I.J. Information Technology and Computer Science, **2014, 01, 25-32**
Published Online December 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2014.01.03

Improved Decomposition for a System of Completely Specified Boolean Functions

Saeid Taghavi Afshord
Computer Engineering Department, Shabestar Branch, Islamic Azad University, Shabestar, Iran
E-mail: taghavi@iaushab.ac.ir

Yuri Pottosin
United Institute of Informatics Problems, National Academy of Sciences of Belarus, Minsk, Belarus
E-mail: pott@newman.bas-net.by

***Abstract*—** Functional decomposition is an important and powerful technique in the logic synthesis. The ternary matrix cover approach is one of the existing methods of this type. This method is also used in decomposition of a system of completely specified Boolean functions. Before constructing the desired superposition, it needs to encode a table. There is a trivial encoding method. But to find a better solution, it is important to use a special approach, because the result of the encoding has a direct influence on the obtained functions. In this paper, an efficient algorithm to encode this table is presented. It uses the approach connected with the assembling Boolean hyper cube method. The proposed algorithm is explained in details with an example. The benefits and impacts of the suggested technique are also discussed.

***Index Terms*—** Boolean Functions, Compact Table, Boolean Hypercube, Optimization, Encoding and Logic Synthesis

I. Introduction

The problem of decomposition of a system of Boolean functions takes an important place in the logic design of discrete devices based on VLSI circuits [1], [2]. This problem is to represent a given system as a superposition of two or more systems of functions that are less complex than the given one [3], [4], [5]. Two block disjoint decomposition [4], [5] is the simpler and the most applicable type of decomposition which is discussed in this paper.

The problem of minimization of a system of Boolean functions related to the search for the most compact (minimal as for the number of terms and letters in them) representation of the system. The necessity of the search for minimal representations of Boolean function systems and their superpositions arises when the problems of synthesis of digital devices on the base on micro-electronic technology are solved. The conditions that the solutions of those problems must satisfy are determined by the technology. The minimization and decomposition problems are different as for both the statement and the methods for solving, but there is some connection between them. The main goal of the current paper is minimizing the total number of disjunctive normal forms (DNFs) of the decomposed systems. This minimization will decrease the size of a PLA (Programmable Logic Array) which is important from practical point of view [5].

To decompose a system of completely specified Boolean functions, first of all, one should search for an appropriate partition [6], [7]. This is also a challenging task and this paper does not deal with this problem. The suitable partition is supposed that already has been prepared.

Various methods for decomposition of Boolean functions are based on representations of functions. A Boolean function can be given in the form of the compact table [8], [9] that is a two-dimension table as a Karnaugh map or decomposition chart but may have the less size. Using the compact table one can learn rather easily about existence of a solution of the problem for the given function, and find easily the corresponding superposition of functions if such a solution exists. In this paper, the ternary matrix cover approach for decomposition of a system of Boolean functions is used [9]. Cover map and compact table are the key features of this approach. To obtain the systems of Boolean functions as a result of decomposition, one must encode the columns and rows of the compact table according to the obtained covers of ternary matrices. They should be encoded by values of the subsets of arguments separated in a certain way. In addition, each column should be encoded by binary codes in optional manner. But the result of this encoding and consequently its method has the direct influence on the obtained superposition. A novel method for encoding of the columns is suggested that leads to lowering the number of elements of DNFs. The encoding process in the suggested method is described as a constructing a Boolean k-dimensional hypercube. It is like assembling a simple mechanical structure. Here k is related to the number of distinguished columns of the compact table.

The remainder of the paper is organized as follows. Section 2 briefly describes the theoretical part of the problem. Section 3 explains the suggested method for encoding and its computational complexity analysis in details. Section 4 presents an example along with discussion to increase the clarity. Conclusion and future works are given in the final section.

II. Overview of the Problem

2.1 The Problem Definition

The problem of decomposition of a system of Boolean functions is considered in the following statement. Given a system of completely specified Boolean functions $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, the superposition $\boldsymbol{y} = \varphi(\boldsymbol{w}, \boldsymbol{z}_2)$, $\boldsymbol{w} = g(\boldsymbol{z}_1)$ must be found where $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are vector variables whose components are Boolean variables in the subsets Z_1 and Z_2 of the set $X = \{x_1, x_2, \ldots, x_n\}$ of the arguments, respectively such that $X = Z_1 \cup Z_2$ and $Z_1 \cap Z_2 = \varnothing$. At that, the number of components of the vector variable $\boldsymbol{w}$ must be less than that of $\boldsymbol{z}_1$. Such a kind of decomposition is called two-block disjoint decomposition [4], [5]. The subsets Z_1 and Z_2 are called bound and free sets respectively.

