VLSI DESIGN
1995, Vol. 3, Nos. 3-4, pp. 289-300
Reprints available directly from the publisher
Photocopying permitted by license only

A General Approach to Boolean Function Decomposition and its Application in FPGA-Based Synthesis

TADEUSZ ŁUBA AND HENRY SELVARAJ
Warsaw University of Technology, Institute of Telecommunications, Nowowiejska 15/19, 00–665 Warsaw, Poland

An effective logic synthesis procedure based on parallel and serial decomposition of a Boolean function is presented in this paper. The decomposition, carried out as the very first step of the synthesis process, is based on an original representation of the function by a set of r-partitions over the set of minterms. Two different decomposition strategies, namely serial and parallel, are exploited by striking a balance between the two ideas. The presented procedure can be applied to completely or incompletely specified, single- or multiple-output functions and is suitable for different types of FPGAs including XILINX, ACTEL and ALGOTRONIX devices. The results of the benchmark experiments presented in the paper show that, in several cases, our method produces circuits of significantly reduced complexity compared to the solutions reported in the literature.

Key Words: *Decomposition, Boolean function, Partition, FPGA, PLD*

1. INTRODUCTION

The dominant direction of growth in electronics industry nowadays is in the area of highly functional and universal programmable devices, in particular, user programmable circuits like PLDs and FPGAs. Several CAD systems developed in the 1980s and aimed at programmable logic have been very effective in significantly reducing the number of circuits required for implementing a given logic function using PLDs, but not so effective for FPGA-based implementations. This has been a primary reason for recent interest in FPGA based logic synthesis.

There are different types of FPGA structures with different number of inputs and outputs and the design strategies for those cells differ from one another. Most design methods are based on Multilevel Boolean Networks [4], functional decomposition [2], [6], [8] or Binary Decision Diagrams (BDDs) [9]. Recently published papers try to improve functional decomposition procedures to make them applicable to Look Up Table (LUT) structures [8], [27]. Their promising results seem to indicate that the concept of functional decomposition should be investigated more generally and in more detail. The only disadvantage of the functional decomposition is its restriction to only one type of FPGA architecture, i.e. LUT structures.

The intention of this paper is to develop a general method of decomposition for different types of FPGAs, for which until now there is no common design procedure.

Therefore the proposed design technique relies exclusively on the functional capabilities of FPGA logic cells. In other words, the logic cell is treated as a universal cell capable of implementing any Boolean function with fixed number of inputs and outputs. Such an assumption raises the possibility of developing a method applicable to a variety of FPGA structures. This assumption restricts the logic synthesis strategies mainly to functional decomposition methods i.e. to the process of reexpressing a function of n variables as a function of functions of fewer variables. For example, a function F(X) is decomposable if it can be expressed as F = H(A,G(B)), where A and B are proper subsets of the set of input variables X, G and H are components of F, and H has fewer input variables than F.

Numerous decomposition algorithms have been developed. Ashenhurst, in his fundamental paper [2], stated

the disjunctive decomposition theorem based on the notion of decomposition charts. Curtis extended the Ashenhurst's results to multiple decomposition when F if expressed as $F = H(A, G_1(B),\ldots,G_k(B))$ [6]. The use of decomposition charts for functional decomposition of logic networks is applicable only to restricted classes of functions. Therefore, a number of decomposition improvements were suggested. In the Roth-Karp scheme, a more compact representation of a function in the form of a cover of the on-set and a cover of the off-set has been used [23]. Later, in the early 70's, an attempt was made to apply orthogonal transform techniques to the design of digital circuits. The problem of constructing optimal decomposition schemes using spectral techniques was also considered [13]. Recently, the spectral approach has been improved and employed in the so called "groupability" method intended for FPGAs [8].

In the early 80's, functional decomposition methods lost their importance because of the rapid development of synthesis techniques for multilevel logic. Algebraic division of sum-of-products expressions represented by the sets of cubes has been a basic operation in the procedures of substitution and kernel extraction used for decomposition of Boolean functions [4].

Since the late 80's logic decomposition has been again attracting some attention as a technique used for design of PLAs. Devadas et al. proposed a Boolean decomposition of a PLA into two cascaded PLAs [7]. This procedure is however conceptually more similar to multiple-valued symbolic minimization than to the classical decomposition. Moreover, the method is confined to PLA synthesis and cannot be considered as a general functional decomposition approach.

Logic decomposition can play an important role in the design of FPGA-based circuits because their structure imposes constraints on the number of inputs only and the two-level minimization is not needed. However, the multilevel synthesis became so deeply rooted that earlier synthesis methods for FPGAs were based on the traditional, multilevel minimization approach.

This approach was used in MIS-PGA [20], Hydra [10] and Chortle [11]. In the ASYL system, multilevel synthesis was improved by using the idea of lexicographical order [25]. Functional decomposition has been sometimes used in FPGA design, but only as an auxiliary process, as in MIS-PGA and Hydra systems. So far, there is only one FPGA-based technology mapper, namely TRADE, that fully exploits the idea of functional decomposition [27]. Its promising results seem to indicate that the concept of functional decomposition should be investigated more generally and in more detail.

Following this trend, we propose an original decomposition method which in contrast improves functional decomposition through interleaving two different strategies of decomposition and by applying an original calculus based on the representation of a function by a family of partitions over the set of cubes. Our decomposition procedure is universal, i.e., it can be applied to completely or incompletely specified, single- or multiple-output functions. Thus, it favorably compares with the earlier methods, often limited to either single-output or completely specified functions.

The paper is organized as follows. In Section 2, we introduce the basic notions and discuss the partition-based representation of a Boolean function. In Section 3, the theoretical fundamentals of the decomposition methods are given. Section 4 shows how the proposed decomposition algorithm is used for synthesis of different types of FPGAs. The presented results of the benchmark design experiments demonstrate that, in several cases, our method produces circuits of significantly reduced complexity compared to the earlier reported solutions.

2. BASIC NOTIONS

An incompletely specified Boolean function F of n input variables and m output variables is defined as a mapping $F:\{0,1\}^n \rightarrow \{0,1,-\}^m$. The value "$-$" (don't care) at one of the outputs means that a 0 or a 1 will be accepted as the response of the circuit realizing this component of the function.

As there is a natural correspondence between an input vector, a vertex in the n-cube and a minterm, we refer to elements of the domain $\{0,1\}^n$ of F as minterms. The list of minterms with the corresponding values of the function is called the truth table of F. For an incompletely specified function, the truth table usually does not include minterms for which all outputs take "don't care values".

Let M be the set of minterms in the truth table, i.e. minterms for which at least one component of F is specified.

