Functional Multiple-Output Decomposition with Application to Technology Mapping for Lookup Table-Based FPGAs

BERND WURTH and ULF SCHLICHTMANN
Infineon Technologies AG
and
KLAUS ECKL and KURT J. ANTREICH
Technical University of Munich

Functional decomposition is an important technique for technology mapping to lookup table-based FPGA architectures. We present the theory of and a novel approach to functional disjoint decomposition of multiple-output functions, in which common subfunctions are extracted during technology mapping.

While a Boolean function usually has a very large number of subfunctions, we show that not all of them are useful for multiple-output decomposition. We use a partition of the set of bound set vertices as the basis to compute *preferable* decomposition functions, which are sufficient for an optimal multiple-output decomposition.

We propose several new algorithms that deal with central issues of functional multiple-output decomposition. First, an efficient algorithm to solve the variable partitioning problem is described. Second, we show how to implicitly compute all preferable functions of a single-output function and how to identify all common preferable functions of a multiple-output function. Due to implicit computation in the crucial steps, the algorithm is very efficient. Experimental results show significant reductions in area.

Categories and Subject Descriptors: J.6 [**Computer Applications**]: Computer-Aided Engineering

General Terms: Design, Experimentation, Performance, Theory

Additional Key Words and Phrases: Assignable functions, Boolean functions, computer-aided design of VLSI, decomposition, FPGA technology , implicit BDD-based methods, mapping synthesis, multiple-output decomposition, preferable functions, subfunction sharing gain, subfunction sharing potential, variable partitioning for decomposition, TOS

Authors' addresses: B. Wurth and U. Schlichtmann, Infineon Technologies AG, Balanstrasse 73, Munich, 81541, Germany; K. Eckl and K. J. Antreich, Technical University of Munich, Munich, 80290, Germany.

1. INTRODUCTION

Functional decomposition is a logic synthesis technique that has recently attracted much attention, due primarily to field programmable gate-arrays (FPGAs) as a new technology for system implementation and rapid prototyping. A popular class of FPGAs is based on the lookup table (LUT) as the basic logic block. A K-LUT is capable of implementing any Boolean function of up to K variables. Thus, if a logic network consists of nodes with K or fewer inputs only (*K-bounded networks*), each node of the network can be realized by a K-LUT. Logic synthesis for LUT-based FPGAs has the task of transforming a given logic network into a functionally equivalent K-bounded network. As in the case of logic synthesis for conventional libraries, this task is divided into a logic optimization and a technology mapping step. The deficiencies of conventional library-oriented mapping methods for solving the latter problem have spawned the development of a variety of synthesis techniques [Cong and Ding 1996]. In particular, it renewed interest in functional decomposition techniques.

Functional decomposition subsumes Roth-Karp decomposition, which represents functions in sum-of-products form, as well as Ashenhurst decomposition, which is based on truth tables. Independently of the way the Boolean function is represented, functional decomposition is defined by using all the rules of Boolean algebra to manipulate Boolean functions. [1] Thus, functional decomposition is a Boolean method. To make the functional decomposition problem more tractable, most research has focused on the *disjoint decomposition* of single-output functions. This article also concentrates on disjoint decomposition.

The (disjoint) functional decomposition of a Boolean function vector $\mathbf{f}(\mathbf{x}, \mathbf{y})$ can be written as

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) = \mathbf{g}(\mathbf{d}(\mathbf{x}), \mathbf{y}), \tag{1}$$

where $\mathbf{d}$ is called the *subfunction vector* and $\mathbf{g}$ the *composition function vector*. The variables $\mathbf{x}$, on which only the subfunctions depend, are called *bound variables*, the $\mathbf{y}$ variables are the *free variables*. We identify two major steps in a decomposition algorithm: first, partitioning the input variables into the bound set and the free set, and second, computing subfunction vector $\mathbf{d}(\mathbf{x})$. Computing the composition function, on the other hand, is straightforward once the two major steps are carried out.

Functional decomposition is well suited to technology mapping for LUT-based FPGAs, for two reasons. First, it directly minimizes the figure of merit for LUT architectures, i.e., the number of inputs of the new functions. Second, functional decomposition is a powerful Boolean method. It is noteworthy that until recently the latter aspect posed the fundamental problem for practical application of functional decomposition: for any but

[1]Note that *algebraic* decomposition techniques neglect some properties of Boolean algebra, e.g., complements are not defined.

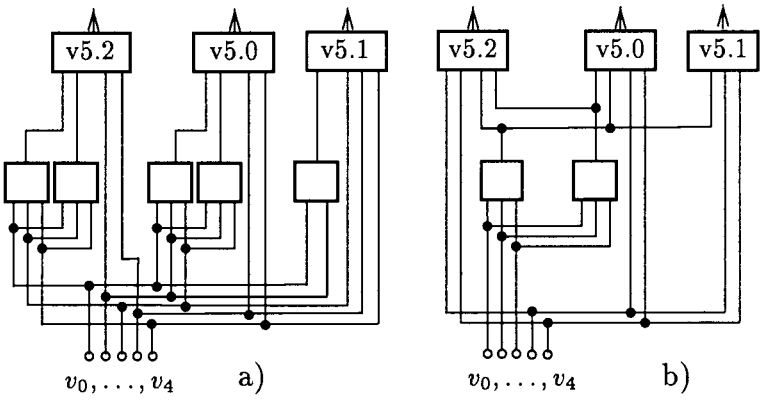

Fig. 1. (a) Single-output and (b) multiple-output decomposition of circuit rd53, $K = 4$.

very small problems, the solution space that must be searched by a Boolean method is huge. The availability of efficient data structures for Boolean functions (binary decision diagrams) was a prerequisite to deal with this problem.

Functional decomposition of multiple-output functions has the additional advantage of being capable of extracting shared logic. This is shown for a small circuit in Figure 1. Extracting common subfunctions has traditionally been performed during logic optimization using algebraic techniques. Multiple-output decomposition thus combines the previously separated steps of common subfunction extraction and technology mapping.

This article presents new theoretical results and an efficient algorithm for functional multiple-output decomposition, which enables the application of functional multiple-output decomposition to problems of practical size, thus improving the quality of logic synthesis.

After a summary of the state-of-the-art in Section 2, a new perspective on disjoint functional decomposition is first introduced for single-output functions in Section 4. This perspective helps to solve the multiple-output decomposition problem later on. The concept of *assignable* functions, resulting from an iterative computation of subfunctions, is also introduced in Section 4. The main contributions of multiple-output decomposition appear in Section 5. The efficiency of the new decomposition algorithm also relies on several implicit and BDD techniques, to which Section 6 is devoted. A novel solution to the variable partitioning problem is suggested in Section 7. Experimental results are presented and compared to a variety of state-of-the-art mapping methods in Section 8.

2. FUNCTIONAL DECOMPOSITION: STATE-OF-THE-ART

We review strategies for functional single-output and multiple-output decomposition, as well as to the variable partitioning problem.

The study of simple functional decompositions was pioneered by Ashenhurst [1953; 1959]. He introduces the *decomposition chart* (an example is shown in Figure 2). Given a variable partition, Ashenhurst formulates conditions for the existence of simple disjoint and nondisjoint decompositions of completely specified functions. Disjoint functional decomposition of completely specified functions using *more* than one subfunction was first studied by Curtis. He formulates a theorem that relates the minimum number of subfunctions to the number of distinct column vectors in the decomposition chart.

The size of decomposition charts is exponential in the number of variables. A more efficient method to test for functional decompositions is given by Roth and Karp [1962], who used sum of product representations. They formulate the decomposition condition stated in Theorem 4.1. Functional decomposition based on sum of product forms, as formulated by Roth and Karp, was state-of-the-art until recently [Murgai et al. 1990; 1991a; Sentovich et al. 1992]. Hwang et al. [1990; 1992; 1994] propose a special disjoint decomposition method, which directly synthesizes subfunctions and composition functions using AND and EXOR gates.

Roth and Karp [1962] as well as Curtis [1962] are concerned with computing a minimum number c of subfunctions d_i, but they do not pay attention to which subfunctions are selected. As expressed in Theorem 4.1, many different choices of subfunctions d_i are possible. Karp [1963] describes techniques that exploit the degrees of freedom and select simple subfunctions, i.e., subfunctions with special properties— functions that are independent of some bound variables, symmetric, unate, as well as threshold, functions are considered. A recent algorithm by Murgai exploits the mentioned degrees of freedom in selecting subfunctions with the goal of obtaining simple composition functions [Murgai et al. 1994].

Use of binary decision diagrams (BDDs) has drastically improved the efficiency of functional decomposition algorithms. The reason is that there is a one-to-one correspondence between classes of compatible bound set vertices and certain nodes of the BDD of f after constructing it with a proper variable order (see Section 7). The advantage of using binary decision diagrams for functional decomposition was discovered independently by several researchers [Sasao 1993; Lai et al. 1993; Schlichtmann 1993]. Lai et al. [1993], who study disjoint and nondisjoint decompositions of completely and incompletely specified functions, give the most general treatment of the problem.

We use the terminology of Section 5 in the ensuing discussion of multiple-output decomposition techniques. A first approach to functional multiple-output decomposition is proposed by Karp [1963]. Since it is based on the decomposition chart, it cannot be used for large functions. Moreover, it is applicable to functions with two outputs only. However, it has the advantage that several codes may be assigned to one class of compatible bound set vertices, as allowed by Theorem 4.1. Such an encoding is called "nonstrict" by Karp [1963] and "multicoding" by Lai et al. [1994a]. If just

one code is assigned to each compatible class (called "strict" decomposition by Karp and "unicoding" by Lai), not all common subfunctions can (usually) be detected.

Lai et al. propose a method called *shared subfunction encoding*, which carries out strict decompositions [Lai et al. 1994a; 1994b]. For each output f_k of the multiple-output function, the set of assignable functions constructable with respect to the *sum* of the local compatibility partitions is computed and represented by a bit vector. A common subfunction is identified by an entry at the same position in the bit vectors of several outputs. Since the length of the bit vectors is exponential in the number of bound variables, the method can only be used for small bound set sizes. This works well in Lai's decomposition approach, since he limits the bound set size to the number of LUT inputs, which is typically small. Lai et al. also describe a confined nonstrict decomposition method called *column encoding* [Lai et al. 1994a; 1994b], which enforces that all composition functions have inputs from all subfunctions. The subfunctions are obtained by iteratively assigning unique codes to global classes. A drawback of this approach is that the outputs of **f** are decomposed with a nonminimum number of subfunctions.

Molitor and Scholl [1994] suggest an approach that performs nonstrict decompositions . Multiple-output decomposition is carried out by solving a sequence of NP-complete decision problems. Such a decision problem questions the existence of a decomposition of a function vector **f** with a given number of common subfunctions. Thus, the maximum number of common subfunctions can be found. To solve a decision problem, a tree search is performed. After computing the maximum number of subfunctions common to all outputs of **f**, subsets of the set of outputs are recursively tested for common subfunctions. The drawback of this approach is the size of the search tree, which is exponential in the number of common subfunctions and in the number of global classes. A more recent algorithm by Scholl and Molitor circumvents this problem by performing strict decompositions only [Scholl and Molitor 1994]. This drastically reduces the depth of the search tree. Then, the decomposition is similar to Lai et al.'s [1994a] *shared subfunction encoding* algorithm.

The approaches to the variable partitioning problem for disjoint functional decomposition can be classified into enumerative, constructive, and iterative improvement methods. The bound set cardinality is denoted by b, the number of all variables by n.

Curtis [1961] suggest enumerating the $2^n - 2$ nontrivial variable partitions and estimating the circuit cost after decomposition for each variable partition using cost bounds for the composition function and the subfunctions. The cost bound of a function only depends on the number of its variables. Therefore, the cost after decomposition can be estimated using the bound set size and the number of subfunctions. This enumerative approach determines a good bound set size as well as the assignment of

variables to the bound set and free set. Of course, this approach becomes prohibitively expensive for large n.

More recent enumerative approaches assume a fixed bound set size. This is motivated by the application of functional decomposition for technology mapping to K-LUT architectures. If decompositions with bound set size K are carried out, each subfunction can directly be implemented by a K-LUT. The functional decomposition method implemented in the logic synthesis system SIS either selects the first variable partition with bound set size K that yields a nontrivial decomposition [Murgai et al. 1990], or enumerates all partitions of size K [Murgai et al. 1991a]. This enumerative approach is adapted for BDD-based functional decomposition by Lai et al. [1993] and similarly by Sasao [1993]. Since enumeration is very expensive, it is not applicable for functions with many variables. Therefore, it is usual to specify an upper bound on the number of variable partitions that are evaluated.

