Appl. Math. and Comp. Sci., 1994, vol.4, No.1, 125–138

DECOMPOSITION OF MULTIPLE–VALUED BOOLEAN FUNCTIONS

TADEUSZ ŁUBA*, ROBERT LASOCKI*

In this paper we consider the problem of decomposition of Multiple–Valued Boolean functions and its implementation using Programmable Logic Arrays (PLAs) as examples. The main reason behind using the described method is its economic realization of logic circuits in PLA implementations and in all the other applications of multiple–valued logic systems. The procedure is very general and suitable for various implementation styles, including standard PLAs, PLDs and PLAs with two–bit decoders.

1. Introduction

Multiple–valued logic has been the subject of considerable study during the past decade (Ciesielski and Yang, 1992; Devadas *et al.*, 1988; Muzio and Wesselkamper, 1986). In particular, multiple–valued logic (MVL) has gained more attention due to its application in optimal PLA design.

The PLA optimization goals are to minimize the area occupied by the PLA and to minimize the delay through the PLA. A complete strategy for the design of a PLA macrocell involves: (1) logic optimization of the PLA equations including input variables assignment and output phase assignment (Sasao, 1984; 1988), (2) optimization of the PLA layout using simple folding or partitioning techniques; and (3) generation of the mask geometries implementing the PLA.

Devadas *et al.* have proposed a Boolean decomposition of a PLA into two cascaded PLAs (Devadas *et al.*, 1988). This procedure is conceptually similar to multiple–valued symbolic minimization, rather than to the classical decomposition. Moreover, the method is confined to PLA synthesis and cannot be considered as a general functional decomposition approach applicable to different implementations. However, we can find several similarities between MVL systems and binary logic with respect to functional decomposition concept.

Recently published papers try to generalize old decomposition procedures to make them applicable to more complex cases (Ciesielski and Yang, 1992). This approach has proven to be quite successful; in many cases, the synthesized circuits have been less complex than the circuits designed using a conventional approach based on logic minimization. Following this trend, we propose an original method for decomposition which produces even better results.

* Warsaw University of Technology, Institute of Telecommunications, ul. Nowowiejska 15/19, 00–665 Warszawa, Poland

A distinguishing feature of our method is an original calculus based on the representation of a function by a family of partitions over the set of minterms. Our decomposition procedure is universal, i.e. it can be applied to completely or incompletely specified, binary or multiple–valued Boolean functions. Thus, it favorably compares with the earlier PLA–based decomposition methods, limited to two–valued Boolean 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 algorithms are given. Section 4 shows how the proposed decomposition procedure is used for synthesis of PLAs. The presented results of the benchmark design experiments demonstrate that, in several cases, our method produces circuits of significantly reduced complexity compared to the solutions reported earlier.

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 Boolean function* with n input, m output variables is defined as a mapping:

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

where $D = \{0,1,-\}$ represents the binary value of the function (0 or 1)[1]. The value — (don't care) at one of the outputs means that the value is unspecified, and a value of 0 or 1 will be accepted to realize this part of the function.

Every element of the domain $C_1 \times C_2 \times ... \times C_n$ is called a *minterm*. A listing of minterms with the value of the function is called a *truth table*. Truth tables do not include minterms with the function value not specified for all outputs. Set of minterms for which the function value is unspecified is called a *DC–set* (Don't Care–set). Functions with non–empty DC–set are called *partially defined*.

For the sake of clarity, truth tables may be viewed as a set $T = (M,\ A,\ X,\ Y)$, where M is a non–empty, finite set of objects, A is a finite set of variables (arguments); $A = X \cup Y$, where X is a set of input variables and Y is a set of output variables, $X \cap Y = \emptyset$. Moreover, a is a function assigning a value of variable for every object m, i.e.

$$a : M \rightarrow V_a$$

where V_a is a domain (set of possible values) of variable a. According to physical implementations we assume that the set of values of output variables $V_y = \{0,1,-\}$.