Let X be the set of input variables and Y the set of output variables of F. Let $B \subseteq X$, and $m \in M$. Denote by m_B an element of $\{0,1\}^{|B|}$ ($|B|$ denotes the cardinality of B) obtained by deleting from m the values of input variables not contained in B. For example, for $X = \{x_1,x_2,x_3,x_4,x_5\}$, $B = \{x_1,x_2,x_4\}$, and $m = 10101$, we have $m_B = 100$.

In general, any pair of minterms may have identical values for some number of input variables. A convenient way to reflect the similarity of two minterms is to use the so called indiscernibility relation [22], [16]. This relation, denoted by IND, is defined as follows.

Let $B \subseteq X$ and $m', m'' \in M$.

Then

$$(m', m'') \in IND(B) \text{ iff } m_B' = m_B''.$$

In other words, $(m', m'') \in IND(B)$ if the values of the arguments (input variables) from B are identical for both m' and m". Minterms m' and m" are said to be indiscernible by arguments of B. The indiscernibility relation is an equivalence relation on M and

$$IND(B) = \bigcap_{x \in B} IND(x).$$

Thus, the relation IND partitions M into equivalence classes M/IND(B). We denote partition M/IND(B) by P(B) and call such a partition an input partition induced by B. Thus

$$P(B) = \bigsqcap_{x \in B} P(x)$$

where $\sqcap$ denotes the product of partitions.

To define another relation on the set of minterms, we introduce the concept of consistent values of function F. We say that two values of F (output vectors) $z', z'' \in \{0,1,-\}$ are consistent, which is denoted $z' \sim z''$, if for each output variable their respective components corresponding to that variable, if both specified, are equal. For example, 10-1 is consistent with 1-11 (10-1 $\sim$ 1-11) but 10-1 is not consistent with 00-1.

The output consistency relation, CON, is defined on the set of minterms as follows.

Let $m', m'' \in M$. $(m', m'') \in CON$ iff $F(m') \sim F(m'')$.

We say that a set of minterms $M' \subseteq M$ constitutes a consistent class if every pair of minterms in M' is output-consistent. An output-consistent class that is not a subset of any other output-consistent class is called a Maximal Consistent Class. For a given function, the set of all MCCs is unique.

Clearly, MCCs are not disjoint and therefore the output consistency relation is not an equivalence relation on M. Hence, it "partitions" M into non-disjoint subsets. Nevertheless, to describe the output consistency relation, we use the same notation as for the indiscernibility relation, i.e. P_F is an output "partition" (index F is intended to distinguish between the input (IND) and output (CON) relation). A "partition" with non-disjoint blocks and such that no block is a subset of some other block is referred to as a rough partition (r-partition). The concept of an r-partition is a simple extension of that of an ordinary partition and typical operations on r-partitions are the same as used in the ordinary partition algebra [12]. In particular, the relation *less than or equal to* holds between two r-partitions Π_1 and Π_2 ($\Pi_1 \preceq \Pi_2$) iff for every block of Π_1, $B_i(\Pi_1)$, there exists a block of Π_2, $B_j(\Pi_2)$, such that $B_i(\Pi_1) \subseteq B_j(\Pi_2)$.

Example 1

Consider the incompletely specified Boolean function F defined in Table 1.

We have:

$M = \{1,\ldots,10\}$,
$X = \{x_1,x_2,x_3,x_4,x_5,x_6\}$,
$Y = \{y_1,y_2,y_3,y_4\}$.

The IND relations for $\{x_1\}$ and $\{x_2,x_3\}$ are:

$P(x_1) = \{ \{1,6,7,10\}, \{2,3,4,5,8,9\} \}$,
$P(x_2,x_3) = \{ \{1,3,4\}, \{2,9\}, \{5,7,10\}, \{6,8\} \}$.

The output-consistency relation CON is given by the following r-partition:

$$P_F = \{ \{1,6\}, \{3\}, \{2,4\}, \{5,6\}, \{6,9\}, \{4,8,9\}, \{7,10\} \}.$$

In the further chapters the blocks of partitions are separated by semicolons only and not by curly parenthesis, for example

$$P_F = (1,6 ; 3 ; 2,4 ; 5,6 ; 6,9 ; 4,8,9 ; 7,10).$$

3. LOGIC DECOMPOSITION

Along with the widely known and frequently used serial decomposition, this paper proposes to interleave a new strategy of decomposition called parallel decomposition. Both the strategies are illustrated in Fig.1.

The parallel decomposition leads to a structure in which two "independent" components operate in parallel (Fig. 1b), whereas the serial decomposition results in

TABLE 1
Truth Table of Example 1

	x_1	x_2	x_3	x_4	x_5	x_6	y_1	y_2	y_3	y_4
1	0	1	0	0	0	1	0	0	0	–
2	1	0	0	0	1	1	0	0	1	–
3	1	1	0	1	1	1	1	1	1	0
4	1	1	0	1	0	0	–	0	1	1
5	1	1	1	1	1	1	1	0	–	0
6	0	0	1	0	1	1	–	0	0	–
7	0	1	1	0	0	1	0	1	–	1
8	1	0	1	1	1	0	1	–	1	1
9	1	0	0	1	1	0	1	0	–	1
10	0	1	1	1	0	1	0	1	1	1

serially connected components (Fig. 1c). In this paper, we focus on the more difficult problem of serial decomposition. In a short subsection on the parallel decomposition, we merely formulate the problem and show an example illustrating its solution.

3.1. Parallel Decomposition

Consider a multiple-output Boolean function F. Assume that F has to be decomposed into two components, G and H, with disjoint sets of output variables, Y_G and Y_H. This task occurs, for example, when we want to implement a large function using PLDs with a limited number of outputs. It must be observed that such a parallel decomposition can also alleviate the problem of excessive number of inputs of F. This is because, for typical functions, most outputs do not depend on all input variables. Therefore, the set of input variables on which the outputs in Y_g depend, X_g, may be smaller than X, i.e., $pX_gp \leq pXp$. Similarly, for the set of input variables on which the outputs in Y_h depend, X_h, we have $pXp_h \leq pXp$. As a result, components G and H have not only fewer outputs than F, but also fewer inputs. The exact formulation of the parallel decomposition problem depends on the constraints imposed by the implementation style. One possibility is to find the sets Y_g and Y_h, such that the combined cardinality of X_g and X_h is minimal. An efficient procedure for this type of parallel decomposition has been presented in [14], [17].

Partitioning of the set of outputs into only two disjoint subsets is not an important limitation of the method because the procedure can be applied again for components G and H. It is essential, however, to require that no logic minimization be performed before the parallel decomposition. This is because two-level procedures often produce a representation in which outputs explicitly depend on more inputs than in the original functional description, which makes the decomposition more difficult.