Let a system of this kind be given by matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ that are the matrix representation of the system of DNFs of the given functions [4]. Matrix $\boldsymbol{U}$ is a ternary matrix of $l \times n$ dimension where l is the number of terms in the given DNFs. The columns of $\boldsymbol{U}$ are marked with the variables $x_1, x_2, \ldots, x_n$, and the rows represent the terms of the DNFs (the intervals of the space of the variables $x_1, x_2, \ldots, x_n$). The matrix $\boldsymbol{V}$ is a Boolean matrix. Its dimension is $l \times m$, and its columns are marked with the variables $y_1, y_2, \ldots, y_m$. The ones in this columns point out the terms in the given DNFs.

2.2 Cover Map and Compact Table

Any family π of different subsets (blocks) of a set L whose union is L, is called a *cover* of L. Let $L = \{1, 2, \ldots, l\}$ be the set of numbers of rows of a ternary matrix $\boldsymbol{U}$. A cover π of L is called a *cover of the ternary matrix* $\boldsymbol{U}$ if for each value $\boldsymbol{x}^*$ of the vector variable $\boldsymbol{x}$ there exists a block in π containing all the numbers of those and only those rows of $\boldsymbol{U}$, which absorb $\boldsymbol{x}^*$. Block $\varnothing$ corresponds to the value $\boldsymbol{x}^*$, which is absorbed by no row of $\boldsymbol{U}$. Other subsets are not in π. Let $t(\boldsymbol{x}^*, \boldsymbol{U})$ be the set of numbers of those rows of $\boldsymbol{U}$, which absorb $\boldsymbol{x}^*$. For every block π_j of π, we define the Boolean function $\pi_j(\boldsymbol{x})$ having assumed that $\pi_j(\boldsymbol{x}^*) = 1$ for any $\boldsymbol{x}^* \in \{0,1\}^n$ if $t(\boldsymbol{x}^*, \boldsymbol{U}) = \pi_j$, and $\pi_j(\boldsymbol{x}^*) = 0$ otherwise.

Let us define an operation $\vee(\pi_i, \boldsymbol{V})$ over the rows of a binary matrix $\boldsymbol{V}$, the result of which is the vector $\boldsymbol{y}^*$ ($\boldsymbol{y}^* = \vee(\pi_i, \boldsymbol{V})$) obtained by component-wise disjunction of rows $\boldsymbol{V}$ whose numbers are in the block π_i. If $\pi_i = \varnothing$, all the components of $\boldsymbol{y}^*$ are equal to 0. It has been shown in [8] that $\boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{y}^* = \vee(\pi_i, \boldsymbol{V})$ if $\pi_i(\boldsymbol{x}^*) = 1$.

There is a convenient way to construct the cover of a ternary matrix $\boldsymbol{U}$ when the number of arguments is not large. This technique uses the cover map that has the structure of the Karnaugh map. In any cell of a cover map of $\boldsymbol{U}$ corresponding to a vector $\boldsymbol{x}^*$, there is the set $t(\boldsymbol{x}^*, \boldsymbol{U})$, which is a block of the cover of $\boldsymbol{U}$.

Let a pair of matrices, $\boldsymbol{U}$ and $\boldsymbol{V}$, give a system of completely specified Boolean functions $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, and let the matrix $\boldsymbol{U}_1$ be composed of the columns of $\boldsymbol{U}$, marked with the variables from the set Z_1 and the matrix $\boldsymbol{U}_2$ from the columns marked with the variables from Z_2. The covers of $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ are $\pi^1 = \{\pi^1_1, \pi^1_2, \ldots, \pi^1_r\}$ and $\pi^2 = \{\pi^2_1, \pi^2_2, \ldots, \pi^2_s\}$.

