertices. The decomposition chart is a Karnaugh map where rows are associated with FS-vertices and columns are associated with BS-vertices. Two BS-vertices $\mathbf{x}_\alpha$ and $\mathbf{x}_\beta$ are compatible if their column patterns are identical. The column multiplicity is equal to the number ℓ of equivalence classes.

Example 1 The decomposition chart of a function f_1 shown in Fig. 2 has 3 distinct columns. The compatibility partition $\Pi_{f_1} = \mathcal{X}/R_{f_1} = \{L_1, L_2, L_3\}$ is depicted in Fig. 3 a), where $L_1 = \{000, 001, 010, 100\}$, $L_2 = \{011, 101, 110\}$, $L_3 = \{111\}$.

The classical *decomposition condition* according to Roth and Karp states that a decomposition exists iff for all pairs of BS-vertices $\mathbf{x}_\alpha \in \mathcal{X}$ and $\mathbf{x}_\beta \in \mathcal{X}$:

$$\neg(\mathbf{x}_\alpha R_f \mathbf{x}_\beta) \Longrightarrow \mathbf{d}(\mathbf{x}_\alpha) \neq \mathbf{d}(\mathbf{x}_\beta). \tag{1}$$

The decomposition functions d_i may evaluate to identical or different *codes* $\mathbf{d}(\mathbf{x})$ for compatible BS-vertices, but they must evaluate to different codes for incompatible BS-vertices. The smallest number of decomposition functions, called *codewidth* c, for (1) to be satisfied is $c = \lceil \mathrm{ld}\, \ell \rceil$. We will always choose this minimum value of c as this minimizes the number of decomposition functions and the number of inputs of g.

Example 2 In Example 1, $\ell = 3 \Rightarrow c = 2$. A valid choice of decomposition functions is $d_1(\mathbf{x}) = x_1x_2x_3 + x_1\overline{x}_2\overline{x}_3$, $d_2(\mathbf{x}) = x_1\overline{x}_3 + \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3$. We thus have $\mathbf{d}(\mathbf{x}) = (00)$ for $\mathbf{x} \in \{000, 001, 010\}$, $\mathbf{d}(\mathbf{x}) = (01)$ for $\mathbf{x} \in \{011, 101, 110\}$, $\mathbf{d}(\mathbf{x}) = (10)$ for $\mathbf{x} \in \{111\}$, and $\mathbf{d}(\mathbf{x}) = (11)$ for $\mathbf{x} \in \{100\}$.

The decomposition functions evaluate to two codes, (00) and (11), for the compatible BS-vertices of class L_1. Therefore, this decomposition is called *non-strict* [4].

Obviously, the decomposition functions define a partition of the BS-vertices into classes with identical code. This partition is illustrated by dashed and dotted lines in Fig. 3 b). More formally, each individual decomposition function d_i defines a partition $\Pi_{d_i} = \mathcal{X}/R_{d_i}$ of the BS-vertices into onset and offset, where R_{d_i} relates two vertices if they belong both to either the onset or the offset of d_i. Forming the product of all these partitions Π_{d_i}, the resulting partition has blocks of vertices with identical code. The decomposition condition (1) can thus be restated using partitions of $\mathcal{X}$:

Decomposition Condition 1 *A functional decomposition of the single-output function f by a set of decomposition functions $\{d_1, \ldots, d_c\}$ exists iff*

$$\prod_{i=1}^{c} \Pi_{d_i} \quad \textit{refines} \quad \Pi_f.$$

$x_1x_2x_3$

y_1y_2	000	001	010	011	100	101	110	111
00	0	0	0	1	0	1	1	1
01	1	1	1	1	1	1	1	0
10	1	1	1	1	1	1	1	0
11	0	0	0	1	0	1	1	0

a)

	000	001	010	011	100	101	110	111
	0	0	0	1	0	1	0	1
	0	1	1	1	1	1	1	0
	0	1	1	1	1	1	1	0
	1	1	1	0	1	0	1	0

b)

Figure 2: Decompositon chart of a) f_1 and b) f_2.