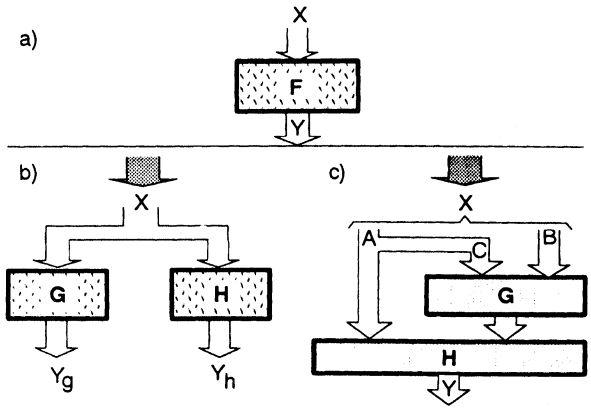

FIGURE 1 Parallel and serial decomposition of function F.

Example 2

Consider the multiple-output Boolean function F given in Table 2. The minimal dependence sets for single-output components of F, i.e., the minimal sets of input variables the outputs depend on, are:

$y_1 : \{x_1,x_2,x_6\}$,
$y_2 : \{x_3,x_4\}$,
$y_3 : \{x_1,x_2,x_4,x_5,x_9\}, \{x_1,x_2,x_4,x_6,x_9\}$
$y_4 : \{x_1,x_2,x_3,x_4,x_7\}$,
$y_5 : \{x_1,x_2,x_4\}$,
$y_6 : \{x_1,x_2,x_6,x_9\}$,

An optimal two-block decomposition, minimizing the cardinality of the sets of input variables, is $Y_g = \{y_2, y_4, y_5\}$ and $Y_h = \{y_1, y_3, y_6\}$, with $X_g = \{x_1, x_2, x_3, x_4, x_7\}$ and $X_h = \{x_1, x_2, x_4, x_6, x_9\}$. The truth tables of the components G and H are shown in Table 3.

In many situations, the parallel decomposition can be used as a subsidiary step to the more general, serial decomposition procedure.

3.2. Serial Decomposition

Let F(X) denote a Boolean function F with the set of input variables X. Let $X = A \cup B$ and $C \subseteq A$.

The problem of serial decomposition of F(X) is illustrated in Fig. 1c. We try to find functions G and H, such that G depends on variables in $B \cup C$, whereas H depends on variables in A and variables in Z, where Z is the set of output variables of G. For each minterm, the value of H is consistent with the value of F.

A more formal definition of serial decomposition is presented below.

We say that there exists a serial decomposition of F if there exist functions G and H

$$G: \{0,1\}^{|B \cup C|} \longrightarrow \{0,1\}^{|Z|}$$

$$H: \{0,1\}^{|A \cup Z|} \longrightarrow \{0,1,-\}^{|Y|}$$

such that, for each $m \in M$, $H(m_A, G(m_{B \cup C}))$ is consistent with F(m) and for each specified (0 or 1) variable in F(m) the corresponding variable in $H(m_A, G(m_{B \cup C}))$ is also specified. If set C is an empty set, the decomposition is called a disjoint serial decomposition otherwise it is called a non-disjoint serial decomposition.

The following theorem states the necessary and sufficient condition for the existence of a serial decomposition.

TABLE 2
Truth Table of Example 2

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_1	y_2	y_3	y_4	y_5	y_6
1	0	0	0	1	1	1	0	0	0	0	0	0	0	–	0
2	1	0	1	0	0	0	0	0	0	0	0	–	1	0	1
3	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1
4	1	1	1	1	0	1	0	0	0	0	1	1	1	1	0
5	1	0	1	0	1	0	0	0	0	0	0	0	–	0	1
6	0	0	1	1	1	0	0	0	0	1	1	0	1	0	0
7	1	1	1	0	0	0	0	0	0	1	0	–	0	1	0
8	1	0	1	1	0	1	0	0	0	1	1	0	0	–	1
9	1	0	1	1	0	1	1	0	0	–	1	0	1	–	1
10	1	1	1	0	0	0	0	1	0	1	0	1	0	1	–
11	0	0	0	1	1	1	0	0	1	0	0	1	0	–	1
12	0	0	0	1	1	0	0	0	1	–	–	1	0	0	0

Theorem 1: There exists a serial decomposition of F iff there exists a partition Π_G, such that

(a) $\Pi_G \geq P(B \cup C)$, and
(b) $P(A) \bullet \Pi_G \leq P_F$.

Proof: ($\Rightarrow$) For function G, define partition Π_G as follows: m' and m" are in one block of Π_G iff G $(m'_{BUC}) = G(m''_{BUC})$. It must be observed that, since G is completely specified, the blocks of Π_G are disjoint, and Π_G is a uniquely defined partition (not r-partition). We will show that Π_G satisfies conditions (a) and (b) stated in the Theorem.

By definition of the IND relation, for any two minterms m' and m" included in one block of P(B $\cup$ C), we have $m'_{BUC} = m''_{BUC}$ Thus $\Pi_G \geq P(B \cup C)$, which means that Π_G satisfies condition (a).

Define now an r-partition Π_H that corresponds to the CON relation for function H, i.e., each block of Π_H is a maximal subset of minterms which are pairwise output-consistent (m' and m" are output-consistent if $H(m'_A, G(m'_{BUC})) \sim H(m''_A, G(m''_{BUC}))$. By definition of the serial decomposition, we have

$$\Pi_H \leq P_F. \quad (1)$$

Consider two minterms, m' and m", included in one block of $P(A) \bullet \Pi_G$. By definition of the IND relation, $m'_A = m''_A$, and by definition of Π_G, $G(m'_{BUC}) = G(m''_{BUC})$. Thus, by definition of Π_H, m' and m" are in one block of Π_H, which implies

$$\Pi_H \geq P(A) \bullet \Pi_G. \quad (2)$$

Combining (1) and (2), we obtain

$$P(A) \bullet \Pi_G \leq P_F,$$

which means that Π_G satisfies condition (b).
($\Leftarrow$) Given partition Π that satisfies (a) and (b), define function G with $\lceil \log_2 p\Pi p \rceil$ output variables for which $G(m'_{BUC}) = G(m''_{BUC})$ iff m′ and m" are in the same block of Π.

This construction of G is always possible because $\Pi \geq P(B \cup C)$, i.e., $(m'_{BUC}) = (m''_{BUC})$ implies that m′ and m" are in the same block of Π. Clearly, this construction is not unique, i.e., many different functions G can be found for a given partition Π.