Let us construct a table M. Assign the blocks $\pi^1_1, \pi^1_2, \ldots, \pi^1_r$ and the corresponding Boolean functions $\pi^1_1(\boldsymbol{z}_1), \pi^1_2(\boldsymbol{z}_1), \ldots, \pi^1_r(\boldsymbol{z}_1)$ to the columns of M, and $\pi^2_1, \pi^2_2, \ldots, \pi^2_s$ and $\pi^2_1(\boldsymbol{z}_2), \pi^2_2(\boldsymbol{z}_2), \ldots, \pi^2_s(\boldsymbol{z}_2)$ to the rows of M. At the intersection of the i-th column, $1 \le i \le r$ and the j-th row, $1 \le j \le s$, of M, we put the value $\boldsymbol{y}^* = \vee(\pi^1_i \cap \pi^2_j, \boldsymbol{V})$. The table M is called the *compact table*. It gives the system of Boolean functions $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ in the following way: the value of the Boolean vector function $\boldsymbol{f}(\boldsymbol{x}^*)$ is $\vee(\pi_{1i} \cap \pi_{2j}, \boldsymbol{V})$ at any set argument values $\boldsymbol{x}^*$, for which $\pi_{1i}(\boldsymbol{z}_1) \wedge \pi_{2j}(\boldsymbol{z}_2) = 1$.

Having the compact table for a system of functions $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, it is easy to construct the desired systems $\boldsymbol{y} = \boldsymbol{\varphi}(\boldsymbol{w}, \boldsymbol{z}_2)$ and $\boldsymbol{w} = \boldsymbol{g}(\boldsymbol{z}_1)$. The columns of the compact table are encoded with binary codes; equal columns may have the same codes. The length of the code is equal to $\lceil \log_2 r' \rceil$ where r' is the number of different columns of the table and $\lceil a \rceil$ is the least integer, which is not less than a. So, the system of functions $\boldsymbol{w} = \boldsymbol{g}(\boldsymbol{z}_1)$ is defined. The value of the vector variable $\boldsymbol{w}$ at any set of values of the vector variable $\boldsymbol{z}_1$ turning the function $\pi^1_i(\boldsymbol{z}_1)$ into 1 is the code of the i-th column, $1 \le i \le r$. Naturally, there is no solution to this task at the given partition $\{Z_1, Z_2\}$ of the set X of arguments if the length of the code is not less than the length of $\boldsymbol{z}_1$. Otherwise, the compact table whose columns are assigned with the values of the variable $\boldsymbol{w}$ can be considered as a form of representation of the other desired system of functions $\boldsymbol{y} = \boldsymbol{\varphi}(\boldsymbol{w}, \boldsymbol{z}_2)$. The value of $\boldsymbol{y}$ at the value of $\boldsymbol{w}$ assigned to the i-th column, $1 \le i \le r$, and at any value of $\boldsymbol{z}_2$ turning $\pi^2_j(\boldsymbol{z}_2)$ into 1, $1 \le j \le s$, is the vector that is at the intersection of the i-th column and the j-th row [8], [9].

III. Constructing a Boolean Hypercube

3.1 The Preliminary Stages

Let M be a compact table whose columns must be encoded by Boolean vectors, and let w_{ij} be a function taking its values from the set of positive integers on the set of pairs of distinguished columns of M. The values of w_{ij} are obtained from the so-called difference table.

The values of compact table are binary codes. So, for each pair of distinguished columns, the number of different peer-to-peer bits is calculated. These values will form the difference table. At the start of the process of the hypercube construction, the vertices of the hypercube are ones of an empty graph (without edges) and related to the distinguished columns of M.

The input data for constructing the n-dimensional hypercube are the values of difference table and the number of distinguished columns γ of the given compact table M. If γ is not an integer power of two, it will be increased to 2^n where $n = \lceil \log_2 \gamma \rceil$, and virtual vertices should be introduced respectively. It is regarded that $w_{kl} = \infty$ if at least one of the vertices k or l is virtual.

3.2 The Process of Construction

The process of constructing the Boolean hypercube can be represented as the sequence of n steps. At the sth step, the set of $(s-1)$ - dimensional hypercubes are considered. They join into pairs, and s - dimensional hypercube is obtained from each pair by adding edges properly. As far as it is possible, those vertices, i and j, are chosen for being connected with an edge, which have the least value of corresponding w_{ij}. After n steps, an n - dimensional Boolean hypercube will be constructed. The n - component Boolean vectors are assigned to the vertices of the hypercube where the neighborhood relation between the vectors should be represented by the edges of the hypercube.

