DECOMPOSITION OF MULTIPLE-VALUED FUNCTIONS

Tadeusz Łuba
Institute of Telecommunications, Warsaw University of Technology
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland

Abstract

This paper presents a generalized method for decomposition of Multiple-Valued functions. The main reason for using the described method is efficient implementation of logic circuits as well as effective representation of data in information systems. In logic synthesis, the method reduces the demand for silicon space required to implement designs. It is shown that the decomposition technique leads to additional compressing capabilities in PLA implementations. Another very promising area of application of decomposition is its effective representation of data in information systems, data bases and in other applications of information storing systems.

1. Introduction.

The notion of decomposition is central to most methods for systems analysis and design. For example, typical applications are functional decomposition, FSM decomposition, information systems decomposition, and Roth-Karp decomposition.

In general, decomposition relies on the breakdown of a complex system into smaller, relatively independent units. The motivation for using decomposition in system analysis and design is to reduce the complexity of the problem by a divide-and-conquer paradigm: A system is decomposed into a set of smaller subsystems such that each of them is easier to synthesize or analyze. A decomposed system can also be superior to a monolithic one.

Decomposition is a fundamental problem in modern logic synthesis. At first glance its goal is to break a logic circuit into a set of smaller interacting components. Such an implementation is desirable for a number of reasons. A decomposed circuit usually leads to improved performance which is reflected both in less silicon area and less signal delay.

Particular stimulus for decomposition methods came recently from the field of programmable multi-block logic devices. Such technologies are characterized by I/O or gate-limited blocks of logic into which the circuit must be mapped. In such a case implementation is impossible without decomposition. Similarly, decomposition is necessary in order to implement large circuits with the more traditional PLAs.

Although a vast majority of digital systems available today use conventional binary circuits, multiple valued logic has recently attracted great interest. Of the possible design approaches to multiple-valued logic circuits, the PLAs with decoders have received the most attention.

Multiple-valued synthesis is an extension of classical binary logic to variables which can assume more than two values. However, its application includes not only minimization of PLAs with input decoders [3], [14] but also Boolean decomposition of a PLA into several serially cascaded PLAs [2].

The strong stimulus for developing decomposition methods and tools comes also from the data compressing problems in machine learning, pattern recognition and in many other areas of AI applications. In machine learning the idea of reduction of instance space is well known, see e.g. an approach to compressing sets of examples, attributes and attribute-value tuples in a technique called a partition triple [5] or other approaches to reduction of an instance space [13].

The above techniques, in comparison with logic synthesis capabilities, suffer from lack of pure functional decomposition strategies. Recent results [11], [12] in decomposition-based methodology seem to indicate that the concept of functional decomposition should be investigated more generally and in more detail. In this paper we concentrate

0-8186-7118-1/95 $04.00 © 1995 IEEE

on generalization of the decomposition methodology to make it applicable to multiple-valued logic synthesis and to information systems as well.

2. Basics of Decomposition

Let x_i be a multiple-valued variable, and C_i = = $\{0,1,...,c_i-1\}$ be a set of values that it may assume. A generalized Multiple-Valued function with n input, m output variables is defined as a mapping:

$$F(x_1,...,x_n): C_1 \times C_2 \times ... \times C_n \longrightarrow D^m,$$

where D = {0,1,-} represents the binary value of the function (0 or 1) and "–" denotes a don't care value.

For the sake of clarity the mapping F may be viewed as pair T = (M, A), where

M – is a non-empty, finite set of minterms,

A – is a finite set of arguments i.e. input and output variables ; A = X ∪ Y, where X is the set of input variables and Y is the set of output variables, X ∩ Y = ϕ. Moreover

a – is a function which maps the argument a ∈ A into their values for every minterm v ∈ M, i.e.:

$$a: M \longrightarrow V_a,$$

where V_a is a domain (set of possible values) of argument a.