In general, any pair of minterms in a specification table of Multiple–Valued Boolean (MVB) function may have identical values for some number of input variables. A convenient way to reflect such similarities can be introduced with the help of the

[1] The assumption of binary values of outputs is implied from the structure of commonly used PLAs, i.e. multi–valued input, two–valued output PLAs (Sasao, 1984; 1988; Rudell and Sangiovanni–Vincentelli, 1987).

so called indiscernibility relation (Pawlak, 1991). This relation, denoted by IND , is associated with any subset of input variables as follows:

let $B \subseteq X, \quad m_1, m_2 \in M,$
$(m_1, m_2) \in IND(B)$ iff $x(m_1) = x(m_2)$ for all $x \in B$

This means that $(m_1, m_2) \in IND(B)$, if the values of the arguments belonging to B are identical for both m_1 and m_2. Minterms m_1 and m_2 are said to be indiscernible by arguments from B. The indiscernibility relation is an equivalence relation on M and

$$IND(B) = \bigcap_{x \in B} IND(x) \tag{1}$$

Thus, the relation IND partitions M into equivalence classes $M/IND(B)$. Such partitions are of primary importance in logic synthesis (Hartmanis and Stearns, 1966; Luba *et al.*, 1991). To simplify, we shall denote partition $M/IND(B)$ by $P(B)$ and call such a partition an input partition generated by set B. Then, the formula equivalent to those of (1) may be written as

$$P(B) = \prod_{x \in B} P(x) \tag{2}$$

where $\prod$ denotes the product of partitions.

Two output vectors, y_1 and y_2, are said to be consistent if their corresponding entries are the same whenever they are both specified, i.e.

$$\forall i \in \{n+1, ..., n+m\}, \ (y_{1_i} = y_{2_i}) \vee (y_{1_i} = *) \vee (y_{2_i} = *)$$

The consistency relation on output vectors is denoted as $y_1 \sim y_2$.

In general, any pair of minterms in a logic specification table may have consistent output values for some number of output variables. Thus, the relation called *output–consistency relation* and denoted as CON, can be associated with any subset B of the output variables. The output–consistency relation is defined as follows:

let $B \subseteq Y, \ p, q \in M,$
$p, q \in CON(B)$ iff $y(p) \sim y(q)$ for every $y \in B$

where $a_1 \sim a_2$ if a_1, a_2 are the same whenever both are specified.

A set of minterms constitutes a consistent class, if every pair of minterms in the set is consistent. Those classes that are not subsets of any other output–consistent class are called *Maximal Consistent Classes* (MCCs).

Clearly, the consistency relation is not an equivalence relation on M. Hence, it "partitions" M into non–disjoint subsets; but for a given CON relation there is a unique collection of maximal output–consistent classes of minterms. Therefore, we can use the same notation for output consistency subset, i.e. $P_F(B)$, where index F is intended to distinguish input (IND) and output (CON) relation. When $B = Y$, then we can denote the CON relation simply as P_F. Because of non–disjointness of blocks of P_F, relation CON is called a *rough–partition* (*r–partition*).

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 partition algebra (Hartmanis and Stearns, 1966), with which Reader's familiarity is assumed.

Especially the relation *less than or equal to* holds between two r–partitions Π_1 and $\Pi_2\,(\Pi_1 \leq \Pi_2)$ iff for every block of Π_1, in short denoted by $B_i(\Pi_1)$, there exists a $B_j(\Pi_2)$ such that $B_i(\Pi_1) \subseteq B_j(\Pi_2)$.

If Π_1 and Π_2 are partitions, this definition is reduced to the conventional ordering relation between two partitions.

This points out the main difference between completely and incompletely specified Boolean functions. While the equivalence classes of partitions in completely specified functions consist of disjoint subsets, the subsets of consistent minterms for partially specified functions may be overlapping. This is the reason for generalizing the typical partition description.

To present a Boolean function F, i.e. a functional dependence between outputs Y and inputs X, usually described by formula $Y = F(X)$, the table specification should be consistent.

A logic specification table is *consistent* iff for every pair of row vectors $\boldsymbol{r}_1 = (\boldsymbol{x}_1, \boldsymbol{y}_1)$, $\boldsymbol{r}_2 = (\boldsymbol{x}_2, \boldsymbol{y}_2)$, $\boldsymbol{x}_1 = \boldsymbol{x}_2$ implies $\boldsymbol{y}_1 \sim \boldsymbol{y}_2$ (i.e. for every $m_1, m_2, m_1 = m_2$ implies $F(m_1) \sim F(m_2)$).

Example 1. Consider partially defined, multiple–valued Boolean function F shown in Table 1.