At the first step of this process, 1 - dimensional hypercubes in the form of 2^{n-1} nonadjacent edges are composed of 0-dimensional hypercubes which is represented by 2^n isolated vertices. At the last, nth step, an n - dimensional hypercube is assembled from two $(n-1)$ - dimensional ones by adding 2^{n-1} edges.

Let us consider the formation of s - dimensional hypercubes on sth step more in details. The form of representation of hypercubes is very important here. Any k – dimensional hypercube is represented by a sequence S of 2^k vertex indices which is taken from the set $\{1, 2, ..., 2^n\}$. The edges are specified implicitly: two vertices are connected with an edge if and only if their places in S correspond to the places of neighbor codes in the Gray code sequence of the same length as the length of S.

Before the performance of any step, the number of hypercubes is always even. More exactly, for sth step it is equal to 2^{n-s}, $0 \le s < n$. The current situation is that some set C_s of s-dimensional hypercubes ($C_s = \varnothing$ before the performance of the step) and some set C_{s-1} of $(s-1)$ - dimensional hypercubes exists. All pairs of hypercubes from C_{s-1} are looked through and one of them is chosen according to the criterion specified at section 3.3. The suitable edges are added to form s - dimensional hypercube from this pair that is introduced to C_s, and then the pair is removed from C_{s-1}. The performance of this step is accomplished when $C_{s-1} = \varnothing$.

3.3 Coupling the Hypercubes

For two $(s-1)$ - dimensional hypercubes which have been represented by sequences S' and S'', the sum Σw_{ij} is calculated. The summing is performed over all pairs i, j of indices of the vertices that take the same places in the sequences. This sum varies with permutations of vertices of one of the sequences, say S''. Of course, only those permutations may be taken here into consideration, which preserve the adjacency relation among the vertices.

For all the proper permutations, Σw_{ij} is calculated. Then according to its minimum value, the equivalent pair of hypercubes is chosen. They are joined into an s - dimensional hypercube by edges between vertices that are in the related places of S' and S''. The sequence that represents the composed hypercube is formed by concatenation of the sequences S' and S''. The sequence S'' may be changed its order according to the selected permutation before the concatenation.

It is convenient to determine the permutation that mentioned above by an operator E_k. The action of E_k results in all the vertices which are adjacent with kth dimension by exchanging their places. Evident properties of such an operator are $E_iE_k = E_kE_i$ and $E_iE_i = 1$. These properties give the possibility to look through 2^{s-1} variants for each pair of hypercubes at the sth step. The variants are generated by applying all combinations of the operators $E_1, E_2, ..., E_{s-1}$.

3.4 Complexity of constructing the hypercube

The number of pairs of hypercubes looked through at the sth step is expressed by the following formula:

$$L'_s = \sum_{i=1}^{2^{n-s}} (2^{n-s} - i + 1)(2^{n-s+1} - 2i + 1)$$
$$= 2^{n-s-1}\left[\tfrac{1}{3}(2^{2(n-s+1)} - 1) + 2^{n-s}\right].$$

Taking into account the permutations which have been determined by the operators E_k, $k = 1, 2, ..., s\text{-}1$, at the sth step of the process, we find the number of variants looked through at this step:

$$L_s = 2^{s-1}L'_s = 2^{n-2}\left[\tfrac{1}{3}(2^{2(n-s+1)} - 1) + 2^{n-s}\right].$$

To facilitate the enumeration of variants, we do not take into consideration the permutations that also preserve the adjacency relation among vertices and they are not concerned with any E_k. Otherwise, L_s' should be multiplied by $(s-1)!\cdot 2^{s-1}$.

Finally, we obtain the number of variants looked through during the whole process of constructing the *n*-dimensional Boolean hypercube:

$$L = \sum_{s=1}^{n} L_s = 2^{n-2}\left[\tfrac{1}{9}(2^{2(n+1)} - 3n - 13) + 2^n\right].$$

Since 2^n is the number of distinguished columns (with virtual vertices) in this case, the number of variants looked through during the described process can be estimated as 3 power polynomial of the number of distinguished columns of a obtained compact table.