TABLE 3a
Results of Parallel Decomposition of Table 2

x_1	x_2	x_3	x_4	x_7	y_2	y_4	y_5
0	0	0	1	0	0	0	0
1	0	1	0	0	0	1	0
1	0	1	1	0	1	0	1
1	1	1	1	0	1	1	1
1	0	1	0	0	0	–	0
0	0	1	1	0	1	1	0
1	1	1	0	0	0	0	1
1	0	1	1	0	1	0	–
1	0	1	1	1	–	1	–

TABLE 3b
Results of Parallel Decomposition of Table 2

x_1	x_2	x_4	x_6	x_9	y_1	y_3	y_6
0	0	1	1	0	0	0	0
1	0	0	0	0	0	0	1
1	0	1	0	0	0	1	1
1	1	1	1	0	0	1	0
1	0	1	0	0	1	0	0
0	0	0	0	0	1	1	0
1	1	1	1	0	1	0	1
1	0	1	1	1	0	1	1
1	0	1	0	1	–	1	0

Let M′ be a subset of minterms in one block of P_F. Denote by F(M′) an output vector consistent with F(m) for each m ∈ M′ and having a "−" component whenever possible.

Define function H as follows: for each m ∈ M, $H(m_A,z) = F(M'')$ where M" is that block of P(A) • Π which includes $\tilde{m} \in M$, such that $\tilde{m}_A = m_A$ and $G(\tilde{m}_{B\cup C}) = z$.

This construction of H is possible because:

> P(A) • Π ≤ P_F, which implies that M" is included in a single block of P_F, $\tilde{m}_A = m_A$ implies that $\tilde{m}$ and m are in the same block of P(A); recalling the definition of G this, in turn, implies that $(\tilde{m}_A,z)$ and (m_A,z) are in the same block of P(A) • Π.

By definition of H, we have either $H(m_A,G(m_{B\cup C})) = F(m)$ or some variables not specified in F(m) are specified in $H(m_A,G(m_{B\cup C}))$. This shows that F can be serially decomposed into G and H, which completes the proof.

As shown in the proof, partition Π in Theorem 1 represents function G. In particular, the number of blocks in Π, p Π p, determines the number of output variables of G, $\lceil \log_2 p\Pi p \rceil$.

Example 3

Consider the function F of 5 input variables and 3 output variables shown in Table 4.

For A = $\{x_1,x_3,x_4\}$, B = $\{x_2,x_5\}$, C = ϕ, we have

P(A) = (1,7 ; 8,13 ; 2,3 ; 9,14,15 ; 4,5 ; 10 ; 6 ; 11,12)
P(B) = (1,3,15 ; 2,13,14 ; 4,6,7,8,9,10,12 ; 5,11)
and

TABLE 4
Truth Table of Example 3 and 4

	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3
1	0	0	0	0	0	0	0	0
2	0	0	0	1	1	0	1	0
3	0	0	0	1	0	1	−	0
4	0	1	1	0	0	0	1	1
5	0	1	1	0	1	0	0	1
6	0	1	1	1	0	−	1	0
7	0	1	0	0	0	0	0	1
8	1	1	0	0	0	0	−	−
9	1	1	0	1	0	0	0	0
10	1	1	1	0	0	1	0	0
11	1	1	1	1	1	0	1	1
12	1	1	1	1	0	−	1	−
13	1	0	0	0	1	0	0	1
14	1	0	0	1	1	−	−	0
15	1	0	0	1	0	1	0	0

P_F = (1,8,9,14 ; 2,6,8,12,14 ; 3,6,12,14 ; 3,10,14,15 ; 4,8,11,12 ; 5,7,8,13)

Consider

Π_G = (1,3,5,11,15 ; 2,4,6,7,8,9,10,12,13,14).

As

P(A) · Π_G = (1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8,13 ; 9,14 ; 10 ; 11 ; 12 ; 15),

we have

$$P(A) \cdot \Pi_G \leq P_F.$$

Thus, according to Theorem 1, function F is decomposable as $F = H(x_1,x_3,x_4,G(x_2,x_5))$, where G is a single-output function.

The main task is to find the subset of inputs B ∪ C for component G which, when serially connected with component H will implement function F.

Consider the subset of input variables, B ∪ C, and the corresponding partition P(B ∪ C). The relation of compatibility of partition blocks is used to verify whether or not partition P(B ∪ C) defines a viable decomposition.

Two blocks $B_i,B_j \in$ P(B ∪ C) are compatible if partition P_{ij}(B ∪ C) obtained from P(B ∪ C) by merging blocks B_i and B_j into a single block satisfies condition (b) in Theorem 1, i.e., iff

$$P(A) \bullet P_{ij}(B \cup C) \leq P_F.$$

A subset of partition blocks in P(B ∪ C) is a compatible class if all blocks in this subset are pairwise compatible. A compatible class is called a Maximal Compatible Class (MCC) if it is not a subset of any other compatible class.

To obtain Π_G that satisfies conditions of Theorem 1, we first find all MCCs for P(B ∪ C) and then find a minimum cover of the set of all blocks of P(B ∪ C) by MCCs. Partition Π_G is subsequently formed by merging blocks in each MCC in the minimal cover and making the resulting blocks disjoint.

Partition Π_G represents function G corresponding to the assumed set B ∪ C. In particular, the number of blocks in Π_G, $p\Pi_G p$, determines the number of outputs of G, m_G ($m_G = \lceil \log_2 p\Pi_G p \rceil$).

Once a suitable set B ∪ C and the corresponding partition Π_G are found, the truth tables of functions G and H can be easily derived from P(A), Π_G, and P_F.

Example 4

For the function of Example 3, let A = $\{x_3,x_4\}$, B = $\{x_1,x_2,x_5\}$, and C = ϕ. Then,

P(A) = (1,7,8 ,13 ; 2,3,9,14,15 ; 4,5,10 ; 6,11,12)

and

$P(B \cup C) =$
$(1,3 ; 2 ; 4,6,7 ; 5 ; 8,9,10,12 ; 11 ; 13,14 ; 15).$

Let us check if $B_1 = \{1,3\}$ and $B_2 = \{2\}$ are compatible. We have

$P_{12}(B \cup C) =$
$(1,2,3 ; 4,6,7 ; 5 ; 8,9,10,12 ; 11 ; 13,14 ; 15).$

As

$P(A) \bullet P_{12}(B \cup C) =$
$(1 ; 2,3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; 10 ; 11 ; 12 ; 13 ; 14 ; 15)$

does not satisfy

$P(A) \bullet P_{12}(B \cup C) \leq P_F$ (for P_F, see Example 4),

B_1 and B_2 are not compatible.

