Determining Covers in Combinational Circuits

Ljubomir Cvetković[1] and Darko Dražić[2]

**[1] Teacher Training College
Sremska Mitrovica, 22000, Serbia**

**[2] Teacher Training College
Sremska Mitrovica, 22000, Serbia**

Abstract

In this paper we propose a procedure for determining 0– or 1– cover of an arbitrary line in a combinational circuit. When determining a cover we do not need Boolean expression for the line; only the circuit structure is used. Within the proposed procedure we use the tools of the cube theory, in particular, some operations defined on cubes. The procedure can be applied for determining 0– and 1– covers of output lines in programmable logic devices. Basically, this procedure is a method for the analysis of a combinational circuit.

Keywords: *Combinational Circuit, Cover, Logical Relation, Cube.*

1. Introduction

Traditionally, 0– or 1– cover of a line in a combinational circuit is determined using Boolean expression. There are well-known procedures for determining covers in the case when the function is given in the form of a minimal disjunctive normal form or minimal conjunctive normal form. In the case of disjunctive normal form a cube is associated to each elementary product and it represents the set of vectors on which this product has value 1. The 1– cover is determined on the basis of the correspondence between elementary products and cubes.

Getting 0– or 1– cover on the basis of truth table or Binary Decision Tree is a difficult task, especialy in the case of a great number of variables.

Since we need Boolean expression for determining the cover of a line in a combinational circuit, some methods of minimization are quoted.

The Quine–McCluskey method is a program–based method that is able to carry out the exhaustive search for removing shared variables. The Quine–McCluskey method is a two step method which comprises of finding Prime Implicants and selecting a minimal set of Prime Implicants [5]. Each Boolean function can be represented by its disjunctive normal form (DNF). A lot of Boolean function research has been devoted to minimal DNFs ([1], [8] and [9]). The generation of prime implicants (PIs) of a given function is an important first step in calculating its minimal DNF, and early interest in PIs was mainly inspired by this problem.

Generally, minimization of functions with a large number of input variables is a very time–consuming process and the results are often suboptimal. Most of the practical applications rely on heuristic minimization methods [6] with a complexity which is roughly quadratic in the number of products.

Using general DT structure, a new worst case algorithm to compute all prime implicants is presented in [4]. This algorithm has a lower time complexity than the well–known Quine–McCluskey algorithm and is the fastest corresponding worst case algorithm so far.

A SOP representation based on a *"ternary tree"* is well known. Compared to BDDs where the size can grow exponentially with the number of input variables, size of ternary tree grows only linearly with the number of inputs in the worst case. The first simple ternary tree minimization algorithms were proposed in [2], [3].

A method proposed in [7] utilizes data derived from Monte–Carlo simulations for any Boolean function with different count of variables and product term complexities. The model allows design feasibility and performance analysis prior to the circuit realization.

2. Preliminary considerations

Our procedure for determining covers in combinational circuits uses cube theory and therefore we provide necessary definitions.

A cube is a vector $a_1a_2...a_n$, where $a_i \in \{0,1,X\}$ and X is a variable of the set $\{0,1\}$ (i=1,2,…,n). Hence, a cube is a set of vectors from $\{0,1\}^n$. Elements $a_1,a_2,...,a_n$ are coordinates of the cube. A cube has rank r, if it contains r coordinates equal to X. A cube of rank r is called r–cube.

A set of cubes is called 0–cover (1–cover) of line i if it contains all input vectors generating signal of value 0 (value 1) on this line.

Definition 1. The intersection of cubes $A = a_1a_2...a_n$ and $B = b_1b_2...b_n$ is the cube $C = c_1c_2...c_n$, where $c_i = a_i \varnothing b_i$, i = 1,2,...,n. The intersection operation $\varnothing$ is defined on the set $\{0,1,X\}$ by Table 1. In Table 1 the symbol $\varnothing$ denotes that the operation $\varnothing$ is not defined. The intersection of cubes A and B is defined, if for any a_i and b_i the intersection operation is defined, i.e. $a_i \varnothing b_i \neq \varnothing$.