IV. An Example

Let the system of completely specified Boolean functions to be given as follows:

```
.i 7
.o 4
.ilb x1 x2 x3 x4 x5 x6 x7
.ob y1 y2 y3 y4
.p 12
1011-1- 1001
-10-1-- 0111
1010-00 1000
00--001 1011
--0---- 1110
--1-00- 0100
1--00-- 0010
-0011-0 1101
1-1---- 1110
1--0100 0101
10--1-0 1101
-1-11-- 1111
.e
```

It has 7 inputs, 4 outputs and the number of DNFs is represented in its description by 12 rows. The partition of the set of arguments into subsets $Z_1 = \{x_2, x_5, x_6, x_7\}$ and $Z_2 = \{x_1, x_3, x_4\}$ is considered. Using the ternary matrix cover approach [8] and according to the section 2, the cover map of the arguments, π will be obtained. Then by dividing π by the covers of the columns of Z_1 and Z_2, π^1 and π^2 are calculated respectively. So the obtained results are: π^1 = {{5, 9}, {1, 5, 9}, {5, 7, 9}, {1, 5, 7, 9}, {2, 5, 9, 12}, {5, 6, 7, 9}, {1, 5, 8, 9, 11}, {2, 5, 9, 10, 12}, {3, 5, 6, 7, 9}, {4, 5, 6, 7, 9}, {3, 5, 8, 9, 10, 11}}; and π^2 = {{4, 6}, {2, 4, 5}, {4, 6, 12}, {1, 6, 9, 11, 12}, {2, 4, 5, 8, 12}, {2, 5, 7, 10, 11}, {2, 5, 8, 11, 12}, {3, 6, 7, 9, 10, 11}}.

According to the section 2, the compact table for the covers π^1 and π^2 is represented by Table 1, as well. It has seven different columns. To encode these columns, three variables are sufficient. The encoding process can be done in an arbitrary manner.

The Boolean hypercube encoding method is used to establish a better superposition. Here, a better solution is related to the size of PLA, and it is specified by the number of rows of the decomposed systems. The smaller number of the rows of each system implies a better solution of the task. The columns are also encoded by trivial encoding method for the comparison purpose.

4.1 The Trivial Encoding Method

The obtained codes for the columns using the trivial and the hypercube encoding method are shown at the bottom of Table 1. The method of trivial encoding is simple. It begins from zero for the first column and continues to the last column by increasing one unit for each distinguished column. The equivalent binary number will be the code of the column. More zeros can be added to the left of each number to fit the length of codes in the desired lengths.

4.2 The Process of the Hypercube Construction

Now, the process of the hypercube construction is explained in a step by step manner. It is like assembling a simple mechanical structure. The difference table is calculated by using the compact table and it is shown in Table 2. It represents the values of function w_{ij}. The rows and columns of this table correspond to the seven distinguished columns of the compact table. To make the number of vertices of the equivalent hypercube equal to an integer power of two (in this example 2^3), a virtual vertex should be added. The infinity values of w_{i8} are not given in Table 2.

Table 2: The difference table for the example of section 4

v_2	v_3	v_4	v_5	v_6	v_7	
1	9	2	5	10	10	v_1
	8	3	4	9	11	v_2
		9	6	1	7	v_3
			7	10	8	v_4
				5	13	v_5
					8	v_6

The minimum is $w_{12} = 1$. Therefore, according to the encoding method which has been explained in section 3.2, the edge v_1v_2 is introduced, then v_3v_6, v_4v_5 and v_7v_8. So, at the first step, four 1 - dimensional hypercubes is obtained that are shown in Fig. 1

Table 1: The compact table for the partition from the example of section 4

	5,9	1,5,9	5,7,9	1,5,7,9	2,5,9,12	5,6,7,9	1,5,8,9,11	2,5,9,10,12	3,5,6,7,9	4,5,6,7,9	3,5,8,9,10,11
4,6	0000	0000	0000	0000	0000	0100	0000	0000	0100	1111	0000
2,4,5	1110	1110	1110	1110	1111	1110	1110	1111	1110	1111	1110
4,6,12	0000	0000	0000	0000	1111	0100	0000	1111	0100	1111	0000
1,6,9,11,12	1110	1111	1110	1111	1111	1110	1111	1111	1110	1110	1111
2,4,5,8,12	1110	1110	1110	1110	1111	1110	1111	1111	1110	1111	1111
2,5,7,10,11	1110	1110	1110	1110	1111	1110	1111	1111	1110	1110	1111
2,5,8,11,12	1110	1110	1110	1110	1111	1110	1111	1111	1110	1110	1111
3,6,7,9,10,11	1110	1110	1110	1110	1110	1110	1111	1111	1110	1110	1111
Trivial	**000**	**001**	**000**	**001**	**010**	**011**	**100**	**101**	**011**	**110**	**100**
Hypercube	**001**	**000**	**001**	**000**	**100**	**011**	**010**	**110**	**011**	**101**	**010**