For $B_1 = \{1,3\}$ and $B_4 = \{5\}$, we obtain

$P_{14}(B \cup C) =$
$(1,3,5 ; 2 ; 4,6,7 ; 8,9,10,12 ; 11;13,14 ; 15)$

and

$P(A) \bullet P_{14}(B \cup C) =$
$(1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; 10 ; 11 ; 12 ; 13 ; 14 ; 15)$
$\leq P_F.$

Thus, B_1 and B_4 are compatible. In a similar way we check the compatibility of each pair of blocks in $P(B \cup C)$ and then find Maximal Compatible Classes:

$MCC1 = \{B_4,B_6,B_7,B_8\},$
$MCC2 = \{B_1,B_4,B_6,B_8\},$
$MCC3 = \{B_2,B_4,B_6,B_7\},$
$MCC4 = \{B_3,B_7,B_8\},$
$MCC5 = \{B_2,B_3,B_7\},$
$MCC6 = \{B_5,B_7\}.$

For the above obtained MCCs, one of the minimal covers is $\{\{B_1,B_4,B_6,B_8\}, \{B_2,B_3,B_7\}, \{B_5,B_7\}\}$, and the corresponding Π_G is

$$\Pi_G = (1,3,5,11,15 ; 2,4,6,7 ; 8,9,10,12,13,14).$$

Thus

$$F=H(x_3,x_4,G(x_1,x_2,x_5)),$$

where G is a 2-output function.

4. DECOMPOSITION ALGORITHM

One important aspect of serial decomposition is the task of selecting 'best candidate' variables for the G function. Decomposition is essentially a process of substituting two or more input variables with a lesser number of new variables. This substitution results in the reduction of the number of rows in the truth table. Hence, we look for variables which are most likely to reduce the number of rows in the truth table as a result of decomposition. Let us consider an input variable purposely avoiding all inter-relationships among the input variables. The only available parameter to evaluate its "activity" is the number of '1's or '0's it has in the truth table. If the variable has only '1's or '0's it is the "best candidate" for decomposition, as it is practically redundant. On the other hand, with the decrease in the difference between the number of '0's and '1's, the chances for a reduction in the number of rows decreases. And hence, the variable having the largest difference in the number of '0's and '1's ought to be the best candidate for decomposition. However, as our experience shows, with this greedy selection strategy, decomposition becomes increasingly difficult with each successive design step. So, the input variables are ordered from the 'candidate with the least chance' to the 'candidate with the best chance'.

The method is architecture independent and can be applied to any type of Logic Block (LB) with a random number of inputs and as well as a random number of outputs. The method looks at the Logic Block as a device capable of implementing any function with a fixed number of inputs and outputs.

Consider a function F: $\{0,1,-\}^n \rightarrow \{0,1,-\}^m$. Let us try to implement the function using a hypothetical logic block with C_{in} inputs and C_{out} outputs, where C_{in} and C_{out} are integers and $C_{in} > C_{out}$.

Unlike the traditional approach of first performing logic minimization and then mapping the design using decomposition and other methods, it is proposed to start with the mapping process straight away using different decomposition strategies discussed in the earlier chapters. The effectiveness of such an approach has been proved by Łuba et al. [15], [17], [18].

The first step is to test the sensibility of performing disjoint serial decomposition. As the given LB has C_{in} inputs and C_{out} outputs, the ideal serial decomposition is such a decomposition, where the number of G block inputs G_{in} is equal to the number of LB inputs C_{in} and the number of G block outputs G_{out} is equal to the number of LB outputs C_{out}. If a serial decomposition is attempted with G_{in} and G_{out} as the number of G block inputs and outputs respectively, the resulting H block will have $n - (G_{in}-G_{out})$ inputs. If this number of inputs is not greater than m, then it is proposed to split the truth table into two parts using parallel decomposition. In other words, if the number of inputs to the resulting H block is not expected to be greater than the number of its outputs, the next step is to split the truth table into two parts using parallel decomposition. Each part then is decomposed separately using the same method being discussed.

Based on the above "balancing idea" a decomposition algorithm can be developed and a simplified version of it is presented in Fig. 2.

For clarity of presentation it is assumed that the decomposed truth table has more inputs than outputs ($n > m$) and that C_{in} and C_{out} are assumed to be 2 and 1 respectively.

In the algorithm, after every disjoint serial decomposition the new truth table H with n' inputs and m' outputs is taken for further consideration. If necessary parallel decomposition is performed before continuing the iteration with disjoint serial decomposition. For the case under consideration if the eventual H truth table has 3 inputs and 1 output and there is no disjoint serial decomposition, then non-disjoint serial decomposition is performed to complete the process. Several examples are presented which illustrate in detail the working of the algorithm under different circumstances.

Example 5

Consider a function F: $\{0,1,-\}^n \rightarrow \{0,1,-\}^m$ as defined in table 5, where n and m are equal to 3 and 2 respectively. Let us decompose it for cells (LBs) with

TABLE 5
Truth Table of Example 5

	x_0	x_1	x_2	y_0	y_1
0	1	0	–	0	1
1	1	–	1	0	1
2	–	1	–	0	–
3	0	1	–	0	0
4	–	1	1	0	–
5	–	1	0	0	0
6	0	0	–	1	0
7	0	–	–	–	0

2-inputs and 1-output, i.e. $C_{in} = 2$ and $C_{out} = 1$.

The G block parameters are assigned the values of the assumed cell parameters, i.e. $G_{in} = 2$ and $G_{out} = 1$. Now the algorithm performs the "sensibility" test for disjoint serial decomposition. If such a serial decomposition exists, then the anticipated H block will have $(n - (G_{in}-G_{out}))$ two inputs and two outputs. As the number of H block inputs is not expected to be greater than the number of its outputs, the algorithm chooses parallel decomposition. Parallel decomposition partitions the set of outputs into two disjoint subsets and finds their minimal input support sets. The minimal support sets for the two components of F are:

$$y_0 : \{x_0,x_1)$$
$$y_1 : \{x_0,x_1x_2).$$

Now, the algorithm decomposes the two functions separately using the same iteration procedure. As the function y_0 depends only on two inputs, it can be directly implemented using a single given cell. The function y_1 is decomposed separately and its solution is given in Fig. 3.