Recently, a heuristic was proposed that directly constructs a bound set of fixed size from the sum of products representation of f [Shen et al. 1995].

Two iterative improvement methods for fixed bound set size b were briefly mentioned by Hwang [1992; 1994]. In both methods, an iteration exchanges the bound and the free variable, resulting in the largest cost reduction. Iterations are performed according to the Kernighan-Lin algorithm or in a greedy manner. Since an iteration involves up to $b \cdot (n - b)$ variable exchanges, this method is costly for functions that depend on many variables. Again, the bound set size must be specified by the user.

3. PRELIMINARIES

We deal with Boolean functions and with partitions of the set $\mathcal{X}$ of bound set vertices $\mathbf{x}$.

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, denoted by a boldface letter, $(\mathbf{f} = f_1, \ldots, f_m)$. Vectors of *Boolean variables* x_i are also printed in boldface, $\mathbf{x} = (x_1, \ldots, x_n)$.

A *partition* Π of set $\mathcal{X}$ 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 $\mathcal{X}$. The partition of $\mathcal{X}$ induced by R is denoted $\mathcal{X}/R$.

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* of Π_1 and Π_2, denoted $\Pi_{prod} = \Pi_1 \cdot \Pi_2$, is the partition of $\mathcal{X}$ that has the smallest number of blocks and refines both Π_1 and Π_2. The product Π_{prod} of c partitions Π_i and is denoted

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

The power set of a set $\mathbf{S}$ is denoted by $\mathcal{P}(\mathbf{S})$.

4. SINGLE-OUTPUT DECOMPOSITION

Section 4.1 reformulates the decomposition condition using partitions of the set of bound set vertices. This formulation is employed in Section 4.2 to iteratively compute valid subfunctions. Section 9 illustrates the algorithm with a small example.

4.1 Theory

Let a completely specified function f and a disjoint partition of its n input variables into the bound set $\{x_1, \ldots, x_b\}$ and the free set $(y_1, \ldots, y_{n-b})$ be given. Functional decomposition determines a set of subfunctions $\mathcal{A}_f = \{d_1, \ldots, d_c\}$, called *assignment* [Karp 1963] and a composition function g such that

$$f(x_1, \ldots, x_b, y_1, \ldots, y_{n-b}) =$$

$$g(d_1(x_1, \ldots, x_b), \ldots, d_c(x_1, \ldots, x_b), y_1, \ldots y_{n-b}). \tag{2}$$

Let $\mathcal{X} = \{0,1\}^b$ denote the set of bound set vertices. The existence of a decomposition according to Eq. (2) can be tested using a relation of compatibility between elements of $\mathcal{X}$. For completely specified functions, which we assume in the sequel, compatibility is an equivalence relation.

Definition 4.1 Two bound set vertices $\hat{\mathbf{x}}_\alpha \in \mathcal{X}$ and $\hat{\mathbf{x}}_\beta \in \mathcal{X}$ are *compatible*, denoted $\hat{\mathbf{x}}_\alpha R_f \hat{\mathbf{x}}_\beta$, iff

$$\forall \hat{\mathbf{y}} \in \{0,1\}^{n-b} : f(\hat{\mathbf{x}}_\alpha, \hat{\mathbf{y}}) = f(\hat{\mathbf{x}}_\beta, \hat{\mathbf{y}}). \tag{3}$$

The partition $\Pi_f := \mathcal{X}/R_f = \{L_1, \ldots, L_\ell\}$ induced by the compatibility relation R_f is called the *compatibility partition*. A class L_i of Π_f is called a *compatible class*, and the number of compatible classes or *rank* of Π_f is denoted by ℓ.[2]

The *decomposition chart* visualizes compatibility of bound set vertices. The decomposition chart is a Karnaugh map where rows are associated with free set vertices and columns are associated with bound set vertices. Two bound set vertices $\hat{\mathbf{x}}_\alpha$ and $\hat{\mathbf{x}}_\beta$ are compatible if their column patterns are identical. The column pattern multiplicity, i.e., the number of distinct column patterns, is equal to the rank of Π_f. The decomposition chart of function f_1 (shown in Figure 2(a)) has three distinct column patterns. The compatibility partition $\Pi_{f_1} = \mathcal{X}/R_{f_1} = \{L_1, L_2, L_3\}$ is depicted in Figure 3(a).

Roth and Karp [1962] formulated the following theorem:

[2]Compatibility partition and compatible classes, which are associated with a single-output function, are later additionally denoted by the term *local*.

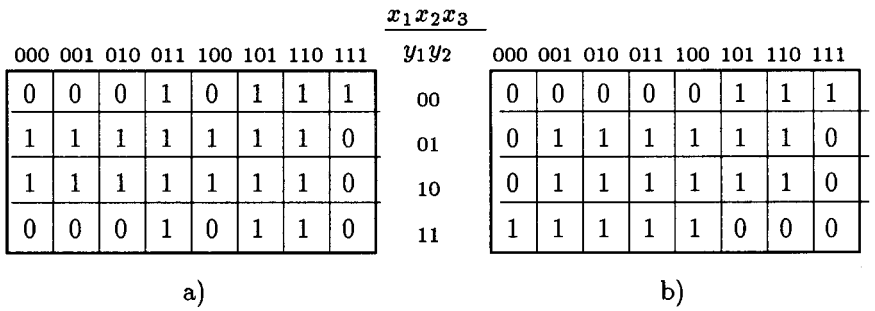

Fig. 2. Decomposition chart of (a) function f_1 and (b) function f_2.

THEOREM 4.1 *Given a function* $f(\mathbf{x}, \mathbf{y})$ *and a subfunction vector* $\mathbf{d}(\mathbf{x})$, *there exists a composition function* $g(\mathbf{v}, \mathbf{y})$ *such that* $f(\mathbf{x}, \mathbf{y}) = g(\mathbf{d}(\mathbf{x}), \mathbf{y})$ *if and only if*

$$\forall \hat{\mathbf{x}}_\alpha, \hat{\mathbf{x}}_\beta \in \mathcal{X} : \neg\, (\hat{\mathbf{x}}_\alpha R_f \hat{\mathbf{x}}_\beta) \Rightarrow \mathbf{d}(\hat{\mathbf{x}}_\alpha) \neq \mathbf{d}(\hat{\mathbf{x}}_\beta). \tag{4}$$

Formula (4) expresses the *decomposition condition*. The *code* of the bound set vertex $\hat{\mathbf{x}}_\alpha$ is the value $\mathbf{d}(\hat{\mathbf{x}}_\alpha)$ of the subfunction vector $\mathbf{d}$ for $\hat{\mathbf{x}}_\alpha$. The subfunction vector $\mathbf{d}$ may evaluate to identical or to different codes for compatible bound set vertices, but according to Formula (1), it must evaluate to different codes for incompatible bound set vertices. The smallest number $\mathbf{d}$ of subfunctions, called *codewidth*, for Theorem 4.1 to be satisfied is such that $2^{c-1} < \ell \leq 2^c$, i.e.,

$$c = \lceil \mathrm{ld}\, \ell \rceil. \tag{5}$$

Choosing this minimum value minimizes the number of subfunctions and the number of inputs of the composition function. We always assume this minimum value. Note that the number ℓ of compatible classes can easily be derived from a BDD representing f, as described by Lai et al [1993].

While the decomposition condition is usually stated as in Theorem 4.1, we now derive a more abstract formulation that is based on partitions. The subfunctions define a partition of the set of bound set vertices into classes with identical code, which is illustrated by dashed and dotted lines in Figure 3(b).

Definition 4.2 The *code partition* $\Pi_{\mathcal{A}_f}$ is given by

$$\Pi_{A_f} = \prod_{d_i \in \mathcal{A}_f} \Pi_{d_i} \tag{6}$$

where $\Pi_{d_i} = \mathcal{X}/R_{d_i} = \{on - set(d_i), off - set(d_i)\}$.

The relation $R_{d_i} = \{(\hat{\mathbf{x}}_\alpha, \hat{\mathbf{x}}_\beta) \mid d_i(\hat{\mathbf{x}}_\alpha) = d_i(\hat{\mathbf{x}}_\beta)\}$ relates two bound set vertices if they both belong to either the onset or the offset of d_i. We use the code partition and the compatibility partition to restate Theorem 1:

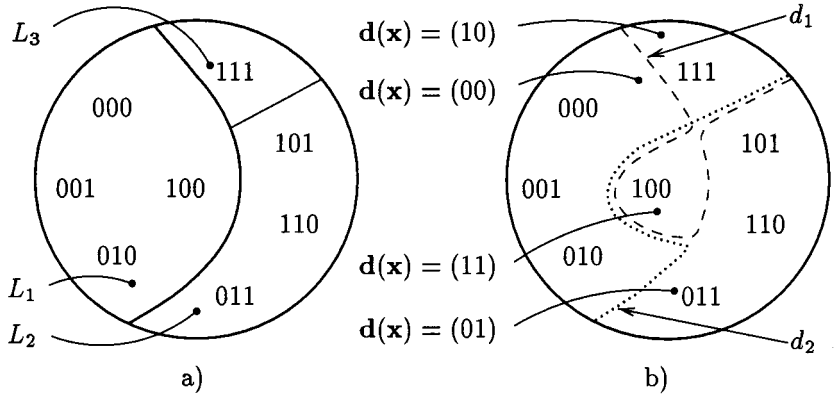

Fig. 3. Partition of bound set vertices: (a) by the compatibility partition Π_{f_1} of function f_1 (solid lines) and (b) by the code partition $\Pi_{\mathcal{A}_f}$ (dashed and dotted lines).

THEOREM 4.2 *A decomposition of the single-output function f by a sub-function vector* $\mathbf{d}$, $\mathcal{A}_f = set(\mathbf{d})$ *exists if and only if the code partition* $\Pi_{\mathcal{A}_f}$ *refines the compatibility partition* Π_f.

PROOF. From Theorem 4.1, we have

$$\forall \hat{\mathbf{x}}_\alpha, \hat{\mathbf{x}}_\beta \in \mathcal{X} : \neg (\hat{\mathbf{x}}_\alpha R_f \hat{\mathbf{x}}_\beta) \Rightarrow \mathbf{d}(\hat{\mathbf{x}}_\alpha) \neq \mathbf{d}(\hat{\mathbf{x}}_\beta) \tag{7}$$

$$\Leftrightarrow \forall \hat{\mathbf{x}}_\alpha, \hat{\mathbf{x}}_\beta \in \mathcal{X} : \mathbf{d}(\hat{\mathbf{x}}_\alpha) = \mathbf{d}(\hat{\mathbf{x}}_\beta) \Rightarrow \hat{\mathbf{x}}_\alpha R_f \hat{\mathbf{x}}_\beta \tag{8}$$

$$\Leftrightarrow \bigcap_{i=1}^{c} R_{d_i} \subseteq R_f \tag{9}$$

$$\Leftrightarrow \Pi_{\mathcal{A}_f} \textit{ refines } \Pi_f. \tag{10}$$

□

From Theorem 4.2 it is obvious that the actual codes assigned to compatible classes are not important, but only the property of the codes to differentiate between different compatible classes is important. Using Theorem 4.2 instead of Theorem 4.1 proves particularly useful in the context of multiple-output decomposition.

Definition 4.3 A decomposition is called *strict* if $\Pi_{\mathcal{A}_f} = \Pi_f$.

Apparently, $\Pi_{\mathcal{A}_f} = \Pi_f \Leftrightarrow \Pi_{\mathcal{A}_f}$ *refines* $\Pi_f \wedge \Pi_f$ *refines* $\Pi_{\mathcal{A}_f}$. Whereas $\Pi_{\mathcal{A}_f}$ *refines* Π_f is the decomposition condition, and Π_f *refines* $\Pi_{\mathcal{A}_f}$ can be transformed into

$$\forall \hat{\mathbf{x}}_\alpha, \hat{\mathbf{x}}_\beta \in \mathcal{X} : \hat{\mathbf{x}}_\alpha R_f \hat{\mathbf{x}}_\beta \Rightarrow d(\hat{\mathbf{x}}_\alpha) = d(\hat{\mathbf{x}}_\beta), \tag{11}$$

by a computation similar to the above proof. As expressed by Formula (11), a strict decomposition encodes compatible vertices by the same code.

The decomposition of function f_1 by the subfunction vector **d** shown in Figure 3 is nonstrict, since the compatible class L_1 contains bound set vertices with the codes (00) and (11).

The example illustrates that the problem of computing subfunctions is equivalent to the problem of encoding the set of bound set vertices. Having selected codes for each bound set vertex, the subfunctions producing these codes are fixed.