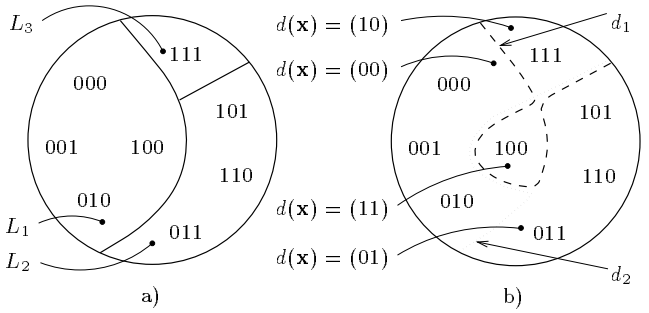

Figure 3: Partition of $\mathcal{X}$ a) by Π_{f_1} (solid lines) and b) by Π_{d_1} (dashed line) and Π_{d_2} (dotted line).

4 Global Partition

We state the multiple-output decomposition condition in a similiar way:

Decomposition Condition 2 *A functional decomposition of the multiple-output function* $\mathbf{f} = (f_1, \ldots, f_m)$ *by a set of decomposition functions* $P_\mathbf{f} = \{d_1, \ldots, d_q\}$ *exists iff for each output* f_k *there is a subset* $P_{f_k} \subseteq P_\mathbf{f}$ *of cardinality* c_k *such that*

$$\prod_{d \in P_{f_k}} \Pi_d \text{ refines } \Pi_{f_k}, \tag{2}$$

where $c_k = \lceil \mathrm{ld}\, \ell_k \rceil$ *is the codewidth and* ℓ_k *is the number of local classes of* Π_{f_k}. *The total number of decomposition functions is denoted by* q.

Note that the Decomposition Condition 2 is satisfied if the Decomposition Condition 1 is satisfied for each output. In the worst case, however, there is no decomposition function shared by several outputs, then $q = \sum_{k=1}^{m} c_k$. Our problem then is:

Problem 1 *Given a multiple-output function* $\mathbf{f}(\mathbf{x}, \mathbf{y})$, *determine a* minimum *set* $P_\mathbf{f}$ *of decomposition functions such that Decomposition Condition 2 is fulfilled.*

According to the laws of set algebra, $A \subseteq B \wedge C \subseteq D \Longrightarrow (A \cap C) \subseteq (B \cap D)$, we take the product of all partitions on the left as well as on the right side of (2), and obtain

$$\prod_{i=1}^{q} \Pi_{d_i} \text{ refines } \prod_{k=1}^{m} \Pi_{f_k}. \tag{3}$$

Definition 2 *The product of the local compatibility partitions* Π_{f_k} *is called the global compatibility partition, short* **global partition** $\hat{\Pi}$, *which partitions the bound set vertices into* p **global classes**:

$$\hat{\Pi} = \{G_1, \ldots, G_p\} := \prod_{k=1}^{m} \Pi_{f_k}.$$

The global partition partitions the BS-vertices into global classes such that the vertices of a class are compatible for each individual output f_k.

Example 3 Let $\mathbf{f} = (f_1, f_2)$, where f_1 and f_2 are the functions of Fig. 2. The local compatibility partitions are $\Pi_{f_1} = \{L_1^1, L_2^1, L_3^1\}$ with $L_1^1 = \{000, 001, 010, 100\}$, $L_2^1 = \{110, 011, 101\}$, $L_3^1 = \{111\}$, and $\Pi_{f_2} = \{L_1^2, L_2^2, L_3^2, L_4^2\}$ with $L_1^2 = \{000\}$, $L_2^2 = \{001, 010, 100, 110\}$, $L_3^2 = \{011, 101\}$ and $L_4^2 = \{111\}$.