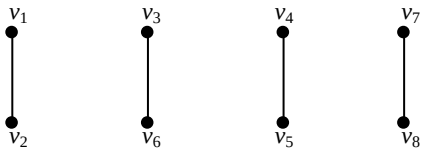

Fig 1: 1-dimensional hypercubes for the example of section 4

Then, 2 - dimensional hypercubes will be constructed. It is performed by adding two edges, v_iv_j and v_kv_l such that $w_{ij} + w_{kl}$ would be the minimum among all the selectable edges. All the variants of such adding are given in Table 3. The edge incident with vertex v_8 is not considered.

Table 3: The variants of the adding edges at the second step

Edges	$w_{ij} + w_{kl}$
v_1v_3, v_2v_6	18
v_1v_6, v_2v_3	18
v_1v_4, v_2v_5	6
v_1v_5, v_2v_4	8
v_3v_4, v_6v_5	14
v_3v_5, v_6v_4	16

Having chosen the first variant, 2 - dimensional hypercube on the vertices v_1, v_2, v_5 and v_4 is obtained. Similarly, the second hypercube is constructed by vertices v_3, v_6, v_8 and v_7. They are illustrated in Fig. 2.

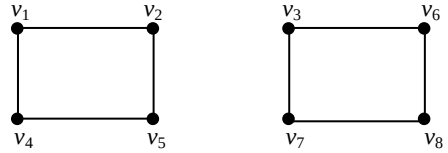

Fig 2: 2-dimensional hypercubes for the example of section 4

To finish the process, four edges should be added to obtain the final 3 - dimensional hypercube. The variants of such adding are given in Table 4. One of the edges among them has infinity value of w_{ij}. Therefore, the value of this edge is not considered in the resulting summation.

Table 4: The variants of the adding edges at the third step

Edges	Σw_{ij}
v_1v_3, v_2v_6, v_5v_8, v_4v_7	25
v_1v_7, v_2v_3, v_5v_6, v_4v_8	23
v_1v_8, v_2v_7, v_5v_3, v_4v_6	27
v_1v_6, v_2v_8, v_5v_7, v_4v_3	31
v_1v_7, v_2v_8, v_5v_6, v_4v_3	24
v_1v_3, v_2v_7, v_5v_8, v_4v_6	30
v_1v_6, v_2v_3, v_5v_7, v_4v_8	31
v_1v_8, v_2v_6, v_5v_3, v_4v_7	23

Having chosen the second variant, the final hypercube is constructed which is shown in Fig. 3.

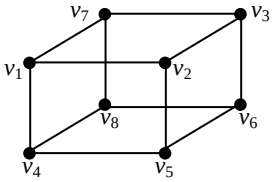

Fig. 3: The final constructed hypercube for the example of section 4

4.3 The Process of Encoding of the Hypercube

The final process to encode the compact table is encoding of the constructed hypercube. The hypercube will be encoded according to the neighborhood relation represented by the graph in Fig. 3.

4.3.1 The basic encoding

The method of the encoding is as follows. At first, the basic encoding is obtained by using Karnaugh map. The sequence of the vertices of the constructed hypercube is put in the Karnaugh map. The order of the vertices in the sequence is considered as Fig. 4a for the hypercube of Fig. 3. In this paper, this sequence is called the vector representation of the hypercube.

It is noticed that the placement of the Gray codes for Karnaugh map is performed according to the

neighborhood relations of the constructed hypercube. It means that the concept of Karnaugh map is used and for some rank of the obtained hypercube some modifications are needed to reorder the map. It is necessary to keep the vicinity relations of the vertices. So, the basic encoding for the hypercube of Fig. 3 is represented in Fig. 4.b.