In the example:

$$M = \{1, ..., 10\}, \quad X = \{x_1, ..., x_6\}, \quad Y = \{y1, y2\}$$

and $V_{x_1} = \{0,1,2\}$, $V_{x_4} = \{0,1,2,3\}$, $V_{x_6} = \{0,1,2\}$. The IND relations for $B_1 = \{x_1\}$ and $B_2 = \{x_2, x_3\}$ are as follows:

$$P(B_1) = \{\{1,2,4,5,8,9\},\ \{3,6,7\},\ \{10\}\}$$

$$P(B_2) = \{\{1\},\ \{2,8\},\ \{3,6,7,10\},\ \{4\},\ \{5,9\}\}$$

Proceeding in the same way for the output–consistency relation CON , we obtain the following r–partitions:

$$P_F(y_1) = \{\{1,2,3,4,6,7,9,10\},\ \{5,8\}\}$$

$$P_F(y_2) = \{\{1,2,5,7,8\},\ \{3,4,5,6,8,9,10\}\}$$

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

[2] For the sake of simplicity we separate the blocks of partitions by semicolons only, i.e. curly brackets are omitted.

Tab. 1. Function F from Example 1.

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

3. Functional Decomposition of Multiple–Valued Functions

3.1. Basic Theorem

Let F be a multiple–valued function representing functional dependence $Y = F(X)$, where X is a set of multiple–valued input variables and Y is a set of binary output variables. Let $X = A \cup B$, $A \cap B = \emptyset$ and $C \subseteq A$.

We say that there is a *functional decomposition* of F iff

$$F = H(A, G(B, C)) = H(A, Z) \tag{3}$$

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

In other words 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 binary outputs of G. The outputs of the function H are consistent with those of function F. So the function F can be implemented as a multi–level PLA structure, but in contrary to the method proposed by Devedas (1988) and Ciesielski and Yang (1992), PLAs implementing components G and H are, in general, PLAs with two–bit decoders (i.e. multiple–valued input, two–valued output PLAs). Moreover, as the decomposition process can be applied iteratively, the first level may contain an arbitrary number of PLAs (components G) and the second level contains a single PLA.

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*

$$P(A) \bullet \Pi_G \geq P_F \tag{4}$$

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.

Proof. For each block H_i of the partition $P(B\cup C)$, there is a corresponding vector $v_i = (a_j)|_{j:x_j\in B\cup C}$ where

$$a_j = \begin{cases} 0, & \text{if } H_i \subseteq P^0(x_j) \\ 1, & \text{if } H_i \subseteq P^1(x_j) \end{cases}$$

(P^0, P^1 denote the first and the second block of P, respectively). If we assign to each block of partition Π_G a binary vector $g_1, ..., g_l$ ($l = \lceil\log_2 L(\Pi_G)\rceil$), then (as $P_G \leq \Pi_G$) for each vector v_i there is one and only one corresponding vector $g = (g_1, ..., g_l)$. Therefore, on the set of vectors $V = \{v_i\}$, an l–output function G is specified. Similarly, for each block of partition $P(A)\bullet\Pi_G$ we can assign a vector q with its positions $x_{i_1}, ..., x_{i_t}, g_1, ..., g_l$ defined by the variables of set A and the auxiliary variables $g_1, ..., g_l$. As $\Pi_H = P(A)\bullet\Pi_G \leq P_F$, 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 $y_1, ..., y_n$. This is ensured by the fact that the minterms in consistent classes of P_F are consistent. The above–mentioned mapping of blocks of Π_H into blocks of P_F is a one–to–one mapping because for each block of Π_H (as $\Pi_H \leq P_F$) there exists one and only one block of P_F. It is evident that partition Π_G represents component G, and the product of partitions $P(A)$ and Π_G corresponds to H. The truth tables of the resulting components can be easily obtained from these partitions.

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

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

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

Consider

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

It can be easily verified that since $P(A)\bullet\Pi_G \leq P_F$, function F is decomposable as $F = H(x_1, x_2, x_3, G(x_4, x_5, x_6))$, where G is one–output function of three variables.

The truth tables of components G and H can be obtained from partitions $P(A)$, Π_G, and P_F. 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.

3.2. 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 $A \subseteq X$, 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.

Tab. 2. Function G from Example 2.

x_4	x_5	x_6	g
1	0	0	0
1	2	1	0
2	0	1	1
3	3	0	0
0	0	0	0
0	0	1	1

Tab. 3. Function H from Example 2.

x_1	x_2	x_3	g	y_1	y_2
1	1	0	0	1	0
1	1	0	1	0	0
2	2	1	0	0	1
2	2	1	1	1	0
1	0	0	0	0	1
1	0	0	1	0	0
0	2	1	0	0	1

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

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

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 be an r–partition, such that $\tau \geq \sigma$. In a quotient partition of τ over σ, denoted $\tau|\sigma$, each block of τ is divided into a minimum number of elements being (not necessarily disjoint) blocks of σ.