Strict decomposition is popular [Murgai et al. 1990; Lai et al. 1993; Schlichtmann 1993] because selecting a unique code for each compatible class is a straightforward method that guarantees a minimal number of subfunctions. It is shown later on, however, that a strict decomposition does not generate a minimal number of subfunctions if a multiple-output function is decomposed.

4.2 An Iterative Decomposition Algorithm

Strict decompositions yield a restricted set of assignments only. We are interested in a general understanding of the decomposition problem, which incorporates nonstrict as well as strict decompositions. We therefore suggest an approach in which subfunctions are selected iteratively. Thus, the codes of compatible classes are determined bit by bit. First, some terminology is introduced.

Definition 4.4 A set of functions $\{d_1, \ldots, d_s\}$, $s \leq c$, is a *partial assignment* $\mathcal{A}_{f,s}$ if there exists a supplementary set of functions $\{d_{s+1}, \ldots, d_c\}$ such that the union of both sets is a complete assignment. The product $\Pi_{\mathcal{A}_{f,s}}$ of the partitions Π_{d_i}, $i = 1, \ldots, s$, associated with the partial assignment,

$$\Pi_{\mathcal{A}_{f,s}} = \{X_1, \ldots, X_\theta\} := \prod_{i=1}^{s} \Pi_{d_i},$$

is called the *partial code partition*. A block of the partial code partition is denoted by X_j, the number of blocks of the partial code partition by θ.

For better differentiation between the compatibility partition and the code partition, we say that the compatibility partition is made up of *classes*, whereas the code partition consists of *blocks*. Let us now briefly consider some special cases: For $s = 0$, we have $\Pi_{\mathcal{A}_{f,0}} = \emptyset$ and $\Pi_{\mathcal{A}_{f,0}} = \{\mathcal{X}\}$. For $s = c$, we have $\mathcal{A}_{f,c} = \mathcal{A}_f$ and $\Pi_{\mathcal{A}_{f,c}} = \Pi_{\mathcal{A}_f}$.

LEMMA 4.1 *A set of subfunctions* $\{d_1, \ldots, d_s\}$ *is a* ***partial assignment*** $\mathcal{A}_{f,s}$ *iff the partial code partition* $\Pi_{\mathcal{A}_{f,s}}$ *has no block* X_j *that contains bound set vertices of more than* 2^{c-s} *compatible classes.*

PROOF. $\Rightarrow$ by contradiction: Let us assume a partial assignment $\Pi_{\mathcal{A}_{f,s}}$, which has a block that intersects more than 2^{c-s} compatible classes. Then the remaining $(c - s)$ subfunctions cannot be chosen such that the vertices

of these classes finally have different codes because not more than 2^{c-s} codes are available. Thus, the set of functions $\{d_1, \ldots, d_s\}$ cannot be a partial assignment, which contradicts the assumption.

$\Leftarrow$: If no block X_j contains bound set vertices of more than 2^{c-s} compatible classes, we can use the remaining $(c - s)$ subfunctions to assign different codes to the vertices of compatible classes in a straightforward way. We can thus construct the supplementary set of functions $\{d_{s+1}, \ldots, d_c\}$, which shows that the set $\{d_1, \ldots, d_s\}$ is a partial assignment. □

For $s = c$, the above lemma corresponds to Theorem 4.2. Now we can define the assignable property:

Definition 4.5 A function $d : \{0,1\}^n \rightarrow \{0,1\}$ is *assignable* with respect to a given partial assignment $\Pi_{\mathcal{A}_{f,s}}$ iff $\Pi_{\mathcal{A}_{f,s}} \cup d$ is a partial assignment $\mathcal{A}_{f,s+1}$.

It must be stressed that assignability is always related to an already selected set of assignable subfunctions, i.e., a partial assignment $\mathcal{A}_{f,s}$. Therefore, a function that is assignable at an early stage of the iterative procedure is not necessarily assignable at a later stage.

We now consider the first step of the iterative decomposition procedure, and show how to compute assignable functions.

LEMMA 4.2 *A function d is* ***assignable*** *in the first iteration (i.e., assignable with respect to* $\mathcal{A}_{f,0}$*) iff its* $on - set(d)$ *as well as its* $off - set(d)$ *completely contain at least* $\ell - 2^{c-1}$ *compatible classes.*

PROOF. Let $on - set(d)$ completely contain at least $\ell - 2^{c-1} - \alpha$ compatible classes, $\alpha \geq 0$, and let $off - set(d)$ completely contain at least $\ell - 2^{c-1} - \beta$ compatible classes, $\beta \geq 0$, for fixed values of α and β. Then, the number of compatible classes that are divided among $on - set(d)$ and $off - set(d)$ is

$$\ell - (\ell - 2^{c-1} - \alpha) - (\ell - 2^{c-1} - \beta) = 2^c - \ell + \alpha + \beta. \tag{12}$$

Thus, $on - set(d)$ contains vertices of up to

$$(\ell - 2^{c-1} - \alpha) + (2^c - \ell + \alpha + \beta) = 2^{c-1} + \beta \tag{13}$$

compatible classes, and $off - set(d)$ contains vertices of up to

$$(\ell - 2^{c-1} - \beta) + (2^c - \ell + \alpha + \beta) = 2^{c-1} + \alpha \tag{14}$$

compatible classes. According to Lemma 4.1 and Definition 4.5, function d is assignable in the first iteration iff the partial code partition $\Pi_{\mathcal{A}_{f,1}} = \Pi_d = \{on - set(d), off - set(d)\}$ has no block that contains vertices of more

than 2^{c-1} compatible classes. Using the above equations, we know that neither $on - set(d)$ nor $off - set(d)$ contain vertices of more than 2^{c-1} compatible classes iff $\alpha = \beta = 0$. □

The following theorem, which is a generalization of Lemma 4.2, can directly be applied in a decomposition algorithm to compute all assignable functions in each iteration. It gives a condition for each block X_j of a partial code partition $\Pi_{\mathcal{A}_{f,s}}$, and makes use of the set $X_j / R_f = \{L_1^j, \ldots, L_k^j\}$, which is a collection of compatible classes that are partially or completely contained in block X_j. The cardinality of this set is denoted by k.

THEOREM 4.3 *A function d is **assignable** with respect to a partial assignment $\mathcal{A}_{f,s}$ iff for each block X_j, $X_j / R_f = \{L_1^j, \ldots, L_k^j\}$, of the corresponding partial code partition $\Pi_{\mathcal{A}_{f,s}}$ the following holds: At least $k - 2^{c-s-1}$ classes L_i^j are completely contained in the offset of d (condition C0) and the onset of d (condition C1).*

If condition C0 is not fulfilled, then the onset of d contains bound set vertices of more than $2^{c-(s+1)}$ classes L_i, which violates Lemma 4.1 for the partial assignment $\mathcal{A}_{f,s+1}$. The analogy holds for condition C1. The complete algorithm for single output decomposition is given in Figure 4.

The example in the appendix shows that in each iteration there is a large number of assignable functions from which to select. Later on, experimental results demonstrate that the number of assignable functions can be huge when decomposing functions of moderate size. The next section shows how these degrees of freedom can be exploited for multiple-output decomposition.

5. MULTIPLE-OUTPUT DECOMPOSITION

5.1 Theory

Let a completely specified function vector $\mathbf{f} = (f_1, \ldots, f_m)$ and a disjoint partition of its n input variables into the bound set $\{x_1, \ldots, x_b\}$ and the free set $\{y_1, \ldots, y_{n-b}\}$ be given. Functional multiple-output decomposition determines an assignment $\mathcal{A}_{\mathbf{f}} = \{d_1, \ldots, d_q\}$ and the composition functions $g_1, \ldots, g_m$ such that for each k, $1 \le k \le m$,

$$f_k(x_1, \ldots, x_b, y_1, \ldots, y_{n-b}) =$$

$$g_k(d_1^k(x_1, \ldots, x_b), \ldots, d_{c_k}^k(x_1, \ldots, x_b), y_1, \ldots, y_{n-b}), \text{ where } \forall_i d_i^k \in \mathcal{A}_{\mathbf{f}}. \tag{15}$$

Multiple-output decomposition has the goal of finding subfunctions that can be used for several outputs, thus the total number of subfunctions, denoted q, shall be smaller than the sum over the numbers of subfunctions needed to decompose each individual output $\Sigma_{k=1}^{m} c_k$.

```
algorithm single_output_decomp(f, x, y)
{
    compute compatibility partition Π_f = {L_1, ..., L_ℓ};
    compute codewidth c = ⌈ld ℓ⌉;
    A_{f,0} = ∅;
    Π_{A_{f,0}} = {X};

    for s = 1 to c {
        compute set D_s of functions assignable w.r.t. A_{f,s-1};
        select d_s ∈ D_s;
        A_{f,s} = A_{f,s-1} ∪ d_s;
        Π_{A_{f,s}} = Π_{A_{f,s-1}} · Π_{d_s};
    }
    A_f = A_{f,c};
    compute composition function g from assignment A_f;
}
```

Fig. 4. Iterative algorithm for single-output decomposition.

Note that symbols with a subscript or superscript k refer to the output f_k of the function vector $\mathbf{f}$. Terminology referring to an individual output are additionally denoted *local* in the sequel.

In Theorem 4.2, we stated a condition for the existence of a single-output decomposition, saying that the functional decomposition of f requires the refinement of the local compatibility partition Π_f by the local code partition $\Pi_{\mathcal{A}_f}$. We now give a similar condition for multiple-output decomposition:

THEOREM 5.1 *A decomposition of the multiple-output function* $\mathbf{f} = (f_1, \ldots, f_m)$ *by a set of subfunctions* $\mathcal{A}_\mathbf{f} = \{d_1, \ldots, d_q\}$ *exists iff for each output* f_k *there is a subset* $\mathcal{A}_{f_k} \subseteq \mathcal{A}_\mathbf{f}$ *with cardinality equal to the codewidth* c_k *such that*

$$\Pi_{\mathcal{A}_{f_k}} \text{ refines } \Pi_{f_k}. \tag{16}$$

PROOF. Function vector $\mathbf{f}$ can be decomposed iff the condition for single-output decomposition is fulfilled for each individual output f_k. This condition, as expressed in Formula (10), is simply repeated in Formula (16) for each output. □

The multiple-output decomposition problem can be stated as follows: *Given a multiple-output function* $\mathbf{f}(\mathbf{x}, \mathbf{y})$*, determine a minimal set* $\mathcal{A}_\mathbf{f}$ *of subfunctions such that the decomposition condition expressed in Theorem 5.1 is fulfilled.* A first, obvious step towards this goal is to use the minimum number of subfunctions, $c_k = \lceil \mathrm{ld} \ell_k \rceil$, for each output f_k. As for single-output decomposition, we always choose this minimum number. This guarantees that multiple-output decomposition never yields more subfunctions than single-output decomposition under the same variable partition.

The central task of multiple-output decomposition, however, is to compute subfunctions that can be used for several outputs. One approach to this problem is to first compute the set of assignable functions for each output and then to find functions that are assignable for a maximum number of outputs. However, such an approach is only practical for small bound sets, due to the very large number of assignable functions. We show that it suffices to consider a fraction of the assignable functions without loss of result quality.

Formula (16) can also be written as

$$\bigcap_{d_i \in \mathcal{A}_{f_k}} R_{d_i} \subseteq R_{f_k}. \tag{17}$$

Using the laws of set algebra, we obtain

$$\bigcap_{i=1}^{q} R_{d_i} \subseteq \bigcap_{k=1}^{m} R_{f_k} \tag{18}$$

$$\Leftrightarrow \prod_{i=1}^{q} \Pi_{d_i} \textit{ refines } \prod_{k=1}^{m} \Pi_{f_k} \tag{19}$$

$$\Leftrightarrow \Pi_{\mathcal{A}_\mathbf{f}} \textit{ refines } \Pi_\mathbf{f}. \tag{20}$$

Here, we use the following definitions:

Definition 5.1 The product of the local compatibility partitions Π_{f_k} is called the *global (compatibility) partition* $\Pi_\mathbf{f}$, which partitions $\mathcal{X}$ into p *global classes* G_i:

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

The product of the bipartitions of $\mathcal{X}$ induced by the subfunctions d_i is called the *global code partition*:

$$\Pi_{\mathcal{A}_\mathbf{f}} := \prod_{i=1}^{q} \Pi_{d_i}.$$

The interpretation of the global compatibility partition is simple: $\Pi_{\mathbf{f}}$ partitions the bound set vertices into classes of vertices that are compatible for each of the m function outputs. It thus reflects the multiple-output function and the chosen variable partition. The global code partition, on the other hand, results from the chosen subfunction vector $\mathbf{d}$. It partitions the set of bound set vertices into blocks having identical code. The following example illustrates these concepts.