Table 1: Operation $\varnothing$

$\varnothing$	0	1	X
0	0	$\varnothing$	0
1	$\varnothing$	1	1
X	0	1	X

Definition 2. The cut of cube sets Q_1 and Q_2 is denoted by $Q_1 \cap Q_2$ and is the set of all cuts of a cube from Q_1 with a cube from Q_2.

Definition 3. The union of cube sets Q_1 and Q_2 is denoted by $Q_1 \cup Q_2$. It contains all cubes from both Q_1 and Q_2.

Definition 4. A cube $B = b_1b_2...b_n$ is said to be a part of the cube $A = a_1a_2...a_n$, if all vectors of B belong also to A. Obviously, B is a part of A only if for any $a_i \neq X$ we have $a_i = b_i$.

Definition 5. If a cube B is a part of the cube A and if both cubes belong to the same set of cubes, then B can be deleted from the considered set of cubes. This modification is called cube absorption. In particular, we say that A absorbs B. As noted, this is possible if for any $a_i \neq X$ we have $a_i = b_i$.

Suppose that a cube generates a signal $s \in \{0,1\}$ at a line **i** in combinational circuit which will be denoted by i=s. We shall say that the cube satisfies relation i=s, i.e. represents its solution.

Consider arbitrary lines **i** and **j** in combinational circuit. Let $s_i,s_j \in \{0,1\}$ be the signals at **i** and **j**, respectively. Then the following lemmas hold:

Lemma 1. If cubes A and B satisfy relations $i=s_i$ and $j=s_j$ respectively, then the cube $C=A \varnothing B$ satisfies relation $(i=s_i) \wedge (j=s_j)$.

Proof. The proof follows from the fact that the cut of cubes is equivalent to the cut of sets of vectors represented by these cubes.

Lemma 2. Let S_i, S_j be sets of cubes satisfying $i=s_i$, $j=s_j$, respectively, then all cubes of the set $S_i \cup S_j$ satisfy relation $(i=s_i) \vee (j=s_j)$, while all cubes of the set $S_i \varnothing S_j$ satisfy relation $(i=s_i) \wedge (j=s_j)$.

Proof. The proof immediately follows from the definition of the union and the cut of cube sets.

Let $S_{u_i}(0)$ and $S_{u_i}(1)$ be cube sets generating signal values 0 and 1 on the input lines u_i, i=1,2,...,n of a logical element, considered either separately or within a combinational circuit. Based on Lemma 2 and properties of logical elements, one can formulate the following corollaries.

Corollary 1. In the case of elements OR and NOR the cut $S_{u_1}(0)\phi S_{u_2}(0)\phi...\phi S_{u_n}(0)$ represents the set of cubes generating on the output line v signal value 0 for element OR, and signal value 1 for element NOR. The union $S_{u_1}(1)\cup S_{u_2}(1)\cup...\cup S_{u_n}(1)$ represents the set of cubes generating on the output line v signal value 1 for element OR, and signal value 1 for element NOR.

Corollary 2. In the case of elements AND or NAND the union $S_{u_1}(1)\cup S_{u_2}(1)\cup...\cup S_{u_n}(1)$ represents the set of cubes generating on the output line v signal value 1 for element AND, and signal value 0 for element NAND. The cut $S_{u_1}(0)\phi S_{u_2}(0)\phi...\phi S_{u_n}(0)$ represents the set of cubes generating on the output line v signal value 0 for element AND, and signal value 1 for element NAND.

3. Determining a Cover

0– or 1–cover of input lines of the combinational circuit and of output lines of elements of the first level are determined directly (using basic rules for the logic elements).