4.3.2 The optimized encoding

A supplementary improvement is made on the basic encoding if it is needed and it can be possible. It means with additional efforts, sometimes it is possible to achieve to the more efficient encoding. In case the current hypercube contains some virtual vertices, it would be done.

The final aim is obtaining a feasible version of the encoding vector which assigns the maximum possible 1's for the codes of the virtual vertices. A feasible version is related to such permutations keeping the neighborhood relations. The operations of searching for the feasible version are similar to the hypercube construction.

Unfortunately, it is not possible always to find the optimal version, especially when the number of virtual vertices is more. But the basic encoding is still improved. The optimized encoding for the hypercube of Fig. 3 is shown in Fig. 4c. In the current example, the virtual vertex is only v_8. So the binary code "111" has been assigned for it which has the maximum possible 1's. For this example the obtained encoding is optimal.

a.	v_1	v_2	v_5	v_4	v_8	v_6	v_3	v_7
b.	000	001	011	010	110	111	101	100
c.	001	000	010	011	111	110	100	101

Fig. 4: The encoding for the constructed hypercube. (a) The vector representation of the hypercube of Fig. 3 (b) The basic encoding (c) The optimized encoding

Finally, the distinct columns of the compact table will be encoded as $v_1 - 001$, $v_2 - 000$, $v_3 - 100$, $v_4 - 011$, $v_5 - 010$, $v_6 - 110$ and $v_7 - 101$, by using the hypercube encoding method. The repetitive columns of the compact table will be encoded according to their equivalent distinct encoded columns. In fact, the encoded columns will form the vector $\boldsymbol{w}$.

4.4 To Obtain the Desired Superposition

The main object of the paper for this example was done. But, to obtain the solution of the task, the systems of functions $\boldsymbol{y} = \boldsymbol{\varphi}(\boldsymbol{w}, \boldsymbol{z}_2)$ and $\boldsymbol{w} = \boldsymbol{g}(\boldsymbol{z}_1)$ should be constructed [8], [9]. For that, the functions connected with the blocks of the cover maps of Z_1 and Z_2 must be obtained. Then the DNFs of the functions connected with the blocks of π^1 and π^2 will be calculated. After a minimization with the well-known Espresso logic minimizer; the following systems representing the desired superposition $\boldsymbol{y} = \boldsymbol{\varphi}(\boldsymbol{w}, \boldsymbol{z}_2)$, $\boldsymbol{w} = \boldsymbol{g}(\boldsymbol{z}_1)$ are obtained. For that, each of the calculated codes can be used. As mentioned before, for the comparison purpose; the desired superposition using each of them will be constructed.

4.4.1 The obtained superposition by using the trivial encoding method

The first system of superposition by using the trivial encoding method is:

```
.i 6
.o 4
.ilb w1 w2 w3 x1 x3 x4
.ob y1 y2 y3 y4
.p 13
-01111 0001
10--01 0001
010-0- 0001
011--- 0100
1100-- 1111
010--1 1111
101--1 1111
101-0- 1111
--01-- 1110
0--1-- 1110
10-1-- 1111
--0-0- 1110
0---0- 1110
.e
```

And the second system of superposition by using the trivial encoding method is:

```
.i 4
.o 3
.ilb x2 x5 x6 x7
.ob w1 w2 w3
.p 9
0001 100
111- 010
01-0 100
0-11 001
1100 101
1-01 010
00-0 001
100- 001
-00- 010
.e
```

4.4.2 The obtained superposition by using the hypercube encoding method

Also, the first system of superposition by using the hypercube encoding method is:

```
.i 6
.o 4
.ilb w1 w2 w3 x1 x3 x4
.ob y1 y2 y3 y4
.p 10
--0111 0001
-10-01 0001
011--- 0100
1010-- 1111
0--1-- 1010
0---0- 1110
-0-1-- 1110
1-0--1 1111
1-0-0- 1111
-101-- 1111
.e
```

And the second system of superposition by using the hypercube encoding method is:

```
.i 4
.o 3
.ilb x2 x5 x6 x7
.ob w1 w2 w3
.p 8
0001 100
01-0 010
0-01 001
11-- 100
100- 010
--00 010
10-- 001
-00- 001
.e
```

4.5 Discussion

As it is seen from the obtained superpositions, the numbers of rows of the systems which have been constructed using the hypercube encoding method are less than their equivalent systems from trivial encoding method. This reduction is expected to be the more by increasing the number of the arguments or the DNFs of the original system. We also performed this method on the several examples and observed the similar results. In addition, the comparable work has been done on the problem of the state assignment of a finite state machine (FSM) to decrease the power of the implementing circuit [10]. Those results are confirmed our method as well.