If it is found that a serial decomposition with G_{in} and G_{out} will produce a H truth table with more number of inputs than outputs, the algorithm attempts to find such a decomposition. The G_{in} number of variables are selected on the basis of the established input variable ordering. All possible combinations are checked until a successful solution is found such that:

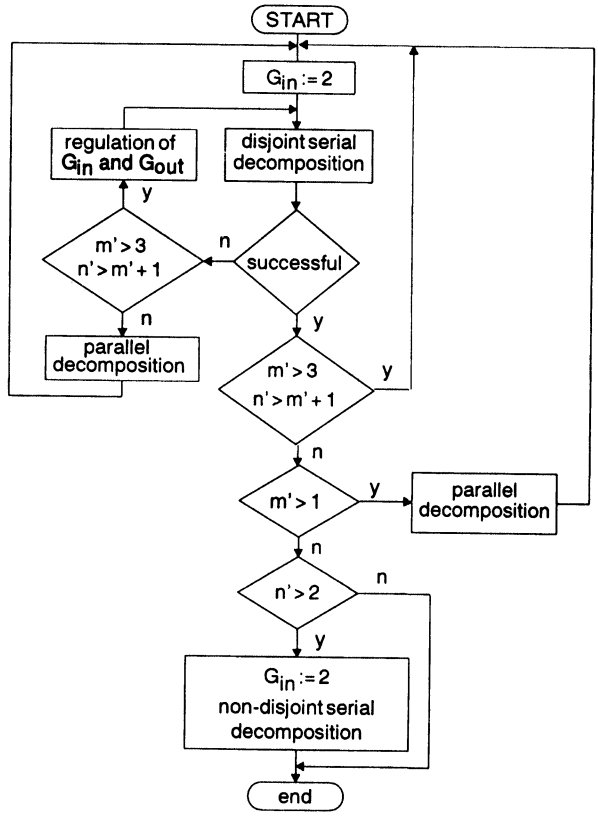

FIGURE 2 The algorithm of decomposition.

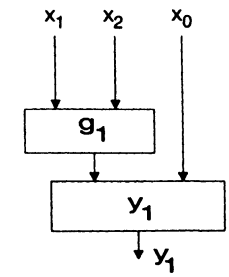

FIGURE 3 Function of Example 5.

$$F = H(A, G(x_{i1},\ldots,x_{iGin})),$$

where $x_{i1},\ldots,x_{iGin} \in B$.

The G function is implemented by a LB and the H function is considered for further decomposition in the same way all over again, this time with the truth table H (with n' inputs and m' outputs) as the starting point instead of the truth table F. This way, by interleaving parallel and serial decomposition strategies, required solution can be found in many cases.

Example 6

Consider the function F as defined in table 6. Let us try to implement the function using Logic Blocks with two inputs and one output.

The algorithm sets the value of G_{in} and G_{out} as 2 and 1 respectively and looks for the existence of a disjoint serial decomposition. Such a decomposition exists for the inputs x_0 and x_2, i.e. $F = H(x_1,x_3,f_0)$ where $f_0 = G(x_0,x_2)$. The function G can be implemented by the given LB and the function H with three inputs is considered for further decomposition. Again, such a decomposition exists for the inputs x_1 and x_3, i.e $H = H'(f_0,f_1)$ where $f_1 = G(x_1,x_3)$. The implementation is shown in fig. 4.

However, it is more interesting to analyse a situation when the number of inputs to the truth table under consideration is greater than C_{in}, parallel decomposition is not recommended on the basis of the "sensibility" test described earlier and the iteration is not able to produce any further result.

In such a situation the algorithm regulates the parameters G_{in} and G_{out} as G'_{in} and G'_{out} respectively and continues with serial decomposition to look for the existence of any possible G' function such that:

$$F' = H(A, G'(x_{i1},\ldots,x_{iG'in})),$$

where F' is the function under consideration and $x_{i1},\ldots,x_{iG'in} \in B$.

If a successful decomposition is found and the number of inputs to the H truth table continues to be more than C_{in} (number of inputs to the given logic block), the iteration is continued with the H truth table as the new starting point after resetting the number of G block inputs to C_{in} and the number of G block outputs to C_{out}.

The G' function is decomposed separately, using the same iteration method, so that:

$$G' = H(A,G(x_{i1},\ldots,x_{iGin})).$$

One important aspect of the algorithm not discussed so far is the procedure for regulating the G block input/output parameters G_{in} and G_{out}. If C_{in} and C_{out} are the number of inputs and outputs of the Logic Block for which the synthesis is done, initially G_{in} and G_{out} assume the values C_{in} and C_{out} respectively. Now, let us consider the situation when there is no further serial decomposition and the algorithm does not recommend parallel decomposition. Under such a situation, it is proposed to increament the value of G_{in} by one i.e., $G'_{in} = G_{in} + 1$. However, before increasing the number of G block inputs, it is made sure that for the given number of inputs G_{in}, there is no serial decomposition for $G_{out} = C_{out}$, $C_{out} + 1,..G_{in} - 1$. For a LB with $C_{in} - C_{out} > 1$, the possibility of the existence of a decomposition is verified for $G_{in} = C_{in}$, $C_{in}-1,..G_{out} + 1$.

For example, if the selected FPGA LB has 2 inputs and 1 output then the consecutive G block parameters G_{in} and G_{out} shall be 2,1; 3,1; 3,2; 4,1; 4,2; 4,3; 5,1 etc. On the other hand, if the selected FPGA LB has 4 inputs and 1 output then the consecutive G block parameters G_{in} and G_{out} shall be 4,1; 3,1; 2,1; 4,2; 3,2; 4,3; 5,1 etc.

TABLE 6
Truth Table of Example 6

	x_0	x_1	x_2	x_3	y
0	0	–	1	0	0
1	1	–	0	0	0
2	0	1	1	1	0
3	1	0	1	1	0
4	0	0	0	1	0
5	1	1	0	1	0
6	1	1	1	–	1
7	0	–	0	0	1
8	1	0	1	0	1
9	0	0	1	1	1
10	0	1	0	1	1
11	1	0	0	1	1

Example 7

Consider the function in Table 7. The initial attempts to decompose the function with the values of G_{in} and G_{out} set as 2,1 and 3,1 do not yield any result. Hence G_{in} and G_{out} are regulated to 3 and 2 respectively and the

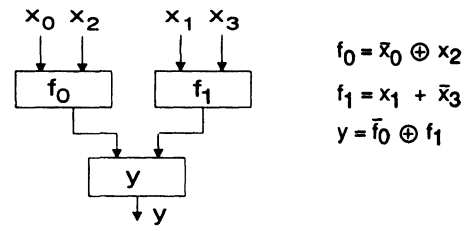

FIGURE 4 Function of Example 6.

iteration is continued. Such a decomposition exists for the inputs x_1, x_2 and x_3, i.e $F = H(x_0,G'(x_1,x_2,x_3)$. The function H with three inputs (g_1,g_2 and x_0) is decomposed further after resetting the values of G_{in} and G_{out} to 2 and 1 respectively. Again, such a decomposition exists for the inputs g_2 and x_0, i.e $H = H'(g_1,f_0)$ and the iteration is continued until a complete solution is obtained. The resulting solution is presented in Fig. 5a.