We have $\ell_1 = 3 \Rightarrow c_1 = 2$ and $\ell_2 = 4 \Rightarrow c_2 = 2$. The global partition is then given by $\hat{\Pi} = \{G_1, G_2, G_3, G_4, G_5\}$ with $G_1 = \{000\}$, $G_2 = \{001, 010, 100\}$, $G_3 = \{110\}$, $G_4 = \{011, 101\}$ and $G_5 = \{111\}$. For the local classes of Π_{f_1}, $L_1^1 = G_1 \cup G_2$, $L_2^1 = G_3 \cup G_4$ and $L_3^1 = G_5$. The function $\mathbf{f}$ can be decomposed with the following three d-functions, $d_1(\mathbf{x}) = \overline{x}_1 x_3 + x_2 \overline{x}_3 + x_1 \overline{x}_2$, $d_2(\mathbf{x}) = \overline{x}_1 \overline{x}_2 x_3 + x_2 \overline{x}_3 + x_1 \overline{x}_3 + x_1 x_2$, $d_3(\mathbf{x}) = \overline{x}_1 \overline{x}_2 x_3 + \overline{x}_1 x_2 \overline{x}_3 + x_1 \overline{x}_2 \overline{x}_3 + x_1 x_2 x_3$, such that $f_1 = g_1(d_1(\mathbf{x}), d_3(\mathbf{x}), \mathbf{y})$ and $f_2 = g_2(d_1(\mathbf{x}), d_2(\mathbf{x}), \mathbf{y})$. Decomposition function d_1 is shared among both outputs.

We will now show the relevance of functions which can be constructed from global classes. Such functions are defined as *constructable*:

Definition 3 *Given a global partition* $\hat{\Pi} = \{G_1, \ldots, G_p\}$, *a function* $d(\mathbf{x}) : \{0,1\}^b \rightarrow \{0,1\}$ *is called* **constructable** *with respect to* $\hat{\Pi}$ *iff each global class* G_i *is completely contained in either the onset or the offset of* d.

In Example 3, the decomposition functions d_1, d_2, d_3 are constructable with respect to the given global partition $\hat{\Pi}$. E.g., $d_1(\mathbf{x}) = 1 \Leftrightarrow \mathbf{x} \in (G_2 \cup G_3 \cup G_4)$. Note that d_1 and d_2 in Example 2 are not constructable. The following theorem is central to our theory:

Theorem 1 *Any set* $P_\mathbf{f}$ *containing non-constructable decomposition functions can be replaced by a set* $P'_\mathbf{f}$ *of constructable decomposition functions without increasing the number* q *of required decomposition functions.*

Especially, every set of non-constructable decomposition functions corresponding to an optimum solution can be replaced by a set of constructable decomposition functions. Therefore, an optimum decomposition with a minimum number of decomposition functions can be obtained by choosing only constructable decomposition functions.

Therefore it suffices to consider only constructable functions as decomposition functions without detriment to the final number of decomposition functions. To build a constructable function $d(\mathbf{x})$, each global class is assigned to either the onset or the offset. Then, the number of constructable functions is 2^p. We derive another useful property from (3):

Property 1 *The number* p *of global classes determines a lower bound on the number* q *of decomposition functions:* $\lceil \mathrm{ld}\, p \rceil \leq q$.

For Example 3 we have $p = 5 \Rightarrow q \geq 3$. The found solution with $q = 3$ therefore is an optimum solution w.r.t. the number of decomposition functions. This property allows to abort multiple-output decomposition at an early stage if the number of shared decomposition functions is too small (this may be due to a bad variable partitioning).

5 Preferable Decomposition Functions

Constructable functions are not necessarily suitable for decomposition. A trivial example is the function $d(\mathbf{x}) = 0$, which is constructable with respect to any global partition. Thus, a property is needed which captures the suitability of a function for decomposition.

Karp [4] gave conditions for a function $d(\mathbf{x}) : \{0,1\}^b \rightarrow \{0,1\}$ to be suitable for single-output decomposition. His definitions of *(partial) assignment* and *assignable* relate to a decomposition procedure which iteratively selects the decomposition functions for a single-output function. An *assignment* P_f is a set of decomposition functions $\{d_1, \ldots, d_c\}$ which satisfies the decomposition condition. A set of decomposition functions $\{d_1, \ldots, d_s\}$, $s \leq c$, is a *partial assignment* if there exist decomposition functions $\{d_{s+1}, \ldots, d_c\}$ such that the union of both sets is an assignment. We restate the definition of a partial

assignment in terms of partitions, and then use this definition to define the property *assignable*:

Definition 4 *Given function f, a set of decomposition functions $\{d_1, \ldots, d_s\}$, $s \leq c$, is a* **partial assignment** *$P_{f,s}$ iff the product $\Pi_{P_{f,s}}$ of the partitions induced by these functions,*

$$\Pi_{P_{f,s}} := \prod_{i=1}^{s} \Pi_{d_i},$$

called the **partial partition**, *has no block which contains BS vertices of more than 2^{c-s} local classes of the compatibility partition Π_f.*