With every subset of arguments B ⊆ X, an equivalence relation IND(B), called indiscernibility relation, is defined as follows:

$$IND(B) = \{(m_1,m_2) \in M: \forall x \in B, x(m_1) = x(m_2)\}$$

Minterms satisfying relation IND(B) are indiscernible by arguments from B. Thus, the relation IND partitions M into equivalence classes M/IND(B), which form a partition P(B) on M.

Similarly, the output-consistency relation, denoted by CON, can be related to every subset B of output variables. This relation can be formally defined as follows:

Let B ⊆ Y and p,q ∈ M, minterms p,q ∈ CON(B) iff y(p) ~ y(q) for every y ∈ B, where $a_1 \sim a_2$ means that a_1 and a_2 are equal if defined.

The CON relation is not an equivalence relation on the set M, as the output-consistency classes of the set M can be conjunct. However, an unique set of Maximal Output-Consistency Classes (MOCC) of minterms exists for every given CON relation.

On the analogy of partition P(B), the collection of MOCCs of minterms creates a "rough" partition (r-partition for short) on the set M.

Conventions used in denoting r-partitions and their typical operators are the same as in the case of partitions i.e. an r-partition on a set M may be viewed as a collection of non-disjoint subsets of M, where the set union is M. Thus r-partition concepts are simple extensions of the partition algebra [6], with which reader's familiarity is assumed.

Example 1. For a function F given in Table 1, where M = = {1,...,10}, X = $\{x_1,...,x_6\}$, and Y = $\{y_1,y_2,y_3\}$, example IND and CON relations can be written as follows:

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

Table 1

a)

	x_i								
	1	2	3	4	5	6	y_1	y_2	y_3
1	0	0	0	0	0	0	1	1	0
2	0	0	1	1	0	0	1	1	0
3	1	2	2	0	1	1	0	1	1
4	0	1	1	0	0	1	0	1	1
5	0	1	0	2	0	1	1	0	1
6	1	2	2	3	2	0	0	1	1
7	1	2	2	2	0	1	1	1	0
8	0	0	1	1	0	1	1	0	1
9	0	1	0	3	2	0	0	1	0
10	2	2	2	3	2	0	0	1	0

b)

	y_1	y_2	y_3
1	1	1	0
2	1	1	0
3	0	1	–
4	0	1	1
5	1	0	1
6	0	1	1
7	–	1	0
8	1	0	1
9	0	1	0
10	0	1	0

On the other hand, if the respective output vectors are as it is shown in Table 1b, then the corresponding r-partition P_F can be written:

P_F = {{1,2,7}, {3,4,6}, {5,8}, {3,7,9,10}} or
P_F = (1,2,7 ; 3,4,6 ; 5,8 ; 3,7,9,10), in short.

Let F be a multiple-valued function representing functional dependency Y = F(X), where X is the set of multiple-valued input variables and Y is the set of binary output variables. Let X = A ∪ B, A ∩ B = ϕ and C ⊆ A. We say that there is a functional decomposition of F iff

$$F = H(A,G(B,C)) = H(A,Z) \quad (1)$$

where G and H denote functional dependencies: G(B,C) = Z and H(A,Z)= Y and Z is the set of two-valued variables. If in addition, C = ϕ, then H is called a simple disjoint decomposition of F.

With the decomposed structure, a multilevel circuit can be realized as serially connected components G and H, where the outputs of G form intermediate variables. Thus we will refer to this process as to a serial decomposition.

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

Theorem 1: Functions G and H represent a serial decomposition of function F i.e. F = H(A,G(B,C)) if there exists a partition $\Pi_G \geq P(B \cup C)$ such that

$$P(A) \cdot \Pi_G \leq P_F \quad (2)$$

where all the partitions are over the set of minterms and the number of two-valued output variables of component G is equal to $\lceil \log_2 L(\Pi_G) \rceil$, where $L(\Pi)$ denotes the number of blocks of partition Π, and $\lceil x \rceil$ denotes the smallest integer equal to or larger than x.