Functions f_0, f_1 and f_2 are
$f_0 = g'_2 \bullet x_0$; $f_1 = g'_1 \bullet \bar{x}_0$; $f_2 = f_0 \oplus f_1$.

The three-input two-output function G' is decomposed separately using the same algorithm and its implementation is presented in Fig. 5b where f_3, f_4, f_5 and f_6 are
$f_3 = x_1 \oplus x_3$ $f_4 = x_1 \bullet x_2$
$f_5 = f_3 \bullet \bar{x}_2$ $f_6 = x_4 + x_3$.

5. EXPERIMENTAL RESULTS

The described method has been implemented in a prototype decomposition program. The input to the program is a truth table and the output is a network of n-input m-output cells, each realising an n-variable function of m outputs. However the numbers n, m can be fixed arbitrarily, in the present version we assumed $2 \leq n \leq 5$ and $1 \leq m \leq 2$ as it covers all the existing applications. Results of such a decomposition for some known benchmarks are presented in Table 8 and compared with the results obtained by Woo [21].

Because of the variable number of cell inputs, the applications are more wide than in other FPGA-based tools. This is because the logic cell is treated as a universal cell capable of implementing any n-input m-output Boolean function. For example, in the XILINX 3000 family, an FPGA consists of up to 320 Configurable Logic Blocks (CLBs). Each CLB can be programmed to implement any single-output function of up to 5 input variables or any two-output function of up to 5 variables, with each output depending on at most 4 input variables. Similarly, the ACTEL cell which consists of three interconnected 2-to-1 multiplexers can be treated as a cell capable of implementing any Boolean function of two variables v_1 and v_2. The internal structure of the cell is irrelevant for our analysis. This leads us to conclude that the analysis is valid for other types of FPGAs, like those of Algotronix and Quick Logic. An Algotronix FPGA consists of logic cells, each of which includes several multiplexers driving a Function Unit [5]. The Function Unit is capable of implementing any logic function of its x_1 and x_2 inputs and hence it is a 'universal logic cell' implementing any Boolean function of two variables. Also, the structure of Quick Logic FPGA [5] is very similar to that of ACTEL and in the same way its cell is capable of implementing any Boolean function of two variables v_1 and v_2.

TABLE 7
Truth Table of Example 7

	x_0	x_1	x_2	x_3	y_1
0	0	0	0	0	0
1	1	0	0	1	0
2	0	0	0	1	0
3	1	0	1	0	1
4	0	0	1	0	1
5	1	0	1	1	0
6	1	0	0	0	1
7	1	1	0	0	1
8	1	1	0	1	0
9	1	1	1	0	0
10	0	1	1	1	1
11	1	1	1	1	0
12	0	1	0	1	0

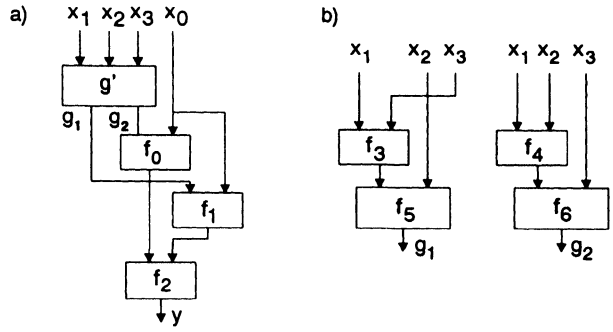

FIGURE 5 Function of Example 7.

Table 9 shows the results of decomposition of the benchmark circuits into five-input two-output cells, which is in fact the decomposition aimed at the Xilinx Logic Blocks. The comparison of out results with the other published results shows that the proposed method does not suffer because of its universality but, in fact, provides better solutions in many cases.

Table 10 contains the number of cells obtained by programs dedicated to multiplexer-based FPGA cells also and, hence, utilizing the unique characters of their structures. It is worth noting that, in spite of the universal character of our method, the results obtained are not worse than the results of the other programs dedicated exclusively to multiplexer-based circuits. Better results of the ASYL program, dedicated to ACTEL FPGAs, are understandable, as our program does not make use of the fact that ACTEL cells are built of multiplexers.

6. SUMMARY

A general method to decompose incompletely specified multiple-output Boolean functions has been presented.

TABLE 8
Results for Different Types of Cells

Name	two input one output cells		three input one output cells		four input one output cells		five input one output cells	
	our	Woo	our	Woo	our	Woo	our	Woo
rd84	28	107	13	49	10	43	8	42
rd73	22	—	9	—	7	—	6	—
rd53	13	—	6	—	5	—	5	—
z4	15	—	6	—	6	—	5	—
5xp1	44	53	25	35	18	24	13	27
sao2	56	69	38	44	31	34	22	41
f51m	45	—	24	—	18	—	13	—
adr4	19	—	9	—	7	—	6	—
sqn	90	—	40	—	20	—	9	—
con1	17	—	9	—	9	—	4	—
9sym	27	97	11	62	9	51	7	52

The general method is based on supplementing the better known decomposition strategy called serial decomposition with parallel decomposition and on an original representation of Boolean functions by a set of r-partitions and the corresponding calculus.

Based on the presented decomposition algorithms, a prototype version of a logic synthesis system has been developed. Our results demonstrate that logic synthesis based on functional decomposition is usually more efficient than the conventional approach in which technology-independent minimization is followed by technology mapping. This observation, first formulated in our earlier paper [17] and supported by the results of other studies [8], [27], is especially true for designs involving PLD or FPGA components.

The presented decomposition procedures are very general. The user is given the possibility of working on minterms or cubes and can arbitrarily specify the maximum number of inputs and maximum number of outputs for all the components of a function to be decomposed. Alternatively, the designer is given the option of interactively controlling the decomposition process by selecting, for each iteration of the decomposition, the component of the partially decomposed function to be dealt with and the type of decomposition (parallel or serial). This way, the decomposition procedure, allows the designer to examine several alternative solutions. In particular, it makes it possible to compare different implementation styles, e.g. among various FPGA structures, and select the one which is most suitable for a given project. As the conceptual layer of the method and its core are general, it is possible to apply the method to decompose multiple-valued functions also. This has lead to the development of a PLA-based Synthesis System which finds its application specially in designs using PLAs with decoders [19]. The presented algorithms can therefore form a basis for development of a general decomposition-based synthesis tool which would accept a set of design constraints and decompose a given function so that to meet those constraints.