Definition 5 *A function d is* **assignable** *with respect to a given partial assignment $P_{f,s}$ iff $P_{f,s} \cup d$ is a partial assignment $P_{f,s+1}$.*

Two aspects need to be emphasized here. First, assignability is a property which is in no way related to the multiple-output decomposition problem, but only to the fulfillment of the decomposition condition for a single-output. Second, assignability is always related to an already selected set of (assignable) decomposition functions, i.e., a partial assignment $P_{f,s}$. Therefore, a function which is assignable at an early stage of the decomposition procedure is not necessarily assignable at a later stage. As an example, both the function d_1 in Example 2 and the (different) function d_1 in Example 3 are assignable for f_1 if $P_{f_1,0} = \emptyset$. After selecting either of them as a partial assignment, the other one is not assignable any more.

We now have two properties of decomposition functions. A decomposition function must be assignable to satisfy the basic decomposition condition for a single-output. Constructability is sufficient for optimal sharing of decomposition functions. Decomposition functions with both properties are called *preferable*:

Definition 6 *A function $d(\mathbf{x}) : \{0,1\}^b \rightarrow \{0,1\}$ is* **preferable** *for a single output f_k of a multiple-output function $\mathbf{f}$ iff two conditions are satisfied. First, it must be assignable with respect to the output f_k and a partial assignment $P_{f_k,s}$. Second, it must be constructable with respect to the global partition determined by $\mathbf{f}(\mathbf{x}, \mathbf{y})$.*

6 Implicit Computation

To handle the large number of preferable decomposition functions, we must represent them efficiently.

Since preferable decomposition functions are constructable by definition, we employ a bijective mapping from the set of constructable functions to $\{0,1\}^p$. A vertex $\mathbf{z} = (z_1, \ldots, z_p) \in \{0,1\}^p$ then represents a constructable function d in *positional-set* form. The variable z_i assumes value 1 if the global class G_i is contained in the onset of d, and it assumes value 0 if the global class G_i is contained in the offset of d.

Example 4 For the function f_1 in Example 3, $d_1(\mathbf{x}) = 1 \Leftrightarrow \mathbf{x} \in G_2 \cup G_3 \cup G_4$. The number of global classes is $p = 5$. The function d_1 is then represented in positional-set form by the vertex $\mathbf{z} = (01110)$.

A set S of constructable functions is represented as a set of z-vertices by a characteristic function $\chi : \{0,1\}^p \rightarrow \{0,1\}$. We have $\chi(\mathbf{z}) = 1$ iff the constructable function represented by $\mathbf{z}$ is in the set S. A set of preferable decomposition functions is represented by its characteristic function and thus in a single BDD. However, this compact representation is valuable only if it is also possible to compute all preferable decomposition functions in an implicit way. Since preferability includes assignability, this computation must be performed for each output of the multiple-output function $\mathbf{f}$. Furthermore, since assignability depends on a specific partial assignment, preferable decomposition functions must be recomputed after updating a partial assignment.

We will now describe a procedure to compute the set of preferable decomposition functions, represented by its characteristic function $\chi_k(\mathbf{z})$, for output k. For ease of notation, we explain the procedure for $P_{f_k,0} = \emptyset$, i.e., no decomposition function has been assigned so far.

From Definitions 4 and 5 it follows that a function d is assignable for $P_{f_k,0}$, if neither onset nor offset of d contain vertices of more than 2^{c_k-1} local classes. Equivalently, function d is assignable if there are at least $\ell_k - 2^{c_k-1}$ local classes that completely belong to the onset (condition C1), and if there are at least $\ell_k - 2^{c_k-1}$ local classes that completely belong to the offset (condition C0) of d.

We introduce auxiliary variables $v_1, \ldots, v_{\ell_k}$ to represent subsets of Π_{f_k} (the set of ℓ_k local classes) in positional-set form. In a first step, the set of all subsets of Π_{f_k} containing at least $\ell_k - 2^{c_k-1}$ local classes is implicitly computed using the *subset algorithm* of Fig. 4. The characteristic function for this set is $\tau_k(\mathbf{v}) = subset(\ell_k - 2^{c_k-1}, \ell_k)$. Note that this characteristic function $\tau_k(\mathbf{v})$ is a threshold function which evaluates to 1 iff at least $\ell_k - 2^{c_k-1}$ out of ℓ_k variables take value 1. The complexity of $subset(\delta, \ell)$ is $O(\delta \cdot \ell)$.