In the theorem, partition Π_G represents component G and the product of partitions P(A) and Π_G corresponds to H.

Given Π_G we can verify whether condition (2) is satisfied. However, the procedure for calculating Π_G is complicated and will be described in section 5.

The theorem is general in the sense that any function can be processed i.e., it can be applied to binary or multiple-valued functions, completely or incompletely specified. However, with respect to binary outputs of component G, for PLA implementations – multiple-valued inputs and binary outputs have to be assumed both for function F and decomposed components G and H.

Example 2. Let us decompose the function F of Table 1. For A = $\{x_1,x_6\}$, B = $\{x_2,x_3,x_4,x_5\}$, C = ϕ, we have

P(A) = (1,2,9 ; 3,7 ; 4,5,8 ; 6 ; 10)
P(B) = (1 ; 2,8 ; 3 ; 4 ; 5 ; 6,10 ; 7 ; 9)

Consider Π_G = (1,2,3,5,6,8,10 ; 4,7,9). It can be easily verified that since $P(A) \bullet \Pi_G \leq P_F$, function F is decomposable as $F = H(x_1,x_6,,G(x_2,x_3,x_4,x_5))$, where G is one-output function of four variables.

Table 2

x_2 x_3 x_4 x_5	g
0 0 0 0	0
0 1 1 0	0
2 2 0 1	0
1 1 0 0	1
1 0 2 0	0
2 2 3 2	0
2 2 2 0	1
1 0 3 2	1

Table 3

x_1 x_6 g	y_1 y_2 y_3
0 0 0	1 1 0
1 1 0	0 1 1
0 1 1	0 1 1
0 1 0	1 0 1
1 0 0	0 1 1
1 1 1	1 1 0
0 0 1	0 1 0
2 0 0	0 1 0

The truth tables of components G and H can be obtained from partitions P(A), Π_G, and P_F. If we assign to each block of partition Π_G a binary vector $g_1,...,g_u$ ($u = \lceil \log_2 L(\Pi_G) \rceil$), then (as $P(BUC) \leq \Pi_G$) for each input vector to G there is one and only one corresponding vector $g = (g_1,...,g_u)$. Similarly, for each block of partition $P(A) \bullet \Pi_G$ we can assign a vector q with its elements $x_{i1},...,x_{it}$, $g_1,...,g_u$ defined by the variables of set A and the intermediate variables $g_1,...,g_u$. As $\Pi_H = P(A) \cdot \Pi_G \leq P_F$, each block of Π_H corresponds to one and only one block of P_F, and consequently to one and only one vector of the output variables $y_1,...,y_m$.

In the example, u=2, therefore encoding the blocks of Π_G respectively as 0 and 1, we immediately obtain the truth table of function G; it is presented in Table 2. The truth table of function H can be derived by reencoding input vectors of F using an intermediate variable g. The truth table obtained in this way is shown in Table 3.

3. PLA Decomposition

In serial decomposition a function F can be implemented as a multi-level PLA structure, but in contrary to the existing methods [2], the circuits implementing components G and H have, in general, multiple-valued inputs and two-valued outputs.

The structure of the decomposed circuit is not limited to that of PLAs and standard decoders as in [14], however the PLAs with decoders are the most suitable devices for implementing MVL circuits. The method, therefore, is not confined to PLA synthesis and can be considered as a general function decomposition approach.

For PLA implementation, the design objective is to decompose the original function into components corresponding to a set of interconnected PLAs, so that the overall area of the resulting logic network is minimized. The silicon area taken by a PLA with two-bit decoders can be estimated as $S = (4n_v + 2n_b + m)P$, where n_v, n_b denote the number of multiple-valued and binary input variables, respectively, m is the number of output variables, and P is the number of product terms.