TABLE 9
Results for Cells with 5 Inputs and 2 Outputs

Name	Our method	Sasao [24]	mis-pga (new)	HYDRA	ASYL	Chortle	Xnfmap
rd84	5	11	10	27	17	35	50
rd73	4	6	6	13	30	16	24
misex1	7	22	11	8	13	11	12
z4	3	5	5	4	4	3	3
5xpl	8	18	18	21	24	24	20
sao2	11	—	28	36	36	27	37
9sym	4	7	7	33	8	51	54
f52m	6	—	—	—	—	—	—
adr4	4	—	—	—	—	—	—
sqn	6	—	—	—	—	—	—
con1	4	—	—	—	—	—	—

TABLE 10
Results for Cells with 2 Inputs and 1 Output

Name	Our Method	ASYL [25]	MIS-PGA [10]	XBOOLE [3]
rd84	28	30	36	–
z4	15	9	18	–
rd53	13	10	–	28
rd73	22	20	25	42
9sym	27	–	17	105
sao2	56	45	49	109
5xp1	44	37	51	67
f51m	45	35	39	–
bw	75	54	60	129
adr4	19	–	–	–
sqn	90	–	–	–
root	106	–	–	–
add	7	–	–	–
con1	17	–	–	–

References

[1] Algotronix Ltd., "The Configurable Logic Data Book", Edinburg, 1990.

[2] R.L. Ashenhurst, "The Decomposition of Switching Functions", Proc. of International Symp. Theory of Switching Functions 1959.

[3] D. Bochman, F. Dresig, B. Steinbach, "A New Decomposition Method for Multilevel Circuit Design", Euro-DAC'91.

[4] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A.R. Wang, "MIS: Multiple-Level Logic Minimization System", IEEE Trans. on CAD, vol. CAD-6, Nov. 1987.

[5] Brown S.D., Francis R.J., Rose J., Vranesic Z.G., Field-Programmable Gate Arrays. Kluwer Academic Publisher, Dordrecht 1992.

[6] H.A. Curtis, A New Approach to the Design of Switching Circuits, D. Van Nostrand Company, 1962.

[7] S. Devadas, A.R. Wang, A.R. Newton, A. Sangiovanni-Vincentelli, "Boolean Decomposition in Multi-Level Logic Optimization", Proc. IEEE International Conf. on Computer-Aided Design, 1988.

[8] F. Dresig, Ph. Lanches, O. Rettig, U.G. Baitinger, "Functional Decomposition for Universal Logic Cells using Substitution", Euro-DAC'92.

[9] S. Ercolani, G. De Micheli, "Technology Mapping for Electrically Programmable Gate Array", 28th ACM/IEEE- DAC'91.

[10] D. Filo, J.C. Yang, F. Mailhot, and G.D. Micheli, "Technology Mapping for a Two-Output RAM-based Field-Programmable Gate Array", Euro-DAC'91.

[11] R. Francis, J. Rose, Z. Vranesic, "Chortle-crf: Fast Technology Mapping Look-up Table-Based FPGAs", Proc. 28-th ACM/IEEE Design Automation Conf., 1991.

[12] J. Hartmanis, R.E. Stearns, Algebraic Structure Theory of Sequential Machines, Prentice-Hall, Inc., 1966.

[13] S.L. Hurst, D.M. Miller, J.C. Muzio, Spectral Techniques in Digital Logic, Academic Press, 1985.

[14] L. Jozwiak, F. Volf, "An Efficient Method for Decomposition of Multiple-Output Boolean Functions and Assigned Sequential Machines", Proc. European Conference on Design Automation, 1992.

[15] T. Łuba, M. Markowski, B. Zbierzchowski, "Logic Decomposition for Programmable Gate Arrays". Euro-ASIC'92.

[16] T. Łuba, J. Rybnik, "Rough Sets and Some Aspects in Logic Synthesis", in R. Słowiński (ed.), Intelligent Decision Support—Handbook of Application and Advances of the Rough Sets Theory, Kluwer Academic Publishers, 1992.

[17] T. Łuba, J. Kalinowski, K. Jasiński, A. Kraśniewski, "Combining Serial Decomposition with Topological Partitioning for Effective Multi-Level PLA Implementations", in P. Michel and G. Saucier (ed.), Logic and Architecture Synthesis, Elsevier Science Publishers B.V. (North-Holland), 1991.

[18] T. Łuba, H. Selvaraj, A. Kraśniewski: "A New Approach to FPGA-based Logic Synthesis". Workshop on Design Methodologies for Microelectronics and Signal Processing, 1993.

[19] T. Łuba, R. Lasocki, J. Rybnik: "An Implementation of Decomposition Algorithm and its Application in Information Systems Analysis and Logic Synthesis". International Workshop on Rough Sets and Knowledge Discovery, (RSKD-93), 1993.

[20] R. Murgai, Y. Nishizaki, N. Shenoy, R.K. Brayton, A. Sangiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays", Proc. 27th ACM/IEEE Design Automation Conf., 1990.

[21] Nam-Sung Woo, "A Study on the Structure of the Intermediate Network in an FPGA Technology Mapping", Proc. Field Programmable Logic and Applications, Oxford, 1991.

[22] Z. Pawlak, Rough Sets. Theoretical Aspects of Reasoning about Data, Kluwer Academic Publishers, 1991.

[23] P.J. Roth, R.M. Karp, "Minimization over Boolean Graphs", IBM Journal of Research and Development, vol. 6, 1962.

[24] T. Sasao, "Logic Synthesis and Optimization", Kluwer Academic Publishers, 1993.

[25] P. Sicard, M. Crastes, K. Sakouti, G. Saucier, "Automatic Synthesis of Boolean Functions on Xilinx and Actel Programmable Devices", Proc. Euro ASIC '91.

[26] XACT LCA Development System, vol. II, XILINX Inc., 1989.

[27] W. Wan, M.A. Perkowski, "A New Approach to the Decomposition of Incompletely Specified Multi-Output Function Based on Graph Coloring and Local Transformations and Its Application to FPGA Mapping", Proc. European Design Automation Conf., 1992.

Biographies

TADEUSZ ŁUBA received the M.S. degree in electronics engineering from the Warsaw University of Technology, Poland, in 1971. He received his Ph.D. and D.Sc. degrees from the same University in 1980 and 1988 respectively. Currently he is a Professor in the Department of Electronics at the Warsaw University of Technology. His research interests include switching theory, CAD tools for logic synthesis and optimization, compilers for Programmable Logic Devices, logic synthesis and testing of digital circuits and algebraic theory of automata.

HENRY SELVARAJ received the M.S. degree in electronics engineering from the Warsaw University of Technology, Poland, in 1986. From 1987 to 1990 he was a lecturer at KIT, Coimbatore, India. Currently he is working towards his Ph.D. at the Warsaw University of Technology. His research interests include CAD tools for logic synthesis and optimization, compilers for Programmable Logic Devices, logic synthesis of digital circuits and digital signal processing.