For example, if

$$\sigma = (1;\ 2,6;\ 3,6;\ 5,7;\ 4,5), \qquad \tau = (1,2,3,6;\ 4,5,7)$$

then, we have quotient partition $\tau|\sigma$:

$$\tau|\sigma = ((1)(2,6)(3,6);\ (5,7)(4,5))$$

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 $\sigma \leq \tau$, let $\tau|\sigma$ denote the quotient partition and $\eta(\tau|\sigma)$ the number of elements in the largest block of $\tau|\sigma$. Let $e(\tau|\sigma)$ denote the smallest integer equal to or larger than $\log_2 \eta(\tau|\sigma)$, i.e. $e(\tau|\sigma) = \lceil \log_2 \eta(\tau|\sigma) \rceil$. 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 $r = k + e(\Pi|\Pi_F)$.* ∎

Proof. Let us modify $\Pi|\Pi \bullet P_F$ so that the elements in each block of the resulting quotient partition are disjoint. By the definition of a quotient partition, each element of every block of the modified $\Pi|\Pi \bullet P_F$ is included in one or the other block of P_F. Let us construct a partition Π' in such a way that, for each block of the modified $\Pi|\Pi \bullet P_F$, its elements are placed in different blocks of Π'. It can easily be seen that partition Π' with $\eta(\Pi|\Pi \bullet P_F)$ blocks can be constructed. For such a partition Π', we have $\Pi \bullet \Pi' \leq P_F$. As each q–block partition can be represented as a product of $\lceil \log_2 q \rceil$ two–block partitions (Hartmanis and Stearns, 1966), we obtain

$$\Pi \bullet P'_1 \bullet ... \bullet P'_s \leq P_F$$

where $P'_1, ..., P'_s, s = \lceil \log_2 \eta(\Pi|\Pi \bullet P_F) \rceil$, are two–block partitions, which completes the proof.

Example 3. The following set of partitions on $M = \{1, ..., 15\}$ represents the function F of three two–valued variables, x_1, x_2, x_4, and one four–valued variable, x_3, as shown in Table 4.

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

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

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

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

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

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

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

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

Hence, $r = 1 + \lceil \log_2 5 \rceil = 4$, i.e. $\{P_1\}$ is 4–admissible.

Tab. 4. Function F from Example 4.

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

Also, as

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

$\{P_1, P_3\}$ is 4–admissible. Similarly, we can show that $\{P_3, P_4\}$ is 4–admissible. Therefore, $F = H(x_1, x_3, G_1(x_2, x_4, C_1))$ with $C_1 \subset \{x_1, x_3\}$ or $F = H(x_3, x_4, G_2(x_1, x_2, C_2))$ with $C_2 \subset \{x_3, x_4\}$, where both G_1 and G_2 are single–output functions.

3.3. The Compatibility Relation

Using the r–admissibility test, we can find a suitable set of input variables for component H. To find the corresponding set of inputs for component G, we have to find $P(B \cup C)$, such that there exists $\Pi_G \geq P(B \cup C)$ that satisfies condition (2) in Theorem 1. To solve this problem, consider a subset of primary inputs, $B \cup C$, and the q–block partition $P(B \cup C) = (B_1, B_2, ..., B_q)$ generated by this subset.

A relation of compatibility of partition blocks is used to verify whether or not partition $P(B \cup C)$ is suitable for the decomposition.

Two blocks $B_i, B_j \in P(B \cup C)$ are *compatible* iff partition $P_{ij}(B \cup C)$ obtained from partition $P(B \cup C)$ by merging blocks B_i and B_j into a single block satisfies condition (4) in Theorem 1, i.e. iff

$$P(A) \bullet P_{ij}(B \cup C) \leq P_F \qquad (5)$$

A subset of partition blocks in a $P(B\cup C)$ is a *compatible class* iff all blocks in this subset are pairwise compatible. A compatible class is called a *Maximal Compatible Class* (MCC) iff it is not included in any other compatible class.

To obtain Π_G, we first find all MCCs and then find a minimum cover of the set of all blocks of $P(B\cup C)$ by MCCs. Partition Π_G is formed by merging blocks in each MCC in this minimal cover and making the resulting blocks disjoint.