From $\tau_k(\mathbf{v})$ two functions $\psi_k^0(\mathbf{z})$ and $\psi_k^1(\mathbf{z})$ are derived, where $\psi_k^0(\mathbf{z})$ represents all decomposition functions satisfying condition C0 and $\psi_k^1(\mathbf{z})$ represents all functions satisfying condition C1. We obtain function $\psi_k^0(\mathbf{z})$ by replacing each v-literal (which represents a local class) in $\tau_k(\mathbf{v})$ by the conjunction of the negative z-literals that represent the global classes contained in the local one. Similarly, we get $\psi_k^1(\mathbf{z})$ by replacing each v-literal in $\tau_k(\mathbf{v})$ by the conjunction of positive z-literals. The product of both functions and $\overline{z}_1$ yields the characteristic function $\chi_k(\mathbf{z}) = \overline{z}_1 \cdot \psi_k^0(\mathbf{z}) \cdot \psi_k^1(\mathbf{z})$ representing the preferable decomposition functions. Multiplication of $\overline{z}_1$ eliminates complementary decomposition functions.

Example 5 We again consider the function f_1 of Example 3, $\ell_1 - 2^{c_1-1} = 3 - 2^{2-1} = 1$. We have $\tau_1(\mathbf{v}) = v_1 + v_2 + v_3$. For $\psi_1^1(\mathbf{z})$, literal v_1 is replaced by $z_1 z_2$ since $L_1^1 = G_1 \cup G_2$, literal v_2 by $z_3 z_4$ since $L_2^1 = G_3 \cup G_4$, and literal v_3 by z_5 since $L_3^1 = G_5$. Similarly, for $\psi_1^0(\mathbf{z})$ literal v_1 is replaced by $\overline{z}_1 \overline{z}_2$ etc. Then $\chi_1(\mathbf{z}) = \overline{z}_1\overline{z}_2 z_3 z_4 + \overline{z}_1 z_3 z_4 \overline{z}_5 + \overline{z}_1 \overline{z}_2 z_5 + \overline{z}_1 \overline{z}_3 \overline{z}_4 z_5$. For function f_2 in Example 3, $\chi_2(\mathbf{z}) = \overline{z}_1 z_2 z_3 z_4 \overline{z}_5 + \overline{z}_1 z_2 z_3 \overline{z}_4 z_5 + \overline{z}_1 \overline{z}_2 \overline{z}_3 z_4 z_5$. The functions $\chi_k(\mathbf{z})$ are shown as row vectors in Fig. 5.

For $s \neq 0$, the partial partition $\Pi_{P_{f_k,s}}$ itself consists of several blocks. Then the above procedure must be applied for each block and $\chi_k(\mathbf{z})$ is built by the product of all obtained characteristic functions.

Having computed the characteristic functions $\chi_k(\mathbf{z})$ of each output k, the task is now to find a z-vertex contained in the on-

```
t_0 := 1
for j := 1 to δ
    t_j := 0
for i := 1 to ℓ
    for j := δ to 1
        t_j := t_j + t_{j-1} · v_i
return t_δ(v_1, ..., v_ℓ)
```

Figure 4: $subset(\delta, \ell)$ algorithm.

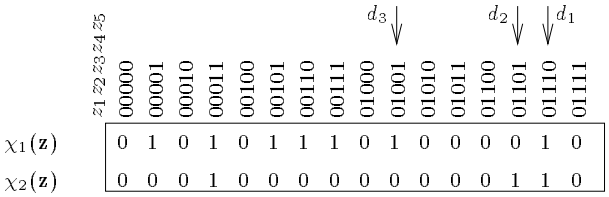

Figure 5: Preferable d-functions for f_1 and f_2.

set of a maximum number of $\chi_k(\mathbf{z})$. Such a vertex corresponds with a decomposition function which is preferable for a maximum number of outputs. This problem is solved implicitly by the *Lmax* algorithm suggested by Kam [14].