The top part of Figure 5 shows the local compatibility partitions Π_{f_1} and Π_{f_2} of the functions given in Figure 2 and the global partition $\Pi_{\mathbf{f}} = \Pi_{f_1} \cdot \Pi_{f_2}$. The bottom part shows the partitions induced by three subfunctions d_1, d_2, d_3 that satisfy the condition of Theorem 5.1. The vertices in the dark shaded areas constitute the offset of the subfunctions. The individual assignments are $\mathcal{A}_{f_1} = \{d_1, d_2\}$ and $\mathcal{A}_{f_2} = \{d_1, d_3\}$. Thus the multiple-output function $\mathbf{f} = (f_1, f_2)$ can be decomposed with three subfunctions such that $f_1 = g_1(d_1(\mathbf{x}), d_2(\mathbf{x}), \mathbf{y})$ and $f_2 = g_2(d_1(\mathbf{x}), d_3(\mathbf{x}), \mathbf{y})$. Subfunction d_1 is shared between both outputs. In the example, $\Pi_{\mathcal{A}_{\mathbf{f}}} = \Pi_{d_1} \cdot \Pi_{d_2} \cdot \Pi_{d_3} = \Pi_{\mathbf{f}}$. Note that Formula (20) also allows a proper refinement of $\Pi_{\mathbf{f}}$ by $\Pi_{\mathcal{A}_{\mathbf{f}}}$. The subfunctions d_1, d_2, and d_3 are constructed using global classes. We now show the relevance of such functions, which are defined as constructable:

Definition 5.2 Given a global partition $\Pi_{\mathbf{f}} = \{G_1, \ldots, G_p\}$, a function $d : \{0,1\}^b \rightarrow \{0,1\}$ is called *constructable* with respect to $\Pi_{\mathbf{f}}$ iff each class G_i is completely contained in either the onset or the offset of d,

$$on - set(d) \in \mathcal{P}(\Pi_{\mathbf{f}}). \tag{21}$$

An assignment that consists of constructable subfunctions only is called a *constructable assignment*.

If an assignment $\mathcal{A}_{\mathbf{f}}$ is constructable, then instead of Formula (20) the equation

$$\Pi_{\mathcal{A}_{\mathbf{f}}} = \Pi_{\mathbf{f}} \tag{22}$$

holds.

In Figure 5, the assignment $\{d_1, d_2, d_3\}$ is obviously constructable. Note that d_1 and d_2 in Figure 3 are not constructable.

The following theorem is central to our theory:

THEOREM 5.2 *Any nonconstructable assignment can be replaced by a constructable assignment without increasing the number q of required subfunctions.*

Every nonconstructable assignment that performs a decomposition with a minimum number of subfunctions is especially replaceable by a constructable assignment. *Therefore, an optimum decomposition with a minimum*

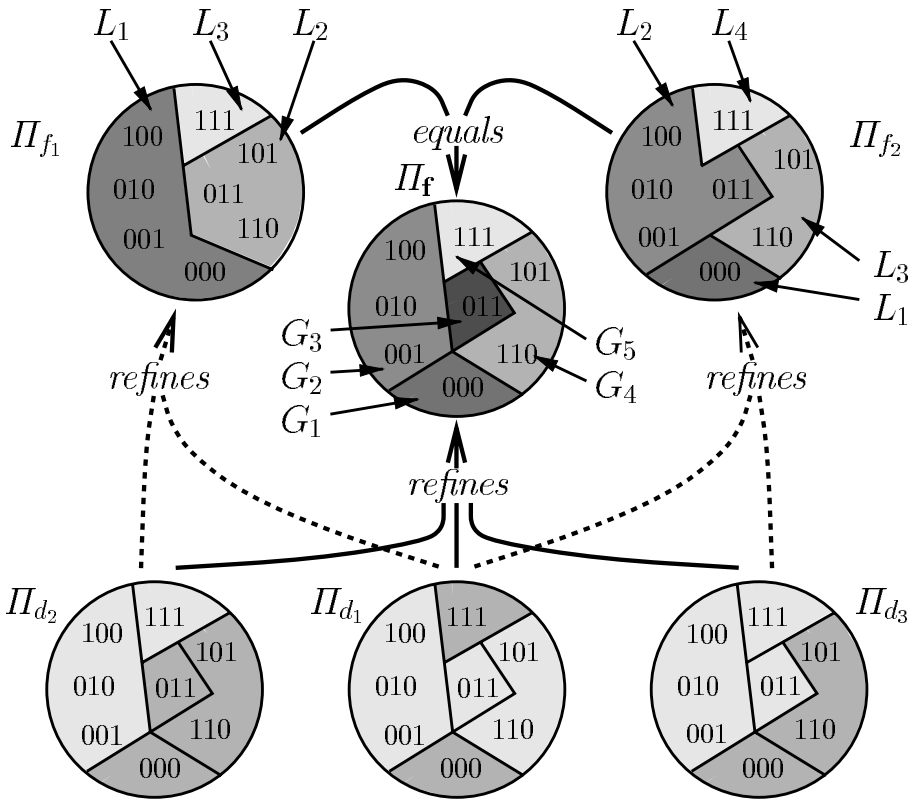

Fig. 5. The global compatibility partition $\Pi_{\mathbf{f}}$ derived from the local compatibility partitions Π_{f_k}; partitions Π_{d_i} of a minimum assignment $\mathcal{A}_{\mathbf{f}} = \{d_1, d_2, d_3\}$.

number of subfunctions can be obtained by choosing constructable subfunctions only.

PROOF. Let $\mathcal{A}_{\mathbf{f}}$ be a nonconstructable assignment. By Definition 5.2, there exists at least one class $G_i \in \Pi_{\mathbf{f}}$ that contains disjoint subsets $S_1 \subset G_i$ and $S_2 \subset G_i$ with different codes. Since the $\mathbf{x}$-vertices of S_1 and S_2 belong to the same global class, they are compatible for each output, and may as well have identical codes. The subfunctions can thus be changed such that the vertices of S_1 obtain the code associated with the vertices of S_2. Repeating this procedure for all subsets S_1 and S_2, a constructable assignment is obtained. □

So it suffices to consider only constructable functions as subfunctions, without detriment to the final number of subfunctions. Note, however, that assignments of equal size may have different costs after further decomposition.

To build a constructable function d, each global class is assigned to either the onset or the offset. Then, the number of constructable functions is 2^p. Due to the regularity of most applications, this is much smaller than the

number of all functions $d : \{0,1\}^b \rightarrow \{0,1\}$, which is 2^{2^b}. As an example, for benchmark circuit alu4, a multiple-output decomposition is performed with $p = 32$ and $b = 8$. The number of constructable functions is $2^{32} = 4.295 \cdot 10^9$, while $2^{2^8} = 1.158 \cdot 10^{77}$.

We have derived two properties of subfunctions, *assignability* and *constructability*. A subfunction must be assignable to satisfy the basic decomposition condition for a single-output. Constructability is sufficient for an optimal sharing of subfunctions during multiple-output decomposition. Subfunctions with both properties are called *preferable*:

Definition 5.3 A function $d : \{0,1\}^b \rightarrow \{0,1\}$ 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 partial assignment $\mathcal{A}_{f_k, s_k}$. Second, it must be constructable with respect to the global compatibility partition determined by $\mathbf{f}(\mathbf{x}, \mathbf{y})$.

We derive another useful property from (19):

Property 5.1 The number p of global classes determines a lower bound on the number q of subfunctions: $\lceil \mathrm{ld}\, p \rceil \leq q$. This property can be derived easily. The q subfunctions cannot create more than 2^q codes, i.e., $|\Pi_{d_1} \cdot \ldots \cdot \Pi_{d_q}| \leq 2^q$. With (19), we have

$$2^q \geq |\prod_{i=1}^{q} \Pi_{d_i}| \geq |\prod_{k=1}^{m} \Pi_{f_k}| = p \Leftrightarrow \tag{23}$$

$$q \geq \lceil \mathrm{ld}\, p \rceil. \tag{24}$$

In Figure 5, we have $p = 5 \Rightarrow q \geq 3$. The found solution with $q = 3$ is therefore an optimum solution with respect to the number of subfunctions. This property allows us to abort multiple-output decomposition at an early stage if the number of shared subfunctions is too small, which may be due to a bad variable partition.

We use this property to define the sharing potential of a multiple-output function.

Definition 5.4 The *subfunction sharing potential* φ is defined as

$$\varphi = 1 - \frac{\lceil \mathrm{ld}\, p \rceil}{\sum_{k=1}^{m} c_k},$$

where $\lceil \mathrm{ld}\, p \rceil$ is the lower bound on the number of subfunctions, and the denominator is the number of subfunctions needed for single-output decomposition.

The sharing potential gives an upper bound on the reduction of the number of subfunctions, which is possible by multiple-output decomposi-

tion compared with single-output decomposition. For the example of Figure 5, we have $\varphi = 1 - \lceil \mathrm{ld}5 \rceil / 2 + 2 = 25\%$, i.e., the number of subfunctions is reduced by 25%. This sharing potential is realized by the assignment $A_{\mathbf{f}} = \{d_1, d_2, d_3\}$. As the experimental results demonstrate, the sharing potential can not always be realized. The actual success in sharing subfunctions is expressed by the *sharing gain*.

Definition 5.5 The *subfunction sharing gain* ω is defined as

$$\omega = 1 - \frac{q}{\sum_{k=1}^{m} c_k},$$

and gives the actual reduction in the number of subfunctions realized by multiple-output decomposition, compared with single-output decomposition.

It is interesting to note that the *column-encoding* method presented by Lai et al. [1994a; 1994b] always realizes the complete sharing potential by using $\lceil \mathrm{ld}\, p \rceil$ subfunctions. However, the constraints expressed by Eq. (5) are not taken into account in that method. Thus, the composition functions may actually depend on more variables than necessary. This shows that the quantities *sharing potential* and *sharing gain* are only useful if the constraint on the number of subfunctions for each output is obeyed.

The *shared subfunction encoding* method of Lai et al. [1994a; 1994b], as well as the most recent approach of Scholl and Molitor [1994], are strict decomposition methods. These methods are based on the sum of the local compatibility partitions,

$$\bar{\Pi} = \sum_{k=1}^{m} \Pi_{f_k}.$$

These methods first search common subfunctions that are constructable in the classes of $\bar{\Pi}$. The sum partition $\bar{\Pi}$ typically has many fewer classes than the product partition $\Pi_{\mathbf{f}}$, e.g., the sum partition for the example in Figure 5 has only two classes. By only considering subfunctions that are constructable from classes of $\bar{\Pi}$, many common subfunctions may be overlooked. In the example in Figure 5, no common subfunction can be computed from the classes of $\bar{\Pi}$.

5.2 An Iterative Decomposition Algorithm

The complete algorithm for multiple-output decomposition is given in Figure 6. After computing the local compatibility partitions and initializing the local partial assignments, $\mathcal{A}_{f_k,0}$, and local partial code partitions, $\Pi_{\mathcal{A}_{f_k,0}}$, the global compatibility partition $\Pi_{\mathbf{f}}$ is computed. Then, the set of preferable functions is computed for each output f_k. In each iteration of the main

Table I. Characteristics of Multiple-Output Decomposition

			p				sharing		
f	b	ℓ_k	ub	actual	#assign.	#prefer.	φ	ω	CPU/sec
$\mathbf{f}_{f51m}$	5		32	5	$(4.3 \cdot 10^9)$	(32)	50%	50%	0.052
$m = 3$		2			2	2			
		4			6	6			
		5			$1.3 \cdot 10^7$	30			
$\mathbf{f}_{alu4}$	8		256	32	$(1.2 \cdot 10^{77})$	$(4.3 \cdot 10^9)$	66%	53%	3.413
$m = 3$		24			$2.1 \cdot 10^{48}$	$3.1 \cdot 10^{10}$			
		25			$8.8 \cdot 10^{44}$	$2.8 \cdot 10^9$			
		26			$1.4 \cdot 10^{44}$	$2.6 \cdot 10^9$			
$\mathbf{f}_{term1}$	7		128	64	$(3.4 \cdot 10^{38})$	$(1.8 \cdot 10^{19})$	82%	61%	60.880
$m = 6$		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			$3.4 \cdot 10^{37}$	$2.8 \cdot 10^{18}$			

loop, a subfunction d_i is selected from the union set of all preferable functions such that d_i is assignable for a maximum number of outputs f_k. For each of these outputs, the partial assignment and the partial code partition are updated, the set of preferable functions is recomputed, and the subfunction counter s_k is incremented. The main loop is repeated until the number s_k of subfunctions selected for each output equals $c_k = \lceil \mathrm{ld} \ell_k \rceil$, i.e., until complete assignments are obtained. Finally, each composition function g_k is computed using the assignment $\mathcal{A}_{f_k}$.