Partition Π_G represents function G corresponding to the assumed set C (set B is obtained from the admissibility test). In particular, the number of blocks in Π_G, $|\Pi_G|$, determines the number of outputs of G, m_G.

For $P(B\cup C)$ to be suitable for the decomposition we must have m_G equal to the number of outputs of G obtained from the admissibility test, i.e. $m_G = r-k$.

If this condition is not satisfied, then $P(B\cup C)$ is not suitable for the decomposition and another C, possibly including more variables, must be tried.

Once an appropriate set C and the corresponding partition Π_G are found, the truth table description 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 $A=\{x_3\}$, $B=\{x_1,x_2,x_4\}$ and $C=\emptyset$. 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

$$P(A)\bullet P_{12}(B\cup C) \le P_F$$

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

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

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

$$\mathrm{MCC1} = \{B_4, B_6, B_7, B_8\}$$

$$\mathrm{MCC2} = \{B_1, B_4, B_6, B_8\}$$

$$\mathrm{MCC3} = \{B_2, B_4, B_6, B_7\}$$

$$\mathrm{MCC4} = \{B_3, B_7, B_8\}$$

$$\mathrm{MCC5} = \{B_2, B_3, B_7\}$$

$$\mathrm{MCC6} = \{B_5, B_7\}$$

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

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

We can easily see that Π_G corresponds to a 2–output function G. As the admissibility of $A = \{x_3, x_4\}$, $r(A) = 4$, the number of outputs of G required by the admissibility test is 2 and the above Π_G satisfies this requirement.
Thus

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

where G is a 2–output function.

3.4. The Algorithm

The algorithm for functional decomposition based on the presented theory is summarized below. The algorithm first determines the input variable sets A and B for components G and H using the r–admissibility criterion. Next, for an assumed set C, it calculates the maximal compatibility classes for the blocks of partition $P(B \cup C)$ and a minimal cover of the compatibility classes. The number of blocks in that cover is equal to the number of outputs of component G. If the number of outputs of G is consistent with the result of the r–admissibility test, then the solution is found, if not, another set C is tried. In the first run of the decomposer, we check the existence of a disjoint decomposition by assuming $C = \emptyset$. If such a decomposition does not exist, we add variables to C until the decomposition existence criteria are satisfied.

4. Experimental Results

Decomposition algorithm presented in this paper has been implemented for MS–DOS and HP–UX operating systems. For testing purposes we were using a large set of benchmark examples. However, most of them were binary examples so a proper translation was necessary. It usually meant pairing binary variables into 4–valued variables, as PLA with two–bit input decoders was our target. Several results are presented in Table 5.

Tab. 5. Decomposition results for benchmark examples.

Name	Orginal area [OA]	Area after decomposition [AAD]	Profit rate $(1 - \frac{\text{AAD}}{\text{OA}})$ 100%
z9sym	1045	475	54%
rd84	1620	660	59%
life	798	369	54%
rd53	234	180	23%
test4	4830	3524	27%
z4	720	556	23%
ard4	3360	1585	53%

The Table provides PLA area of the original example minimized using ESPRESSO-MV (Brayton *et al.*, 1984) logic minimizer and the total PLA area of the decomposed and then minimized example. To calculate these values we applied the following formula:

$$\text{AREA} = \left(\sum_{i_k \in I} v_k + \lceil \log_2 v_0 \rceil \right) \bullet P \tag{6}$$

where I is the set of input variables, v_k – the number of values of input variable i_k, v_0 – the number of output values, and P – the number of product terms.

3. Conclusions

We have presented a general method for decomposition of incompletely specified multiple–valued Boolean functions. Our approach is based on the original representation of an incompletely specified Boolean function by a set of r–partitions and the corresponding calculus.

Based on the decomposition algorithms presented here, we have developed a prototype version of a logic synthesis system. In this system, the functional decomposition is always carried out at the very beginning of the design process, when the existing don't care conditions can be effectively exploited to minimize the complexity of the resulting components. The system has been used to design several circuits implemented with PLAs, and PLDs.

The presented decomposition procedures are very general. The user can arbitrarily specify the maximum number of inputs and maximum number of outputs required for all the components of a function to be decomposed. Alternatively, the designer can interactively control 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. The maximum acceptable number of inputs, outputs, and cubes (if the minimization procedure is included) for the resulting subfunctions can also be specified. In this way, our decomposition procedure combined, if necessary, with an appropriate minimization procedure allows the designer to examine several alternative solutions. In particular, it makes it possible to compare different implementation styles, e.g., standard PLA vs. PLA with two–bit decoders, and select the one which is most suitable for a given project. This means that the presented algorithms 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 function so that to meet those constraints.