Example 6 Fig. 5 visualizes this problem as finding a column with a maximum number of 1's in a covering table. There are two $\mathbf{z}$-vertices which are in the onset of both characteristic functions $\chi_1(\mathbf{z})$ and $\chi_2(\mathbf{z})$. The vertex $\overline{z}_1z_2z_3z_4\overline{z}_5$ is chosen. It represents the subset of global classes $\{G_2, G_3, G_4\}$ and thus the preferable function $d(\mathbf{x}) = \overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2\overline{x}_3 + \overline{x}_1x_2x_3 + x_1\overline{x}_2\overline{x}_3 + x_1\overline{x}_2x_3 + x_1x_2\overline{x}_3$. This is the function d_1 used in Example 3.

The decomposition function, which is derived from the chosen vertex $\mathbf{z}$, is a partial assignment for all outputs k with $\chi_k(\mathbf{z}) = 1$. The characteristic functions of the affected outputs are recomputed, taking into account the given partial assignment. Generally, the number of preferable functions decreases with each recomputation. The algorithm stops if the partial assignment of each output constitutes an assignment.

Example 7 In the example, the further $\mathbf{z}$-vertices chosen are $\overline{z}_1z_2z_3\overline{z}_4z_5$, corresponding with decomposition function d_2, and $\overline{z}_1z_2\overline{z}_3\overline{z}_4z_5$, which corresponds with decomposition function d_3 in Example 3.

7 Implementation and Experiments

Implementation Aspects. The implicit algorithm was implemented in program IMODEC (Implicit Multiple-Output DEComposition), which is embedded into the synthesis tool TOS. Before IMODEC is applied, the output and the variable partitioning problem must be solved.

Output partitioning concerns the problem of grouping functions in a network to function vectors $\mathbf{f}$. We use a greedy heuristic which initializes the function vector with the function having a maximum number of inputs. Then, a function which has a maximum number of inputs in common with the current vector is combined with it. Multiple-output decomposition is performed for the current $\mathbf{f}$. If the decomposition gain in comparison to Single-output decomposition of each f_k decreased by the last combination, the combination is undone. This is repeated until no further suitable function remains to be combined.

Variable partitioning is solved heuristically [15]. In the sequel, CPU times are given in seconds for a DECstation 3000/500.

Problem Characteristics. We first demonstrate the reduction of complexity by the concept of perferable decomposition functions. Table 1 shows characteristical data on multiple-output decompositions of some function vectors, which occured during the decomposition of circuits `f51m`, `alu4` and `term1`.

The name of the function vector and the number of outputs m are given in the first column. The cardinality b of the bound set, the number of local classes ℓ_k of each output and the number of global classes p are given in columns 2 to 4. The number

Table 1: Characteristics of decompositions

f	b	ℓ_k	p	# assign.	# prefer.	CPU/sec
$\mathbf{f}_{\text{f51m}}$	5		5	$(4.3 \cdot 10^9)$	(32)	0.167
m=3		2		2	2	
		4		6	6	
		5		$1.3 \cdot 10^7$	30	
$\mathbf{f}_{\text{alu4}}$	8		32	$(1.2 \cdot 10^{77})$	$(4.3 \cdot 10^9)$	12.757
m=3		24		$2.1 \cdot 10^{48}$	$3.1 \cdot 10^9$	
		25		$8.8 \cdot 10^{44}$	$2.8 \cdot 10^9$	
		26		$1.4 \cdot 10^{44}$	$2.6 \cdot 10^9$	
$\mathbf{f}_{\text{term1}}$	7		64	$(3.4 \cdot 10^{38})$	$(1.8 \cdot 10^{19})$	72.937
m=6		12		$2.2 \cdot 10^{38}$	$1.4 \cdot 10^{19}$	
		32		$6.0 \cdot 10^8$	$6.0 \cdot 10^8$	
		63		$3.4 \cdot 10^{37}$	$2.8 \cdot 10^{18}$	
		63		$3.4 \cdot 10^{37}$	$2.8 \cdot 10^{18}$	
		63		$3.4 \cdot 10^{37}$	$2.8 \cdot 10^{18}$	
		63		$3.4 \cdot 10^{37}$	$2.8 \cdot 10^{18}$	

of assignable functions # *assign.* and of preferable functions # *prefer.* is shown. The values in parenthesis give an upper bound on the number of functions, which is 2^{2^b} and 2^p respectively. The CPU time includes calculation of local and global classes, implicit computation of characteristic functions $\chi_k(\mathbf{z})$ for each output, choosing a perferable function, re-calculating $\chi_k(\mathbf{z})$ for each affected output until complete assignments are determined for each output.