The truth table of F (Table 1a) can be minimized to five product terms, which in the positional cube notation can be written as:

1101 1111 1111 1111 1111 10 001
1111 1111 1111 1110 1011 11 100
1111 1101 1111 1101 1001 11 110
1011 1111 1110 1001 1111 11 011
1111 1111 1111 1111 1111 01 010

thus the estimated area is $S_F = (4 \bullet 5 + 2 \bullet 1 + 3) \bullet 5 = 125$.

When the function F is decomposed as shown in Example 2, the product terms for components G and H, represented in cube notation as well are as follows:

G: 1110 1110 1111 1001 1
1010 1111 1011 1111 1

H: 1001 11 01 100
1101 10 11 001
1010 11 01 011
1110 11 10 100
1111 01 11 010
1111 11 10 010

Therefore the PLA area estimated as $S_G + S_H$, where $S_G = 34$ and $S_H = 66$, is only 80% of the original function after the minimization procedure.

4. Decomposition of Decision Tables

The proposed functional decomposition procedure can also be effectively used for reduction of space requirements in many problems related to data representation in machine learning [5], [13].

Let F be a mapping function

$$F: c_1 \times \ldots c_n \longrightarrow D$$

representing functional dependency D = F(C), where C is the set of condition attributes and D is the set of decision attributes. Let A, B be the subsets of C such that $C = A \cup B$ and $A \cap B = \phi$.

The structure of the decomposed decision function is shown in Fig.1. The procedure of making a final decision is as follows: an intermediate decision is made on the basis of the attributes' subset B and then taking into consideration both the intermediate decision and the attributes subset A, the goal decision is made, which is equal to the corresponding value of the function F. A simple counterpart of Theorem 1 for a case of decision tables was formulated in [7].

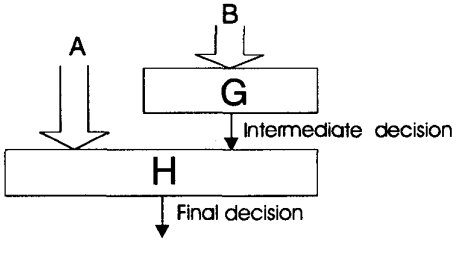

Fig.1. Two-stage realization of a decision table

The decomposition gain comes from the fact that in some cases the size of table representation of the function generated using decomposition can be much smaller than the representation of a single unified function [7].

5. The Compatibility Relation

As it is seen from Theorem 1, the main task of decomposition is to verify if a subset of input variables for function G which, when applied as a subfunction to function H will generate final function F , i.e. to find P(B), such that there exists $\Pi_G \geq P(B)$ that satisfies condition (2) in Theorem 1.

A relation of compatibility of partition blocks will be used to verify whether or not partition P(B) is suitable for serial decomposition.

Two blocks $B_i, B_j \in P(B)$ are compatible if and only if partition Π'_G obtained from partition P(B) by merging blocks B_i and B_j into a single block B'_{ij} satisfies condition (2) in Theorem 1, i.e., if and only if

$$P(A) \bullet \Pi'_G \leq P_F \qquad (3)$$

A subset of n partition blocks, $\mathcal{B} = \{B_{i_1}, B_{i_2}, \ldots, B_{i_n}\}$, where $B_{i_j} \in P(B)$, is a class of compatible blocks for partition P(B) if and only if all blocks in $\mathcal{B}$ are pairwise compatible.

A compatible class is called Maximal Compatible Class (MCC) if it is not a subset of any other compatible class.

The set $M = \{MCC_1, \ldots, MCC_r\}$ of all Maximal Compatible Classes can be formed from the set of all compatible pairs (B_i, B_j), which in this case can be interpreted as arcs of a graph G = (**B**, COM), where elements of **B** represent its vertices, COM represents the compatibility relation and where two vertices are connected by an arc iff B_i and B_j are compatible i.e. $(B_i, B_j) \in$ COM. In such a formulation finding the Maximal Compatible Classes for P(B) blocks is equivalent to finding the maximum cliques in a graph G.