References

Ashenhurst R.L. (1959): *The decomposition of switching functions.* — Proc. Int. Symp. *Theory of Switching Functions.*

Brayton R.K., G.D. Hachtel, C.T. McMullen and A. Sangiovanni–Vincentelli (1984): *Logic Minimization Algorithms for VLSI Synthesis.* — Kluwer Academic Publishers.

Brayton R.K., G. Hachtel and A. Sangiovanni–Vincentelli (1990): *Multilevel logic synthesis.* — Proc. IEEE, v.78, No.2, pp.264–300.

Bolton M. (1990): *Digital Systems Design with Programmable Logic.* — Wokingham: Addison–Wesley Publishing Company.

Ciesielski M. and S. Yang (1992): *PLADE: A two stage PLA decomposition.* — IEEE Trans. on CAD, v.11, No.8, pp.943–954.

Curtis H.A. (1962): *A New Approach to the Design of Switching Circuits.* — Princeton, N.J: D. Van Nostrand Company.

Devadas S., A.R. Wang, A.R. Newton and A. Sangiovanni–Vincentelli (1988): *Boolean decomposition in multi–level logic optimization.* — Proc. Int. Conf. *Computer–Aided Design*, pp.290–293.

Dresig F., Ph. Lanches, O. Rettig and U.G. Baitinger (1992): *Functional decomposition for universal logic cells using substitution.* — Proc. European Conf. *Design Automation*, Brussels, Belgium, pp.38–42.

Hartmanis J. and R.E. Stearns (1966): *Algebraic Structure Theory of Sequential Machines.* — New York: Prentice–Hall.

Hurst S.L., D.M. Miller and J.C. Muzio (1985): *Spectral Techniques in Digital Logic.* — New York: Academic Press.

Jóźwiak L. and F. Volf (1992): *An efficient method for decomposition of multiple–output boolean functions and assigned sequential machines.* — Proc. European Conf. *Design Automation*, Brussels, Belgium, pp.114–122.

Luba T, J. Kalinowski, K. Jasiński and A. Kraśniewski (1991): *Combining serial decomposition with topological partitioning for effective multi–level PLA implementations.* — In: P. Michel and G. Saucier (Eds.), Logic and Architecture Synthesis, pp.243–252, Amsterdam: Elsevier Science Publishers B.V. (North–Holland).

Luba T., J. Kalinowski and K. Jasiński (1991): *PLATO: A CAD tool for logic synthesis based on decomposition.* — Proc. European Conf. *Design Automation*, Amsterdam, The Netherlands, pp.65–69.

Luba T., M. Markowski and B. Zbierzchowski (1992): *Logic decomposition for programmable gate arrays.* — Proc. *Euro–Asic'92*, Paris, France, pp.19–24.

Łuba T. and J. Rybnik (1992): *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, Dordrecht: Kluwer Academic Publishers.

Łuba T., K. Górski and L.B. Wroński (1992): *ROM-based finite state machines with PLA address modifiers.* — Proc. European Conf. *Design Automation*, Hamburg, Germany, pp.272–277.

Muzio J.C. and T.C. Wesselkamper (1986): *Multiple-valued Switching Theory.* — Bristol and Boston: Adam Hilger Ltd.

Pawlak Z. (1991): *Rough Sets. Theoretical Aspects of Reasoning about Data.* — Dordrecht: Kluwer Academic Publishers.

Rudell R.L. and A. Sangiovanni–Vincentelli (1987): *Multiple-valued minimization for PLA optimization* . — IEEE Trans. Computer–Aided Design, v.6, No.5, pp.727–750.

Sasao T. (1984): *Input variable assignment and output phase optimization of PLA's.* — IEEE Trans Comput., v.C–33, No.10, pp.879–894.

Sasao T. (1988): *Multiple-valued logic and optimization of programmable logic arrays.* — IEEE Computer, v.21, pp.71–80.

Saucier G., P. Sicard and L. Bouchet (1990): *Multi-level synthesis on PALs.* — Proc. European Conf. *Design Automation*, Glasgow, U.K., pp.542–546.

Wan W. and M.A. Perkowski (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 Conf. *Design Automation*, Hamburg, Germany, pp.230–235.

Received March 19, 1993