5.3 Preferable versus Assignable Functions

Experiments show the reduction in average complexity achieved by using preferable instead of assignable subfunctions. Characteristic data on various multiple-output decompositions is given in Table I, which reflects the decomposition of some function vectors that occurred during multiple-output decomposition of circuits `f51m`, `alu4` and `term1` [Yang 1991].

The name of the function vector and the number of outputs m are given in the first column. The bound set size b and the number of local classes ℓ_k of each output are shown in columns 2 and 3. The next two columns refer to the number of global classes p, where the upper bound $ub = min((\prod_{i=1}^{m} \ell_i), 2^b)$ and the actual number is given. The number of assignable functions # *assign.* and of preferable functions # *prefer* is shown. The values in parentheses give an upper bound on the number of functions, which is 2^{2^b} and 2^p, respectively. The sharing potential φ and the sharing gain ω are shown next. The CPU time needed to decompose the characterized function vector is also given. The CPU times in all experiments are measured on a DEC AlphaStation 250 4/266 with 128 MByte.

```
algorithm multiple-output_decomp(f, x, y)
{
    for k = 1 to m {
        compute local compatibility partition Π_{f_k} = {L_1, ..., L_{ℓ_k}};
        compute codewidth c_k = ⌈ld ℓ_k⌉;
        A_{f_k,0} = ∅;
        Π_{A_{f_k,0}} = {X};
        s_k = 1;
    }
    compute global compatibility partition Π_f = Π_{f_1} · .... · Π_{f_m};
    for k = 1 to m {
        compute set P_{k,1} of functions preferable w.r.t. A_{f_k,0};
    }
    q = 0;
    while (there is a k ∈ {1, ..., m} with s_k ≤ c_k) {
        q = q + 1;
        select d_q ∈ (P_{1,s_1} ∪ ... ∪ P_{m,s_m});
        for k = 1 to m {
            if (d_q ∈ P_{k,s_k}) {
                A_{f_k,s_k} = A_{f_k,s_k-1} ∪ d_q;
                Π_{A_{f_k,s_k}} = Π_{A_{f_k,s_k-1}} · Π_{d_q};
                if (s_k < c_k) {
                    compute set P_{k,s_k+1} of functions preferable w.r.t. A_{f_k,s_k};
                }
                if (s_k ≤ c_k) {
                    s_k = s_k + 1;
                }
            }
        }
    }
    for k = 1 to m {
        A_{f_k} = A_{f_k,c_k};
        compute composition function g_k from assignment A_{f_k};
    }
```

Fig. 6. Iterative algorithm for multiple-output decomposition.

It can be seen that the number of global classes, which significantly influences memory and CPU time consumption, is mostly much smaller than its theoretical upper bound. In our experience, decompositions with fewer than 25 global classes are usually performed in less than a second.

Since the number of global classes is typically much smaller than its upper bound, the number of preferable functions, which has an upper bound of 2^p, is also much smaller than the number of assignable functions, which is bounded by 2^{2^b}. The computation of the number of assignable and preferable functions is described in detail by Eckl [1995]. The number of preferable functions can still be very large, e.g., it exceeds 10^{18} during decomposition of $\mathbf{f}_{\text{term1}}$. Such a set is still too large to be handled explicitly, and emphasizes the necessity of using implicit techniques.

6. IMPLEMENTATION ASPECTS

The algorithm in Figure 6 shows the three main tasks that must be solved during multiple-output decomposition.

First, the local and global classes must be computed, as well as the sets of global classes that comprise each local class. From this point on, we do not work with bound set vertices any more; instead, global classes are used as the elementary objects.

The second task is the computation of preferable subfunctions. Since this computation is done for each output f_k several (exactly c_k) times, and since the number of preferable functions may be very large, an efficient procedure is needed. We show how to represent and calculate the set of preferable functions implicitly.

The third task is the selection of the "best" function from all computed preferable functions. We are interested in functions that are preferable for a maximum number of outputs f_k. This task can be formulated as finding columns with a maximum number of 1s in a covering table. A new implicit technique is presented.

6.1 Implicit Computation of Preferable Functions

Since the sets of preferable functions may be huge, they must be computed efficiently. Recently, exact solutions to other CAD problems with huge sets were devised using implicit techniques [Coudert et al. 1993; Kam et al. 1994]. A set of objects, e.g., a set of states of a finite state machine, is represented as a minterm using the positional-set notation and a set of object sets is represented by its *characteristic function*. In the decomposition method presented here, global classes are used as elementary objects. A set of global classes defines a constructable subfunction and is represented by a positional minterm. The set of all preferable functions of a function f_k is represented by its characteristic function. Thus, using BDDs, it is possible to represent sets of preferable functions in a very compact way.

According to Definition 5.2, the onset of a constructable function is an element of the power set of $\Pi_{\mathbf{f}}$. So we can employ a bijective mapping π from the set of constructable functions, denoted C, to a p -dimensional Boolean space (where $p = |\Pi_{\mathbf{f}}|$):

$$\pi : C \rightarrow \{0,1\}^p$$

$$\mathbf{z} = (z_1, \ldots, z_p) = \pi(d),\ d \in C,$$

where

$$z_i := \begin{cases} 1 & \text{if } G_i \subseteq on - set(d) \\ 0 & \text{if } G_i \subseteq off - set(d). \end{cases} \tag{25}$$

As an example, the function d_1 in Figure 5, $d_1(\hat{\mathbf{x}}) = 1 \Leftrightarrow \mathbf{x} \in G_2 \cup G_3 \cup G_4$, is represented by $\hat{\mathbf{z}} = (01110)$.

A set of constructable functions is represented by the onset of a Boolean formula, and hence in a single BDD. For example, the set $P_{1,1}$ of preferable functions of Table VIII is represented by

$$\bar{z}_3\bar{z}_4z_5 + z_3z_4\overline{z_5} + \bar{z}_1\bar{z}_2z_5 + z_1z_2\bar{z}_5 + \bar{z}_1\bar{z}_2z_3z_4 + z_1z_2\bar{z}_3\bar{z}_4. \tag{26}$$

This compact representation of sets of preferable functions, however, is only valuable if it is also possible to compute these sets efficiently. An *implicit* computation of preferable functions is proposed here. Implicit computation means that we operate with *sets* of constructable functions, and the number of operations performed is not related to the number of elements contained in the sets.

Theorem 4.3, which is applied in the implicit procedure, gives conditions C0 and C1 for each block X_i of a partial code partition $\Pi_{\mathcal{A}_{f_k, s_k}}$. For ease of notation, we explain the procedure for $s_k = 0$: thus there is only one block $X_1 = \mathcal{X}$ and $X_1/R_{f_k} = \Pi_{f_k} = \{L_1, \ldots, L_{\ell_k}\}$. In a first step, the set T_k of all subsets of Π_{f_k} containing at least $\ell_k - 2^{c_k-1}$ local classes,

$$T_k = \{\mathcal{T} \in \mathcal{P}(\Pi_{f_k}) \,|\, |\mathcal{T}| \geq \ell_k - 2^{c_k-1}\}, \tag{27}$$

is implicitly computed using the *subset algorithm* in Figure 7. The characteristic function $\tau_k(\mathbf{v}) = subset(\ell_k - 2^{c_k-1}, \ell_k)$ represents this set, using a variable v_i for each local class $L_i \in \Pi_{f_k}$. Note that this characteristic function τ_k is a threshold function that 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 $O(\delta \cdot \ell)$.

The two functions $\psi_k^0(\mathbf{z})$ and $\psi_k^1(\mathbf{z})$ are derived from $\tau_k(\mathbf{v})$. The function $\psi_k^0(\mathbf{z})$ represents all subfunctions satisfying condition C0, and $\psi_k^1(\mathbf{z})$ represents all subfunctions satisfying condition C1. Function $\psi_k^0(\mathbf{z})$ is obtained by replacing each v_i-literal (which represents local class L_i) in $\tau_k(\mathbf{v})$ with

```
algorithm subset(δ,ℓ)
{
    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);
}
```

Fig. 7. $subset(\delta, \ell)$ algorithm.

the conjunction of those 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_i-literal in $\tau_k(\mathbf{v})$ with the conjunction of positive z -literals. Constructable functions that satisfy condition C0 and condition C1 are assignable according to Theorem 4.3. Thus, the product of the functions $\psi_k^1(\mathbf{z})$ and $\psi_k^0(\mathbf{z})$ yields the characteristic function $\mathcal{X}_k(\mathbf{z})$ that represents all preferable subfunctions

$$\mathcal{X}_k(\mathbf{z}) = \bar{z}_1 \cdot \psi_k^0(\mathbf{z}) \cdot \psi_k^1(\mathbf{z}).$$

Multiplication with $\bar{z}_1$ is performed to eliminate complementary subfunctions.

For $s_k \neq 0$, the partial partition $\Pi_{\mathcal{A}_{f_k, s_k}}$ consists of several blocks X_i. Then the above procedure must be applied for each block, and the product of all the characteristic functions obtained is $\mathcal{X}_k(\mathbf{z})$.

Example. We now demonstrate how to compute $\mathcal{X}_1(\mathbf{z})$, representing the set $P_{1,1}$ of preferable functions in Table VIII. We have $\tau_1(\mathbf{v}) = subset(1,3) = v_1 + v_2 + v_3$. For $\psi_1^1(\mathbf{z})$, literal v_1 is replaced by z_1z_2, since $L_1 = G_1 \cup G_2$; literal v_2 by z_3z_4, since $L_2 = G_3 \cup G_4$; and literal v_3 by z_5, since $L_3 = G_5$. Thus, $\psi_1^1(\mathbf{z}) = z_1z_2 + z_3z_4 + z_5$. Similarly, for $\psi_1^0(\mathbf{z})$, literal v_1 is replaced by $\bar{z}_1\bar{z}_2$, etc., thus $\psi_1^0(\mathbf{z}) = \bar{z}_1\bar{z}_2 + \bar{z}_3\bar{z}_4 + \bar{z}_5$. Then,

$$\mathcal{X}_1(\mathbf{z}) = \bar{z}_1z_3z_4\bar{z}_5 + \bar{z}_1\bar{z}_2z_5 + \bar{z}_1\bar{z}_3\bar{z}_4z_5.$$

Table II. Covering Table of Preferable Subfunctions for f_1 and f_2

G_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G_2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
G_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
G_4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
G_5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$\mathcal{X}_1(\mathbf{z})$	0	1	0	1	0	1	1	1	0	1	0	0	0	0	1	0
$\mathcal{X}_2(\mathbf{z})$	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0
										$\uparrow$ d_2				$\uparrow$ d_3	$\uparrow$ d_1	

The set $P_{2,1}$ of preferable functions for output f_2 is represented by

$$\mathcal{X}_2(\mathbf{z}) = \bar{z}_1 z_2 z_3 z_4 \bar{z}_5 + \bar{z}_1 z_2 z_3 \bar{z}_4 z_5 + \bar{z}_1 \bar{z}_2 \bar{z}_3 z_4 z_5.$$

The characteristic functions $\mathcal{X}_k(\mathbf{z})$ are shown as row vectors in Table X.

6.2 Implicit Selection of the Best Preferable Subfunction

After computation of the characteristic function $\mathcal{X}_k(\mathbf{z})$ for each output f_k, the task is now to find a $\mathbf{z}$-vertex contained in the onset of a maximum number of $\mathcal{X}_k(\mathbf{z})$. Such a vertex corresponds to a subfunction that decomposes a maximum number of outputs f_k. This problem is visualized in Table II as finding a column with a maximum number of 1s in a covering table, which consists of the rows for $\mathcal{X}_1(\mathbf{z})$ and $\mathcal{X}_2(\mathbf{z})$. A maximum column is marked in Table II by d_1.