It can be seen that the number p of global classes is mostly much smaller than its theoretical upper bound given by $min((\prod_{k=1}^{m} \ell_k), 2^b)$, indicating that the individual outputs of $\mathbf{f}$ are closely related. Memory consumption and CPU time are largely determined by p. Decompositions with less than 25 global classes are usually performed in less than one second.

The current bottleneck of our method is the construction of the covering table for the *Lmax* algorithm. Depending on the actual characteristic functions χ_k, the method may become very expensive for $p \geq 50$. In these cases we can limit m to make decomposition tractable. However, since an increase in the number of outputs always implies a degraded variable partitioning for each individual output, thus neutralizing the increased sharing of subfunctions by the increased codewidth, the values of m and thus also p are small in most practical cases. Maximum values of m and p during decomposition of benchmark circuits are given in Table 2.

The number of preferable functions is much smaller than the number of assignable functions. However, the number of preferable functions can still be very large. Our implicit algorithm makes it possible to handle such large numbers of decomposition functions in short CPU times.

Technology Mapping for Xilinx XC3000. We target the Xilinx XC3000 architecture, which has Configurable Logic Blocks (CLBs) made up of 5-input LUTs.

In the first experiment, we compared the multiple-output decomposition with functional single-output decomposition. The input to the program was a collapsed network (circuits marked with an $*$ could not be collapsed). After decomposition the node functions were assigned to CLBs as permitted by the XC3000 technology. The number of CLBs is given in column *IMODEC-CLB*. The maximum values of the number m of function outputs and the number p of global classes, that occured during multiple-output decomposition, are shown in column *IMODEC-m/p*. For comparison, we also decomposed the collapsed networks with our program in a mode which only performs single-output decomposition (column *Single*).

Comparing columns 3 and 4 of Table 2, it is apparent that multiple-output decomposition drastically outperforms single-output decomposition. We achieve an average reduction of the

Table 2: Mapping to Xilinx XC3000 CLBs

net	IMODEC		Single	r+IMODEC		r+FGMap
	$m/\ p$	CLB	CLB	CLB	CPU	CLB
5xp1	5/ 5	9	15	9	3.1	15
9sym	1/ 6	7	7	7	0.9	7
alu2	4/40	46	47	46	902.2	53
alu4	6/49	168	235	-	-	-
apex6	17/30	141	174	129	3.3	-
apex7	10/15	44	61	41	3.6	47
clip	5/14	12	19	12	17.3	20
count	8/ 3	26	35	26	49.1	24
des*	-	-	-	489	582.4	-
duke2	5/54	177	311	122	28.9	-
e64	12/ 3	123	329	55	3.5	55
f51m	3/ 5	8	13	8	1.7	11
misex1	3/ 8	9	11	9	1.9	8
misex2	5/ 7	28	34	21	1.4	21
rd73	3/ 6	5	7	5	1.4	7
rd84	4/ 6	8	11	8	3.9	12
rot*	-	-	-	127	2.1	194
sao2	4/11	17	24	17	17.4	27
vg2	5/12	41	64	19	3.3	23
z4ml	2/ 3	4	4	4	0.6	5
C499*	-	-	-	50	0.0	49
C880*	-	-	-	81	8.6	74
C5315*	-	-	-	295	2.1	-
$\sum$(sub)	-	873	1401	545	-	652

CLB count of 38%. There are some circuits, as e.g. `9sym`, which are optimally decomposed as trees. In these cases, no decomposition functions are shared and multiple-output decomposition does not yield an advantage. Surprisingly, the maximum values for p are small in most cases although functions with up to 17 outputs are decomposed. Only for the circuit `alu4` we had to limit m as discussed above.

In a second experiment, we compared our multiple-output decomposition procedure with a BDD-based single-output decomposition algorithm recently published by Lai et al. [6,7], which seems to be state-of-the-art in decomposition for LUT-architectures.