The calculation process then is simply to select a subset of MCCs that cover the set of all blocks of P_G. The partition Π_G satisfying the inequality (2) and having the minimum number of blocks can be found by solving the following covering problem for the set **B** of blocks of partition P(B):

$$\bigcup MCC_j = \mathbf{B} \text{ and } k = \min.$$

In other words we try to find a subset of MCCs i.e. $MIN(M) = \{MCC_{i_1}, MCC_{i_2}, \ldots, MCC_{i_k}\}$ such that their union results in the set **B**. The minimal k ensures the minimum number of blocks of partition Π_G.

Thus blocks of Π_G can be created from MCCs by eliminating the repeated elements of **B** in the minimal cover. The final Π_G is a result of the union of minterms forming a set of blocks included in any one block of Π_G.

Example 3. For the function F of Example 1, and the same sets A, B as in Example 2, let us denote the blocks of P(B) as $B_1,...,B_8$ respectively, i.e. P(B) = $\{B_1,...,B_8\}$, where $B_1 = \{1\}$, $B_2 = \{2,8\}$, $B_3 = \{3\}$, $B_4 = \{4\}$, $B_5 = \{5\}$, $B_6 = \{6,10\}$, $B_7 = \{7\}$, $B_8 = \{9\}$. By verifying the condition (3) for each pair (B_i,B_j), we obtain the following relation: COM = $\{(B_1,B_2)$, (B_1,B_3), (B_1,B_4), (B_1,B_5), (B_1,B_6), (B_1,B_7), (B_2,B_3), (B_2,B_5), (B_2,B_6), (B_2,B_7), (B_3,B_4), (B_3,B_5), (B_3,B_6), (B_3,B_8), (B_4,B_6), (B_4,B_7), (B_4,B_8), (B_5,B_6), (B_5,B_7), (B_5,B_8), (B_6,B_7), (B_6,B_8), $(B_7,B_8)\}$.

Then applying any algorithm that finds the maximum cliques, the following Maximal Compatible Classes are generated:

MCC1 = $\{B_5,B_6,B_7,B_8\}$, MCC5 = $\{B_1,B_2,B_5,B_6,B_7\}$,
MCC2 = $\{B_4,B_6,B_7,B_8\}$, MCC6 = $\{B_1,B_4,B_6,B_7\}$,
MCC3 = $\{B_3,B_5,B_6,B_8\}$, MCC7 = $\{B_1,B_2,B_3,B_5,B_6\}$,
MCC4 = $\{B_3,B_4,B_6,B_8\}$, MCC8 = $\{B_1,B_3,B_4,B_6\}$.

One of the minimal covers is accomplished by MCC2 ∪ MCC7, therefore, the blocks of Π_G can be built of $\{B_1,B_2,B_3,B_5,B_6\}$ and $\{B_4,B_7,B_8\}$. Hence Π_G = = (1,2,3,5,6,8,10 ; 4,7,9).

6. The r-admissibility Test

Direct application of Theorem 1 to find functions G and H would make the problem computationally intractable. To overcome this difficulty, we present conditions that allow us to check if, for a given set of input variables A ⊂ X, function F is decomposable so that component H has a given number of input variables, and variables in A directly feed H. These conditions are based on the concept of r-admissibility of a set of partitions.

Let P_i be a partition on M induced by some input variable x_i. The set of partitions $\{P_1,...,P_k\}$ is called r-admissible with respect to partition P_F if there exists a set $\{P_{k+1},...,P_r\}$ of two-block partitions, such that

$$P_1 \bullet ... \bullet P_k \bullet P_{k+1} \bullet ... \bullet P_r \le P_F,$$

and there exists no set of r − k − 1 two-block partitions which meets this requirement.

The r-admissibility has the following interpretation. If a set of partitions $\{P_1,...,P_k\}$ is r-admissible, then there exists a serial decomposition of F in which component H has r inputs: k primary inputs corresponding to input variables which induce $\{P_1,...,P_k\}$ and r−k inputs being outputs of G. Thus, to find a decomposition of F in which component H has r inputs, we must find a set of input variables which induces an r-admissible set of input partitions. To formulate a simple condition that can be used to check whether or not a given set of partitions is r-admissible, we introduce the concept of a quotient partition.