A novel algorithm, called *maxcol*, is shown in Figure 8. A single variable w_k is associated with each of the m characteristic functions $\mathcal{X}_k(\mathbf{z})$. The *maxcol* algorithm performs a binary search for the maximum number of rows that can be covered by any of the columns. Each time through the loop, the variable μ is set to the midpoint of the current interval. The *subset*-algorithm is used to compute a function $h_\mu(\mathbf{w})$ with an onset consisting of all $\mathbf{w}$-vertices with at least μ 1s. Substituting the w_k-variables with the corresponding characteristic functions $\mathcal{X}_k(\mathbf{z})$ yields function $h_\mu(\mathbf{z})$. If $h_\mu(\mathbf{z}) = 0$, then there is no $\mathbf{z}$ -vertex contained in the onsets of μ or more characteristic functions $\mathcal{X}_k(\mathbf{z})$, and the right pointer μ_{right} is adjusted. If $h_\mu(\mathbf{z}) \neq 0$, there exists at least one vertex contained in the onsets of at least μ characteristic functions, and the left pointer μ_{left} is adjusted. The algorithm returns a function $h_\mu(\mathbf{z}) \neq 0$ with a maximum value of μ. The onset minterms of $h_\mu(\mathbf{z})$ specify the columns covering a maximum number of rows. Note that these columns may cover different subsets of the row set.

7. VARIABLE AND OUTPUT PARTITIONING

The partitioning of variables into free and bound sets is done with a new iterative improvement method.

```
algorithm maxcol(χ1(z), ..., χm(z))
{
    μleft = 1;
    μright = m;

    while (μleft ≤ μright) {
        μ = (μleft + μright)/2;
        hμ(w) = subset(μ, m);
        for k = 1 to m {
            substitute wk in hμ(w,z) by χk(z);
        }
        if hμ(z) = 0 {
            μright = μ − 1;
        } else {
            μleft = μ + 1;
        }
    }

    if hμ(z) = 0 {
        return hμ−1(z);
    } else {
        return hμ(z);
    }
```

Fig. 8. The *maxcol* algorithm.

Let the single-output function f depend on n variables. The variable partitioning problem is separated into two subtasks, i.e., first, the computation of an optimal variable partition for each bound set cardinality i, $2 \leq i \leq n - 1$; second, the selection of a partition among these optimal partitions. Let us consider the second subtask first.

The key to the selection of the best bound set size and its variable partition is a proper cost function. The cost function estimates the number

of logic modules needed to implement a function of n variables. We use the following cost function, which is an upper bound of the numbers of CLBs (configurable logic blocks) of the Xilinx XC3000 FPGA series. Each XC3000 CLB can implement one Boolean function of up to five variables, or two Boolean functions of up to four variables each that must not use more than five variables in total. The cost function is based on Boole's expansion of f:

$$cost(n) = \begin{cases} 1 & : |n| \leq K \\ \frac{3}{2} \cdot 2^{n-K} & : |n| > K. \end{cases} \tag{28}$$

Here K is the number of LUT inputs. For XC3000, K=5, this cost function considers the mergeability of two functions into one CLB. Using this cost function, the number of modules after a decomposition of f can be estimated for each bound set cardinality b as

$$N_S(f) = cost(n - b + c) + c \cdot cost(b), \tag{29}$$

where $c = \lceil \mathrm{ld}\ell \rceil$ is the number of subfunctions after decomposition with bound set size b. The first term of Eq. (29) accounts for the composition function, the second term for the subfunctions.

The first subtask is solved with an iterative improvement algorithm, which is carried out for each bound set cardinality b. The iterative algorithm starts from an initial partition (with bound set cardinality b and free set cardinality $n - b$) and performs a sequence of cost-reducing variable exchanges between the free set and the bound set. The cost is given by Eq. (29); but since the partitions we deal with have identical bound set size b, the number of local compatible classes suffices to compute relative costs. This number can easily be derived by a traversal of the BDD of f if bound variables are ordered before free variables [Lai et al. 1993]. In each iteration, the *best* free variable and the *best* bound variable are computed. The best bound variable is determined by temporarily moving each bound variable x_α into the free set, thus decreasing the bound set size to $b - 1$ and computing the cost. The best free variable is computed in a similar manner by temporarily moving each free variable y_β into the bound set and computing the cost. The best bound and free variable are exchanged if this yields a cost reduction. Iterations are performed as long as the cost reduces. The initial partition is obtained from a minimal BDD representation of f.

The same two-step approach is adopted for multiple-output decomposition. Note that in the iterative improvement algorithm of the first step, a variable may be included in the bound set only if it is in support of all the functions of a vector.

To estimate the number of modules after multiple-output decomposition of vector $\mathbf{f} = (f_1, \ldots, f_m)$ with a bound set of cardinality b, we have the following, straightforward extension of Eq. (29):

$$N_M(\mathbf{f}) = \sum_{k=1}^{m} cost(|sup(f_k)| - b + c_k) + q \cdot cost(b). \tag{30}$$

Here, $c_k = \lceil \mathrm{ld}\ell_k \rceil$ is the number of subfunctions needed for output f_k. However, we do not know the exact value of the number of subfunctions q, $\lceil \mathrm{ld}p \rceil \leq q \leq \sum_{k=1}^{m} \lceil \mathrm{ld}\ell_k \rceil$ after variable partitioning. This is a significant difference from single-output decomposition, where the number of subfunctions is known after variable partitioning.

Therefore, we use upper bound $\sum_{k=1}^{m} c_k$ on the number of subfunctions, and obtain a cost function

$$N'_M(\mathbf{f}) = \sum_{k=1}^{m} (cost(|sup(f_k)| - b + c_k) + c_k \cdot \mathrm{cost}(b)). \tag{31}$$

Equation (31) is used during the iterative variable exchanges as well as for the final selection of a bound set cardinality.

Please keep in mind that the variable partition computed for a vector $\mathbf{f}$ may not be optimal for its components f_k. So the number of subfunctions needed to decompose function f_k as a component of vector $\mathbf{f}$ may be larger than the number of subfunctions needed for a separate, single-output decomposition of f_k. This effect reduces the gain achieved by multiple-output decomposition below the subfunction sharing gain of Definition 5.5.

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

8. RESULTS

The algorithms for functional multiple-output decomposition proposed in this paper were implemented in program *IMODEC* (Implicit Multiple-Output DEComposition). The following sections first compare our new approach to variable partitioning and then benchmark IMODEC against various state-of-the-art technology mappers for Xilinx XC3000 and XC4000 FPGAs.

8.1 An Overall Algorithm for Technology Mapping to LUT-based FPGAs

The overall procedure for technology mapping to LUT architectures follows. Program IMODEC is used to recursively decompose infeasible nodes until a K-bounded network is obtained. In a postprocessing step, we further try to reduce the LUT-count: a node is collapsed into its fanout nodes if this step yields a network that is still K-bounded. Finally, LUTs are merged into CLBs, as allowed by Xilinx FPGAs.

8.2 Variable Partitioning

Table III compares different approaches to variable partitioning for single-output decomposition. Shen et al. advocate a constructive approach to determining bound and free sets for single-output decomposition [Shen et al. 1995]. This approach delivers the best results in variable partitioning, in comparison to previously presented approaches. In Table III, the results obtained by Shen et al. are compared to that advocated in this paper.

For this experiment, our tool IMODEC was run in single-output mode. Both the decomposition used by Shen et al. and IMODEC in single-output mode employ *strict* functional decomposition. Results are therefore comparable and strongly depend on the various algorithms for variable partitioning.

The comparison is performed for a set of two-level circuits. Results for the Shen et al. method are cited in Shen et al. [1995]. The best result for each circuit is printed **boldface** type.

The table shows that the approach to variable partitioning presented in this paper outperforms that proposed by Shen et al. Especially for larger circuits, the quality of results obtained by IMODEC dominates those of Shen et al.

8.3 Technology Mapping for LUT-Based FPGAs

A variety of technology mapping algorithms targeting LUT architectures has been presented in the last five years. Table IV shows a comparison of program *IMODEC* with other mapping algorithms targeted at reducing area. Promising approaches dedicated to achieving performance optimization [Murgai et al. 1991b; Cong et al. 1993] are not considered in this comparison. The comparison is based on a set of benchmark circuits also used in an overview article by Sangiovanni-Vincentelli et al. [1993]. The tables give the CLB count after mapping to Xilinx XC3000 FPGAs. Since in many publications only a small subset of these benchmarks is used, the last row gives the ratio of the sum of CLB counts for the used subset to the corresponding sum of CLB counts obtained with *IMODEC*. Again,the best result for each circuit is printed in **boldface** type.

Before comparing the results, one should note that the results are sensitive to the starting networks. In most publications, MIS [Brayton et al. 1987] and SIS [Sentovich et al. 1992] scripts, respectively, were used to obtain these networks. Most researchers obtain starting networks by applying just one of the SIS-scripts once. In the sequel, the procedure used

Table III. Different Variable Partitioning Approaches: LUT Count

circuit	[Shen et al. 1995]	IMODEC
5xp1	**19**	**19**
9sym	**6**	7
alu2	77	**55**
apex4	426	**364**
b9	92	**57**
clip	36	**24**
count	52	**40**
duke2	722	**256**
e64	544	**389**
f51m	**16**	**16**
misex1	**16**	17
misex2	43	**40**
rd73	**8**	**8**
rd84	**13**	**13**
sao2	37	**25**
z4ml	**6**	7
Σ	2113	1337
relative to IMODEC	1.58	1.00

to obtain the starting networks is only mentioned if it is not one of the MIS/SIS-scripts.

Sangiovanni-Vincentelli et al. [1993] compare the approaches of *MIS-pga2* [Murgai et al. 1990; 1991a];[3] *Chortle-crf* [Francis et al. 1991]; *Xmap* [Karplus 1991]; and *Hydra* [Filo et al. 1991], none of which focus on functional decomposition techniques. Their finding is that for identical starting networks, *MIS-pga2* outperforms the other approaches. Therefore, in Table IV, only *MIS-pga2* is included, along with more recent approaches. Compared with *IMODEC*, *MIS-pga2* consumes on average 32% more CLBs.

The *ASYL* program decomposes functions by Boole's expansion and various kernel extraction schemes that also try to consider routability. For each circuit, the best result of either Babba and Crastes [1992] or Abouzeid et al. [1993] is given. Starting networks have two levels [Abouzeid et al. 1993] or are not specified further [Babba and Crastes 1992]. Unfortunately, no results have been published for large circuits like `apex4`, `des` and `rot`. ASYL requires 46% more CLBs than *IMODEC*.

The remaining more recent mapping algorithms (with the exception of *SIS-1.3*) are based exclusively on functional decomposition and a relatively simple covering step. A more detailed description of the various decomposition methods is given in Section 2.

The results in column *SIS-1.3* were obtained by experiments with program SIS-1.3 [Sentovich et al. 1992], using the LUT-oriented synthesis

[3]Since only a LUT result but no CLB result is given in Sangiovanni-Vincentelli et al. [1993] for circuit `des`, the CLB value for `des` is estimated from the LUT count of 904, using the average ratio of 87/100 for CLB vs. LUT counts observed in other results of *MIS-pga2*.

Table IV. CLB Count for Xilinx XC3000 FPGAs

circuit	MIS-pga2	ASYL	MIS-94	SIS-1.3	FGMap	FGSyn bx-csn	mulop2	IMODEC	CPU [s]
5xp1	13	13	40	13	15	**9**	**9**	**9**	0.9
9sym	**7**	8	53	**7**	**7**	**7**	**7**	**7**	0.3
alu2	96	60	163	76	53	55	51	**48**	105.6
alu4	**49**	250	85	151	-	56	-	53	9.3
apex2	60	69	133	**56**	-	60	-	77	2.1
apex4	371	-	-	372	356	-	-	**3**33	56.1
apex6	165	156	207	154	-	181	-	**1**40	9.6
apex7	43	42	52	47	47	43	45	**4**0	0.9
b9	32	**18**	77	28	27	28	30	26	1.3
clip	23	33	73	26	20	18	14	**1**2	5.2
count	30	28	26	29	24	**23**	26	2	8.9
des	(786)	-	1265	710	-	-	-	**4**46	304.4
duke2	94	**82**	269	109	178	85	114	**8**2	1.9
f51m	15	14	41	11	11	**8**	**8**	9	0.7
misex1	9	13	10	9	**8**	**8**	9	**8**	0.3
misex2	25	23	27	**2**1	**2**1	22	24	**2**	0.4
rd73	**5**	8	25	**5**	7	**5**	**5**	**5**	0.6
rd84	9	14	67	10	12	**8**	**8**	9	1.6
rot	143	-	240	140	194	136	146	**1**28	5.9
sao2	28	30	78	28	27	25	20	**1**7	4.6
vg2	18	20	23	19	23	**1**7	18	**1**7	1.3
C499	66	-	68	66	**4**9	54	60	50	1.1
C880	**7**2	-	191	76	74	87	87	74	6.8
Σ	2159	881	3213	2163	1153	935	681	1636	
relative to IMODEC	1.32	1.46	2.47	1.32	1.25	1.09	1.16	1.00	

script in Sentovich et al. [1992]. On average, *SIS-1.3* yields mapped circuits with 32% more CLBs than *IMODEC*.