Results for IMODEC are given in column *r+IMODEC*. Results from [7] are repeated in column *r+FGMap*. As recommended in [7], large circuits are pre-structured with the SIS-script *script.rugged* [16]. The subtotal in column 5 gives the sum for those circuits only for which results are given in column 7. IMODEC outperforms FGMap by about 16%. However, IMODEC has often no advantage over single-output decomposition if a pre-structured network is the starting point since many nodes already have 5 or less inputs. This is also the reason for the small CPU times needed to decompose some of the large circuits.

The relatively large CPU time needed for circuit `alu2` results from the greedy output partitioning approach, which performs many trial decompositions until the best one is selected. Thus, better output partitioning approaches with less trial decompositions would reduce computation times significantly.

8 Conclusion

We have extended the classical functional decomposition theory to multiple-output functions. We introduced the concept of *preferable* decomposition functions. These functions are both *constructable* in terms of global classes and *assignable*. While assignability relates to the general suitability of a function as decomposition function, constructability expresses its usefulness for multiple-output decomposition.

Although this concept drastically reduces the number of decomposition functions to be considered, the number of preferable functions may still be very large. We developed an implicit, BDD-based algorithm for multiple-output decomposition which is able to efficiently deal with large sets of preferable functions.

Applied to the synthesis for LUT-architectures, experimental results indicate improvements of 38% over single-output decomposition and improvements of 16% in combination with standard logic synthesis techniques.

References

[1] R. K. Brayton, R. L. Rudell, and A. L. Sangiovanni-Vincentelli, "MIS: A Multiple-Level Logic Optimization System," *IEEE Transactions on Computer-Aided Design*, vol. 6, no. 6, pp. 1062–1081, 1987.

[2] R. L. Ashenhurst, "The Decomposition of Switching Functions," *Ann. Computation Lab. of Harvard Univ.*, vol. 29, pp. 74–116, 1959.

[3] J. P. Roth and R. M. Karp, "Minimization Over Boolean Graphs," *IBM Journal*, pp. 227–238, 1962.

[4] R. M. Karp, "Functional Decomposition and Switching Circuit Design," *J. Soc. Indust. Appl. Math.*, vol. 11, no. 2, pp. 291–335, 1963.

[5] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays," *27th ACM/IEEE Design Automation Conference, DAC*, pp. 620–625, June 1990.

[6] Y. Lai, M. Pedram, and S. Vrudhula, "BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis," *30th ACM/IEEE Design Automation Conference, DAC*, pp. 642–647, June 1993.

[7] Y.-T. Lai, K.-R. R. Pan, M. Pedram, and S. Sastry, "FGMap: A Technology Mapping Algorithm for Look-Up Table Type FPGAs Based on Function Graphs," *Workshop Notes International Workshop on Logic Synthesis IWLS*, pp. 9b1 – 9b4, May 1993.

[8] T. Sasao, *Logic Synthesis and Optimization*. Boston / London / Dordrecht: Kluwer Academic Publishers, 1993.

[9] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimum Functional Decomposition Using Encoding," *31th ACM/IEEE Design Automation Conference, DAC*, pp. 408–414, 1994.

[10] Y.-T. Lai, K.-R. R. Pan, and M. Pedram, "FPGA Synthesis using OBDD-based Function Decomposition," *USC Technical Report*, 1994.

[11] P. Molitor and C. Scholl, "Communication Based Multilevel Synthesis for Multi-Output Boolean Functions," *4th Great Lakes Symposium on VLSI, Indiana*, 1994.

[12] Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula, "EVBDD-Based Algorithms for Integer Linear Programming, Spectral Transformation, and Function Decomposition," *IEEE Transactions on Computer-Aided Design*, vol. 13, no. 8, 1994.

[13] O. Coudert, J. C. Madre, and H. Fraisse, "A New Viewpoint on Two-Level Minimization," *30th ACM/IEEE Design Automation Conference, DAC*, 1993.

[14] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, "A Fully Implicit Algorithm for Exact State Minimization," *31th ACM/IEEE Design Automation Conference DAC*, 1994.

[15] U. Schlichtmann, "Disjunkte Dekomposition Boolescher Funktionen: Eine Neue Betrachtungsweise," *Tagungsband 6. E.I.S.-Workshop*, pp. 319 – 328, Nov. 1993.

[16] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," *IEEE/ACM International Conference on Computer-Aided Design, ICCAD*, pp. 328–333, Oct. 1992.