V. Conclusion and Future Works

The ternary matrix cover approach is an efficient technique for the problem of decomposition of systems of Boolean functions. The encoding of the compact table columns has a direct influence on the quality and cost of the designing of the digital devices. So, its optimization will cause a significant improvement on the obtained solution. In this paper, we suggested the assembling of Boolean hypercube for the encoding of the columns. This encoding improves the desired superposition and reduces the size of PLA which is important in the practical applications.

Development and implementation of a software tool for investigation of the existing benchmarks is proposed. This will demonstrate the full impacts of the suggested method.

Acknowledgements

This work was done in the logical design laboratory at the united institute of informatics problems of the NAS of Belarus. The authors like to thank this laboratory by its support in providing the examples of the completely specified Boolean function to test.

References

[1] Martinelli A. Advances in Functional Decomposition: Theory and Applications. Doctoral Dissertation, Royal Institute of Technology (KTH), Stockholm, Sweden, 2006

[2] Morawiecki P, Rawski M, Selvaraj H. Application of Functional Decomposition in Synthesis of Boolean Function Sets. In: Proceedings of the 19th International Conference on Systems Engineering (ICSENG), Aug. 2008, Las Vegas, USA, 350-355.

[3] Perkowski M A, Grygiel S. A Survey of Literature on Functional Decomposition, Version IV. Technical report, Department of Electrical Engineering, Portland State University, Portland, USA, 1995

[4] Zakrevskij A, Pottosin Yu V, Cheremisinova L. Optimization in Boolean Space. Tallinn, Estonia, TUT Press, 2009

[5] Hassoun S, Sasao T. Logic Synthesis and Verification. The Springer International Series in Engineering and Computer Science, Kluwer Academic Publishers, 2001.

[6] Rawski M. Input Variable Partitioning Method for Decomposition-Based Logic Synthesis targeted Heterogeneous FPGAs. International Journal of Electronics and Telecommunications, 2012, vol 58 (1), 15–20.

[7] Muthukumar V, Bignall R J, Selvaraj H. An efficient variable partitioning approach for functional decomposition of circuits. Journal of Systems Architecture, 2007, vol. 53, no. 1, 53-67.

[8] Pottosin Yu V, Shestakov E A. Tabular Methods for Decomposition of Systems of Completely Specified Boolean Functions. Byelorusskaya Nauka, Belarus, 2006 (in Russian)

[9] Taghavi Afshord, S, Pottosin, Yu V. On Decomposing Systems of Boolean Functions via Ternary Matrix Cover Approach. International Journal of Advanced Science and Technology, June 2013, Vol. 55, 33-42.

[10] Pottosin, Yu V, Pottosina, S A. State assignment of a finite state machine for a low power implementing circuit. In: Proceedings of the 8th International Conference, Nov. 2011, University of Zilina, Zilina, Slovak Republic, 113-116.

Authors' Profiles

Saeid Taghavi Afshord received his BSc degree in applied mathematics and MSc degree in computer engineering from the Islamic Azad University, Tabriz and Qazvin branches in 2003 and 2006, Iran respectively. He joined the Islamic Azad University, Shabestar branch, Iran, as a faculty member in 2008. Currently he is a PhD student in computer engineering at the United Institute of Informatics Problems of the NAS of Belarus, from March 2011. His research areas are Energy Saving in Ad Hoc and Sensor Networks, Methods for Boolean functions Decomposition, and optimization problems.

Yuri Vasilievich Pottosin graduated from Tomsk State University (Russia), department of radio-physics and electronics, in 1960. From the beginning of 1961 until 1973, he worked at that university as a researcher. In 1970, he defended his PhD thesis. From 1973 until now, he works at the United Institute of Informatics Problems of National Academy of Sciences of Belarus. Now he is a leading researcher. His main scientific interest is logical design. He also teaches the related courses for the students of Byelorussian State University of Informatics and Radio-Electronics.