Let τ be a partition and σ an r-partition, such that τ ≥ σ. In a quotient partition of τ over σ, denoted τ|σ, each block of τ is divided into a minimum number of elements being (not necessarily disjoint) blocks of σ.

The following theorem can be applied to check whether or not a set of input partitions is r-admissible.

Theorem 2: For partitions σ and τ, such that σ ≤ τ, let τ | σ denote the quotient partition and η(τ | σ) the number of elements in the largest block of τ | σ. Let e(τ | σ) denote the smallest integer equal to or larger than $\log_2\eta(\tau \mid \sigma)$, i.e., $e(\tau \mid \sigma) = \lceil \log_2\eta(\tau \mid \sigma) \rceil$. Let Π be the product of partitions $P_1,...,P_k$ and $\Pi_F = \Pi \bullet P_F$. Then $\{P_1,...,P_k\}$ is r-admissible in relation to P_F, with $r = k + e(\Pi \mid \Pi_F)$.

Example 4. The following set of partitions on M = {1,...,15} represents a certain function F of three two-valued variables, x_1, x_2, x_4, and one four-valued variable, x_3.

P_1 = (1,2,3,4,5,6,7 ; 8,9,10,11,12,13,14,15)
P_2 = (1,2,3,13,14,15 ; 4,5,6,7,8,9,10,11,12)
P_3 = (1,7,8,13 ; 2,3,9,14,15 ; 4,5,10 ; 6,11,12)
P_4 = (1,3,4,6,7,8,9,10,12,15 ; 2,5,11,13,14)
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)

By examining the admissibility of $\{P_1,P_3\}$ we obtain

$P_1 \bullet P_3 \mid P_1 \bullet P_3 \bullet P_F$ = ((1)(7) ; (8,13) ; (2)(3) ;
(9,14)(14,15) ; (4)(5) ; (10) ; (6) ; (11)(12)).

Hence, $r = 2 + \lceil \log_2 2 \rceil = 3$ i.e. $\{P_1,P_3\}$ is 3-admissible. Similarly, we can show that $\{P_3,P_4\}$ is 3-admissible; $r(P_3,P_4) = 3$, for short.

Therefore, $F = H_1(x_1,x_3,G_1(x_2,x_4,C_1))$ with $C_1 \subseteq \{x_1,x_3\}$ or $F = H_2(x_3,x_4,G_2(x_1,x_2,C_2))$ with $C_2 \subseteq \{x_3,x_4\}$, where each C can contain one, two or none variables and both G_1 and G_2 are single-output functions.

However, if we calculate the admissibility of $\{P_2,P_3\}$ we will conclude that decomposition of F with the set $A = \{x_2,x_3\}$, where G is single-output function, does not exist. This is because $r(P_2,P_3) = 4$, and the only possibility for decomposition of F is with the two-output function G.

In general, the analysis of r-admissibility makes it possible to select some interesting subsets of variables for which the best disjoint decomposition can exist, however to be sure of this fact, the sufficient condition of the decomposition existence (Theorem 1) should be next verified.

7. Experimental Results and Conclusions

The decomposition method presented in this paper has been implemented within experimental program called Functional Decomposer (FD).

The goal of decomposition is to replace the initial truth or decision table with two other tables that occupy less silicon area or memory space and allow faster processing.

For testing our Functional Decomposer we used a set of Logic Benchmarks in the form of truth tables. The results are given in Table 4, where the entries in columns denote: OA – original area of the circuit, AAD – area after decomposition and PR – profit rate.