The technology mapping algorithm *FGMap* presented by Lai et al. [1993] uses functional single-output decomposition for technology mapping. *FGMap* is a significant advance because it produces better CLB results than *SIS-1.3* in an order of magnitude less CPU time.

MIS-94 shows the results of an approach by Murgai that extends the sum of product-based functional decomposition algorithm in MIS [Murgai et al. 1994]. This algorithm carries out nonstrict decompositions with the goal of obtaining composition functions with a minimal literal count. This method's relatively large CLB counts are due to the simple variable partitioning approach. Moreover, even for small and well-decomposable circuits like `9sym` and `rd84`, a multiple-level network is used as a starting point. All the other methods in this table use two-level starting networks for the circuits. The large CLB counts of *MIS-94* illustrate that even the extended functional decomposition algorithm in MIS/SIS is inferior to the BDD-based approaches. Thus, the good results of *SIS-1.3/MIS-pga2* must be attributed to the skillful combination of functional decomposition with algebraic extraction.

Table V. CLB Count for Xilinx XC4000 FPGAs

circuit	FGSyn	IMODEC
5xp1	15	**8**
9sym	**6**	7
alu2	53	**46**
alu4	51	**47**
apex2	**53**	67
apex6	128	**105**
apex7	37	**35**
b9	24	**20**
clip	24	**12**
count	19	**16**
duke2	76	**70**
f51m	10	**8**
misex1	**8**	**8**
misex2	19	**17**
rd73	6	**4**
rd84	10	**7**
rot	119	**105**
sao2	29	**15**
vg2	16	**14**
C880	72	**60**
Σ	768	671
relative to IMODEC	1.14	1.00

Columns *FGSyn* [Lai et al. 1996] and *mulop2* [Scholl and Molitor 1994] show results for the two multiple-output decomposition algorithms discussed in Section 2. Finally, the last columns *IMODEC* and *CPU* give results and CPU times (measured on a DEC AlphaStation 250-4/266 with 128Mbyte memory) of the algorithm described in this article. The experiments show that *IMODEC* outperforms *FGSyn* by 9%. Unfortunately, only the results for small benchmark circuits are given for *mulop2*. Considering these circuits only, circuits mapped by *mulop2* consume 15% more CLBs than those mapped by *IMODEC*.

Table V shows a comparison of *FGSyn* (best of bx4, bx4-T1, bx4-T2 from Table III in Lai et al. [1996]) and *IMODEC* for technology mapping to Xilinx XC4000 FPGAs. As these results show, *IMODEC* reduces the number of XC4000 CLBs on average by 14% compared with *FGSyn*.

To summarize, functional multiple-output decomposition by *IMODEC* applied to technology mapping for LUT-based FPGAs yields

—area improvements of more than 30% over the popular MIS/SIS tool and state-of-the-art single-output decomposition.

—average improvements of 9% to 16% over other multiple-output decomposition approaches.

9. CONCLUSION

This article proposes a new theory and algorithms for functional multiple-output decomposition. Variable partitioning is performed using a novel,

BDD-based, iterative improvement method. The new multiple-output decomposition core combines high quality results with computational efficiency, due to three characteristics of the algorithm:

The algorithm performs the *nonstrict* decomposition of each component of a multiple-output function. Simple circuit examples illustrate that strict decomposition prevents the identification of subfunctions common to several outputs. Thus, nonstrict decomposition is necessary to find all existing common subfunctions. Encoding each bound set vertex separately, however, would make nonstrict decomposition infeasible for large bound set cardinalities. As a major theoretical contribution, it is proven that encoding the bound set vertices of a global compatible class by the same code is sufficient for an optimum decomposition, i.e., for finding a maximum number of common subfunctions. Thus, subfunctions that are built from classes of the global partition (*constructable* subfunctions) are sufficient for an optimum multiple-output decomposition. The practical value of this contribution follows from the experimental evidence that, for typical Boolean functions of combinational circuits, the rank of the global compatibility partition is usually much smaller than the number of bound set vertices. This implies a significant reduction in the average complexity of a decomposition algorithm.

The algorithm applies an *iterative* computation of scalar subfunctions. The functions that are suitable in each iteration are called *assignable*. Since it has been shown that constructable subfunctions suffice to obtain an optimum decomposition, in each iteration the algorithm computes only assignable as well as constructable (called *preferable*) subfunctions. The iterative computation of subfunctions has several advantages. First, since preferable subfunctions are computed for each output separately, finding common subfunctions reduces to computing the cutset of sets of preferable functions. Second, the rules to compute preferable (and in general assignable) functions are simple. Therefore, sets of preferable functions can be computed rapidly. Third, further properties of subfunctions can easily be incorporated into this approach.

The representation and computation of preferable subfunctions is *implicit*. A set of preferable subfunctions is represented by its characteristic function, which is then represented by a BDD. The size of the BDD in almost all practical cases is acceptable even if the number of preferable functions is huge. The compactness of representation reflects the simple construction rule for the set of preferable functions. An efficient algorithm for implicit computation has been developed.

Additionally, a method is presented to determine the partitioning of a function's variables into bound and free variables for decomposition. The method consists of selecting a cost function keyed to the target architecture and an effective iterative improvement step. It has been shown to outperform previous approaches to this problem.

A comparison of the multiple-output decomposition algorithm with state-of-the-art technology mappers shows significant module count reductions.

Table VI. Set D_1 of Functions Assignable w.r.t. $\mathcal{A}_{f,0}$

compatible class	L_1				L_2			L_3
bound set vertex	(000)	(001)	(010)	(100)	(011)	(101)	(110)	(111)
row 1	-	-	-	-	0	0	0	1
row 2	-	-	-	-	1	1	1	0
row 3	0	0	0	0	-	-	-	1
row 4	1	1	1	1	-	-	-	0
row 5	0	0	0	0	11		1	-
row 6	1	1	1	1	0	0	0	-

APPENDIX

SINGLE-OUTPUT DECOMPOSITION

For function f_1 shown in Figure 2(a), $\ell = 3$ and $c = 2$. Thus, a function d is assignable in the first iteration if and only if $on\text{-}set(d)$ as well as $off\text{-}set(d)$ completely contain at least $3 - 2^{2-1} = 1$ compatible class.

Table VI shows all these assignable functions. Each column is associated with a bound set vertex. The vertices are ordered according to their compatibility. In each row, the 0 entries specify offset vertices of an assignable function $d : \{0,1\}^3 \rightarrow \{0,1\}$, and the 1 entries give onset vertices of an assignable function. The - entries specify bound set vertices that may belong to either $on\text{-}set(d)$ or $off\text{-}set(d)$. For example, row 6 describes the functions

$$\overline{x_1x_2} + \overline{x_1x_3} + \overline{x_2x_3} \tag{32}$$

(by putting vertex (111) into the offset) and

$$\overline{x_1x_2} + \overline{x_1x_3} + \overline{x_2x_3 +} \; x_1x_2x_3 \tag{33}$$

(by putting vertex (111) into the onset). We obtain the subfunction d_1 of Figure 3,

$$d_1(\mathbf{x}) = x_1\overline{x_2x_3} + x_1x_2x_3,$$

from row 1, by putting vertices (000), (001), (010) into the offset and vertex (100) into the onset.

The partial assignment is $\mathcal{A}_{f,1} = \{d_1\}$. The partial code partition is $\Pi_{\mathcal{A}_{f,1}} = \Pi_{d_1} = \{X_1, X_2\}$, where $X_1 = on\text{-}set(d_1)$ and $X_2 = off\text{-}set(d_1)$. For $X_1/R_{f_1} = \{L_1^1, L_2^1\}$, we have $L_1^1 = \{(100)\}$ and $L_2^1 = \{(111)\}$. For $X_2/R_{f_1} = \{L_1^2, L_2^2\}$, we have $L_1^2 = \{(000), (001), (010)\}$ and $L_2^2 = \{(011), (101), (110)\}$.

We now compute the second subfunction d_2. Both X_1 and X_2 contain vertices of $k = 2$ compatible classes: $|X_1| = |X_2| = 2$. Thus, in accordance to Theorem 4.3, at least $k - 2^{c-s-1} = 2 - 2^{2-1-1} = 1$ class L_i^j must be completely contained in the offset and onset of the second subfunction d_2.

Table VII. Set D_2 of Functions Assignable w.r.t. $\mathcal{A}_{f,1}$

partial code class	X_1		X_1					
partial compatible class	L_1^1	L_2^1	L_1^2			L_2^2		
bound set vertex	(100)	(111)	(000)	(001)	(010)	(011)	(101)	(110)
row 1	0	1	0	0	0	1	1	1
row 2	1	0	0	0	0	1	1	1
row 3	0	1	1	1	1	0	0	0
row 4	1	0	1	1	1	0	0	0

Since each block X_1 and X_2 contains vertices of two compatible classes, we do not have the degrees of freedom to put bound set vertices into either the onset or the offset, as expressed by the dashes in Table VI. Table VII shows all functions that are assignable in the second step. We obtain the subfunction d_2 of Figure 3,

$$d_2(\mathbf{x}) = \overline{x_1}x_2x_3 + x_1\overline{x_2} + x_1\overline{x_3},$$

from row 2, and thus have the assignment $\mathcal{A}_f = \{d_1, d_2\}$. The code partition $\Pi_{\mathcal{A}_f} = \Pi_{\mathcal{A}_{f,2}} = \Pi_{d_1} \cdot \Pi_{d_2}$ is shown in Figure 3(b).

MULTIPLE-OUTPUT DECOMPOSITION

We apply the algorithm to the function vector (f_1, f_2) of Figure 2 and show how to compute the subfunctions of Figure 5. The local compatibility partitions are Π_{f_1} with $\ell = 3$ classes and Π_{f_2} with $\ell_2 = 4$ classes. Each function must be decomposed with $c_1 = c_2 = 2$ subfunctions.

The global compatibility partition $\Pi_{\mathbf{f}}$ with $p = 5$ classes is also shown in Figure 5. The next step is the computation of the functions that are initially preferable, which are shown in Tables VIII and IX. Each column is associated with a global class. The bound set vertices contained in a global class and the global classes contained in a local class are shown. In each row, the 0 entries specify global classes that are put into the offset of a preferable function $d : \{0,1\}^3 \rightarrow \{0,1\}$, and the 1 entries denote global classes that belong to the onset of a preferable function. The - entries specify global classes that may be put *completely* into either $on\text{-}set(d)$ or $off\text{-}set(d)$.

Note how Table VIII is obtained from Table VI by merging columns associated with globally compatible vertices. Since we now compute preferable functions instead of generally assignable functions, we deal with global classes instead of individual vertices.

Another representation of the preferable functions is given in Table X. Since there are 5 classes, there are 32 constructable functions. Each column of the table represents a constructable function. Vertical lines are drawn only to improve readability. Each of the first five rows is associated with a global class G_i. Thus, the five topmost entries of each column define

Table VIII. Set $P_{1,1}$ of Preferable Functions for Output f_1

local class	L_1		L_2		L_3
global class	G_1	G_2	G_3	G_4	G_5
bound set vertex	(000)	(001),(010),(100)	(011)	(101),(110)	(111)
row 1	-	-	0	0	1
row 2	-	-	1	1	0
row 3	0	0	-	-	1
row 4	1	1	-	-	0
row 5	0	0	1	1	-
row 6	1	1	0	0	-

Table IX. Set $P_{2,1}$ Preferable Functions for Output f_2

local class	L_1	L_2		L_3	L_4
global classes	G_1	G_2	G_3	G_4	G_5
bound set vertex	(000)	(001),(010),(100)	(011)	(101),(110)	(111)
row 1	0	0	0	1	1
row 2	1	1	1	0	0
row 3	0	1	1	0	1
row 4	1	0	0	1	0
row 5	0	1	1	1	0
row 6	1	0	0	0	1

a constructable function, where a 0 specifies a global class contained in the offset, and a 1 specifies a global class contained in the onset of the function. Rows six and seven of the table give the functions that are preferable for the outputs f_1 and f_2 in the first iteration, where a 1 entry in a column indicates that the constructable function associated with that column is assignable.

Apparently, f_1 has 14 and f_2 has 6 preferable functions. Both outputs have four preferable functions in common, corresponding with columns 4, 15, 18 and 29. We select as the first subfunction

$$d_1(\mathbf{x}) = x_1\overline{x_3} + \overline{x_2}x_3 + \overline{x_1}x_2,$$

which corresponds to column 15, i.e., $on\text{-}set(d_1) = G_2 \cup G_3 \cup G_4$.