Table 4

NAME	OA	AAD	PR
z9sym	1045	475	54%
rd84	1620	660	59%
life	798	369	54%
rd53	234	180	23%
test4	4830	3524	27%
z4	720	556	23%
adr4	3360	1585	53%

Other applications of the decomposition procedure in logic synthesis can be found in [8], where the advantages of the functional decomposition were demonstrated for PLD-based logic synthesis. The described method has also been implemented in a prototype decomposition program dedicated to FPGA-based logic synthesis [9].

We hope that the proposed methodology is general in the sense that many kinds of information storing systems and all kinds of Boolean functions can be processed. The conceptual layer of the method and its core are very general. They can be applied to many decomposition problems in knowledge representation, data base and logic systems and especially to the problems where the nominal data cannot be reduced to the quantitative data without substantial loss of information. In the case of logic synthesis the presented procedure is universal, i.e., it can be applied to completely or incompletely specified, binary or multiple-valued Boolean functions and any decomposition topology making it suitable for various implementation styles including PLAs and FPGAs. The input and output routines and the analysis of the problem are only to be tuned to a particular problem. This means that the presented methodology can form a basis for the development of a general decomposition-based synthesis tool which would accept a set of design constraints and decompose a given system so as to meet those constraints.

References

[1] **Brayton R.K., Hachtel G.D., McMullen C.T., Sangiovanni-Vincentelli A. (1984).** Logic Minimization Algorithms for VLSI Synthesis. *Kluwer Academic Publ.*

[2] **Ciesielski M., and Yang S. (1992).** PLADE: A Two Stage PLA Decomposition. IEEE *Trans. on CAD.* vol. 11, pp. 943-954.

[3] **Ciesielski M., Yang S., and Perkowski M. (1989).** Multiple-Valued Minimization Based on Graph Coloring. *Proc. IEEE Int. Conf. Computer Design,* pp. 262-265.

[4] **Grzymała-Busse J.W. (1992).** LERS - A System to Learning from Examples Based on Rough Sets. In Intelligent Decision Support - Handbook of Application and Advances of the Rough Sets Theory, R.Słowiński (ed), *Kluwer Academic Publishers.*

[5] **Grzymała-Busse, J. W. (1990).** On the Reduction of Instance Space in Learning from Examples. In *Methodologies for Intelligent Systems, 5, Z.W.Ras, M.Zemankowa and M.L.Emrich, (eds). Elsevier Science Publ.,* pp. 388-395.

[6] **Hartmanis, J. and Stearns, R. E. (1966).** Algebraic Structure Theory of Sequential Machines. *Prentice-Hall.*

[7] **Łuba T., Lasocki R., Rybnik J. (1994).** An Implementation of Decomposition Algorithm and its Application in Information Systems Analysis and Logic Synthesis. In Rough Sets, Fuzzy Sets and Knowledge Discovery, W. Ziarko (Ed.). Workshops in Computing Series. *Springer Verlag,* pp. 458-465.

[8] **Łuba T., Kalinowski J., Jasiński K. (1991).** PLATO: A CAD Tool for Logic Synthesis Based on Decomposition. *Proc. of European Conference on Design Automation,* pp. 65-69.

[9] **Łuba T., Selvaraj H., Kraśniewski A. (1993).** A New Approach to FPGA-based Logic Synthesis. *Workshop on Design Methodologies for Microelectronics and Signal Processing, Gliwice - Cracow.*

[10] **Pawlak Z. (1991).** Rough Sets. Theoretical Aspects of Reasoning about Data. *Kluwer Academic Publishers.*

[11] **Sasao T. (1993).** Logic Synthesis and Optimization. *Kluwer Academic Publishers.*

[12] **Wan W., Perkowski M.A. (1992).** 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.,* pp.230-235.

[13] **Kohavi R. (1994).** A Third Dimension to Rough sets. *Proc. of The Third International Workshop on Rough Sets and Soft Computing,* pp.244-251, San Jose.

[14] **Sasao T. (1984).** Input Variable Assignment and Output Phase Optimization of PLAs. *IEEE Trans Comput.* Vol. C-33. pp. 879-894.