Thus, the partial assignments are $\mathcal{A}_{f_1,\,1} = \mathcal{A}_{f_2,\,1} = \{d_1\}$. The partial code partitions are $\Pi_{\mathcal{A}_{f_1,\,1}} = \Pi_{\mathcal{A}_{f_2,\,1}} = \Pi_{d_1} = \{X_1, X_2\}$, where $X_1 = on\text{-}set(d_1)$ and $X_2 = off\text{-}set(d_1)$. Analogously to single-output decomposition, the sets of functions preferable in the second iteration, $P_{1,2}$ and $P_{2,2}$, are shown in Tables XI and XII.

These functions are also given in the last two rows of Table X. Note that $P_{1,2} \subset P_{1,1}$ and $P_{2,2} \subset P_{2,1}$. The function associated with column 10, which is preferable for f_1, is selected as the second subfunction:

$$d_2(\mathbf{x}) = x_1x_2x_3 + x_1\overline{x_2x_3} + \overline{x_1x_2}x_3 + \overline{x_1}x_2\overline{x_3}.$$

Table X. Preferable Functions for f_1 and f_2 In Terms of Global Classes

G_1	00000000	00000000	11111111	11111111
G_2	00000000	11111111	00000000	11111111
G_3	00001111	00001111	00001111	00001111
G_4	00110011	00110011	00110011	00110011
G_5	01010101	01010101	01010101	01010101
for output f_1: $P_{1,1}$	01010111	01000010	01000010	11101010
for output f_2: $P_{2,1}$	00010000	00000110	01100000	00001000
for output f_1: $P_{1,2}$	00000001	01000000	00000010	10000000
for output f_2: $P_{2,2}$	00010000	00000100	00100000	00001000

Table XI. Set $P_{1,2}$ of Preferable Functions

partial code class	X_1		X_2		
partial local class	L_1^1	L_2^1	L_1^2	L_2^2	
global class	G_1	G_5	G_2	G_3	G_4
row 1	0	1	0	1	1
row 2	1	0	0	1	1
row 3	0	1	1	0	0
row 4	1	0	1	0	0

Table XII. Set $P_{2,2}$ of Preferable Functions

partial code class	X_1		X_2		
partial local class	L_1^1	L_2^1	L_1^2		L_2^2
global class	G_1	G_5	G_2	G_3	G_4
row 1	0	1	0	0	1
row 2	1	0	0	0	1
row 3	0	1	1	1	0
row 4	1	0	1	1	0

Now, since $s_1 = c_1 = 2$, this selection completes the local assignment $\mathcal{A}_{f_1} = \{d_1, d_2\}$. Note that the local code partition $\Pi_{\mathcal{A}_{f_1}}$ *refines* its compatibility partition Π_{f_1}, i.e., we have performed a nonstrict decomposition.

We finally select the function associated with column 14, which is preferable for f_2, as the third subfunction

$$d_3(\mathbf{x}) = x_2x_3 + x_1\overline{x_2x_3} + \overline{x_1}x_2 + \overline{x_1}x_3.$$

This completes the local assignment $\mathcal{A}_{f_2} = \{d_1, d_3\}$. Note that the local code partition $\Pi_{\mathcal{A}_{f_2}}$ equals the compatibility partition Π_{f_2}, i.e., we have actually performed a strict decomposition for this output, which is necessary due to $\ell_2 = 2^{c_2}$.

REFERENCES

ABOUZEID, P., BABBA, B., DE PAULET, M. C., AND SAUCIER, G. 1993. Input-driven partitioning methods and applications to synthesis on table-lookup-based FPGA's. *IEEE Trans. Comput.-Aided Des. Integr. Circuits 12*, 7, 913–925.

ASHENHURST, R. L. 1953. Non-disjoint decomposition. Harvard Computation Laboratory Rep. No. BL-4, Sec. IV. Harvard Univ., Cambridge, MA.

ASHENHURST, R. L. 1959. The decomposition of switching functions. *Ann. Comput. Lab. Harvard Univ. 29*, 74–116.

BABBA, B. AND CRASTES, M. 1992. Automatic synthesis on table lookup-based PGAs. In *Proceedings of the European Conference on Design Automation*. 25–31.

BRAYTON, R., RUDELL, R., SANGIOVANNI-VINCENTELLI, A., AND WANG, A. 1987. MIS: A multiple-level logic optimization system. *IEEE Trans. Comput.-Aided Des. Integr. Circuits 6* (Nov.), 1062–1080.

CONG, J. AND DING, Y. 1996. Combinational logic synthesis for LUT based field programmable gate arrays. *ACM Trans. Des. Autom. Electron. Syst. 1*, 2, 145–204.

CONG, J., DING, Y., GAO, T., AND CHEN, K.-C. 1993. An optimal performance-driven technology mapping algorithm for LUT-based FPGAs under arbitrary net-delay models. In *Proceedings of the International Conference on Computer-Aided Design and Computer Graphics* (Aug. 1993). 599–604.

COUDERT, O., MADRE, J. C., AND FRAISSE, H. 1993. A new viewpoint on two-level logic minimization. In *Proceedings of the 30th International Conference on Design Automation* (DAC'93, Dallas, TX, June 14–18), A. E. Dunlop, Ed. ACM Press, New York, NY, 625–630.

CURTIS, H. A. 1961. A generalized tree circuit. *J. ACM 8*, 484–496.

CURTIS, H. A. 1962. *A New Approach to the Design of Switching Circuits*. Van Nostrand Reinhold Co., New York, NY.

ECKL, K. 1995. Funktionale Dekomposition von Booleschen Funktionsbuendeln. Master's Thesis. Institute for Design Automation, Technical University of Munich, Munich.

FILO, D., YANG, J. C.-Y., MAILHOT, F., AND MICHELI, G. D. 1991. Technology mapping for a two-output RAM-based field programmable gate array. In *Proceedings of the European Conference on Design Automation* (EDAC, Feb. 1991). 534–538.

FRANCIS, R., ROSE, J., AND VRANESIC, Z. 1991. Chortle-crf: Fast technology mapping for lookup table-based FPGAs. In *Proceedings of the 28th ACM/IEEE Conference on Design Automation* (DAC '91, San Francisco, CA, June 17–21), A. R. Newton, Ed. ACM Press, New York, NY, 227–233.

HWANG, T., OWENS, R. M., AND IRWIN, M. J. 1992. Efficient computing communication complexity for multilevel logic synthesis. *IEEE Trans. Comput.-Aided Des. Integr. Circuits 11*, 5 (Oct.), 545–554.

HWANG, T.-T., OWENS, R. M., AND IRWIN, M. J. 1990. Exploiting communication complexity for multilevel logic synthesis. *IEEE Trans. Comput.-Aided Des. Integr. Circuits 9*, 10 (Oct.), 1017–1027.

HWANG, T.-T., OWENS, R. M., IRWIN, M. J., AND WANG, K. H. 1994. Logic synthesis for field-programmable gate arrays. *IEEE Trans. Comput.-Aided Des. Integr. Circuits 13*, 10 (Sept.), 1280–1287.

KAM, T., VILLA, T., BRAYTON, R., AND SANGIOVANNI-VINCENTELLI, A. 1994. A fully implicit algorithm for exact state minimization. In *Proceedings of the 31st Annual Conference on Design Automation* (DAC'94, San Diego, CA, June 6–10, 1994), M. Lorenzetti, Ed. ACM Press, New York, NY, 684–690.

KARP, R. M. 1963. Functional decomposition and switching circuit design. *J. Soc. Ind. Appl. Math. 11*, 2, 291–335.

KARPLUS, K. 1991. Xmap: A technology mapper for table-lookup field-programmable gate arrays. In *Proceedings of the 28th ACM/IEEE Conference on Design Automation* (DAC '91, San Francisco, CA, June 17–21), A. R. Newton, Ed. ACM Press, New York, NY, 240–243.

LAI, Y.-T., PEDRAM, M., AND VRUDHULA, S. B. K. 1993. BDD based decomposition of logic functions with application to FPGA synthesis. In *Proceedings of the 30th International Conference on Design Automation* (DAC'93, Dallas, TX, June 14–18), A. E. Dunlop, Ed. ACM Press, New York, NY, 642–647.

LAI, Y.-T., PAN, K.-R. R., AND PEDRAM, M. 1994. FPGA synthesis using function decomposition. In *Proceedings of the IEEE International Conference on Computer Design* (Cambridge, MA, Oct. 10-12). 30–35.

LAI, Y.-T., PAN, K.-R. R., AND PEDRAM, M. 1994. FPGA synthesis using OBDD-based function decomposition. Tech. Rep.. Computer Science Department, University of Southern California, Los Angeles, CA.

LAI, Y.-T., PAN, K.-R. R., AND PEDRAM, M. 1996. OBDD-based function decomposition: Algorithms and implementation. *IEEE Trans. Comput.-Aided Des. Integr. Circuits 15*, 8, 977–990.

MOLITOR, P. AND SCHOLL, C. 1994. Communication based multilevel synthesis for multi-output boolean functions. In *Proceedings of the Fourth Great Lakes Symposium on VLSI* (Notre Dame, IN, Mar. 4-5, 1994).

MURGAI, R., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1994. Optimum functional decomposition using encoding. In *Proceedings of the 31st Annual Conference on Design Automation* (DAC'94, San Diego, CA, June 6–10, 1994), M. Lorenzetti, Ed. ACM Press, New York, NY, 408–414.

MURGAI, R., NISHIZAKI, Y., SHENOY, N., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1990. Logic synthesis for programmable gate arrays. In *Proceedings of the ACM/IEEE Conference on Design Automation* (DAC '90, Orlando, FL, June 24-28), R. C. Smith, Ed. ACM Press, New York, NY, 620–625.

MURGAI, R., SHENOY, N., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1991. Improved logic synthesis algorithms for table look up architectures. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '91, Santa Clara, CA, Nov. 11-14). IEEE Computer Society Press, Los Alamitos, CA, 564–567.

MURGAI, R., SHENOY, N., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1991. Performance directed synthesis for table look up programmable gate arrays. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '91, Santa Clara, CA, Nov. 11-14). IEEE Computer Society Press, Los Alamitos, CA, 572–575.

ROTH, J. P. AND KARP, R. M. 1962. Minimization over boolean graphs. *IBM J. Res. Dev.*, 227–238.

SANGIOVANNI-VINCENTELLI, A., GAMAL, A. E., AND ROSE, J. 1993. Synthesis methods for field programmable gate arrays. *Proc. IEEE 81*, 7 (July 1993), 1057–1083.

SASAO, T. 1993. *Logic Synthesis and Optimization*. Kluwer Academic Publishers, Hingham, MA.

SCHLICHTMANN, U. 1993. Boolean matching and disjoint decomposition. In *Proceedings of the IFIP Workshop on Logic and Architecture Synthesis* (Dec.). IFIP, 83–103.

SCHOLL, C. AND MOLITOR, P. 1994. Efficient ROBDD based computation of common decomposition functions of multi-output boolean functions. In *Proceedings of the IFIP Workshop on Logic and Architecture Synthesis* (Nov.). IFIP, 61–70.

SENTOVICH, E., SINGH, K., LAVAGNO, L., MOON, C., MURGAI, R., SALDANHA, A., SAVOJ, H., STEPHAN, P., BRAYTON, R., AND SANGIOVANNI-VINCENTELLI, A. 1992. SIS: A system for sequential circuit synthesis. Tech. Rep. UCB/ERL M92/41. UC Berkeley, Berkeley, CA.

SENTOVICH, E. M., SINGH, K. J., MOON, C., SAVOJ, H., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1992. Sequential circuit design using synthesis and optimization. In *Proceedings of the 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors* (Cambridge, MA, Oct. 11–14). IEEE Computer Society, Washington, DC, 328–333.

SHEN, W.-Z., HUANG, J.-D., AND CHAO, S.-M. 1995. Lambda set selection in Roth-Karp decomposition for LUT-based FPGA technology mapping. In *Proceedings of the 32nd ACM/IEEE Conference on Design Automation* (DAC '95, San Francisco, CA, June 12–16, 1995), B. T. Preas, Ed. ACM Press, New York, NY, 65–69.

YANG, S. 1991. Logic synthesis and optimization benchmarks, user guide version 3.0. In *Proceedings of the International Workshop on Logic Synthesis*. MCNC, Research Triangle Park, NC.

Received: October 1996; revised: August 1997; accepted: January 1998