0– or 1–cover of an arbitrary line of the combinational circuit, which is an output line of an element of the second or higher level is determined by the following two steps:

1. For an arbitrary line i, for which we want to determine a cover, we write logical relation defining conditions for generating the signal of given value. This logical relation is written on the basis of basic laws for the considered element. The left hand side of the relation determines signal values on all input lines of the element whose output line is the line i. For each line on the left hand side of the logical relation, we write a new logical relation defining conditions for generating the signal of expected value. We keep writing logical relations until we come to relations on whose left hand sides only input lines of the network or output lines of the first level appear.
2. For each line in left hand sides of the relation determined in step 1 we determine the distance in the following way. Input lines of the combinational circuit have the greatest distance r. For all lines at distance r–1 we determine cube sets generating expected signal values on these lines. Next, for all lines at distance r–2 we determine cube sets generating expected signal values on these lines using cube sets obtained for lines at distance r–1. We continue in this way until we get the cover for line i.

Example 1. Determine 1–cover of line i in the combinational circuit of Fig. 1.

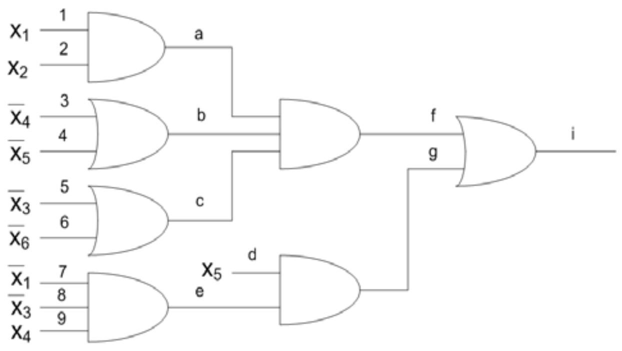

Fig. 1 Combinational circuit.

1. Logical relation defining conditions for generating the signal of value 1 reads:

$$(f = 1) \vee (g = 1) \rightarrow (i = 1) \qquad (1)$$

Signals f=1 and g=1 are defined on the left hand side of the relation. The following logical relations define conditions for generating signals f=1 and g=1:

$$(a = 1) \wedge (b = 1) \wedge (c = 1) \rightarrow (f = 1) \qquad (2)$$
$$(d = 1) \wedge (e = 1) \rightarrow (g = 1) \qquad (3)$$

Since output lines a,b,c,e of the first level and input line d have appeared on the left hand side of the above relations, we proceed to step 2.

2. We determine distances for all lines appearing on the left hand side of logical relations obtained in step 1. Input lines of the circuit 1–9 have the gratest distance. Lines a,b,c,d and e are at distance 2. Lines f and g are at distance 1.

We construct the table Table 2.

Table 2: Line at distance 1

a=1	11XXXX
b=1	XXX0XX XXXX0X
c=1	XX0XXX XXXXX0
d=1	XXXX1X
e=1	0X01XX

Necessary signal values at lines at distance 1 are f=1 and g=1.

By the relation Eq. (2) we get 1–cover of line f:

$$\{11XXXX\} \; \text{Ø} \begin{Bmatrix} XXX0XX \\ XXXX0X \end{Bmatrix} \text{Ø} \begin{Bmatrix} XX0XXX \\ XXXXX0 \end{Bmatrix} = \left\langle \begin{matrix} 1100XX \\ 11X0X0 \\ 110X0X \\ 11XX00 \end{matrix} \right\rangle$$

By the relation Eq. (3) we get 1–cover of line g:

$$\{XXXX1X\} \; \varnothing \; \{0X01XX\} = \{0X01XX\}$$

By the relation Eq. (1) and using operation $\cup$ we get 1–cover of line i:

$$\left\langle \begin{matrix} 1100XX \\ 11X0X0 \\ 110X0X \\ 11XX00 \end{matrix} \right\rangle \cup \{0X011X\} = \left\langle \begin{matrix} 1100XX \\ 11X0X0 \\ 110X0X \\ 11XX00 \\ 0X011X \end{matrix} \right\rangle$$

If at least one cube can be deleted from a cover while the remaining cubes still form a cover, then the cover is redundant. In the other case the cover is irredundant.

The proposed procedure can be applied to combinational circuits with branchings of input and internal lines as well. Programmable logic devices (PLA, PAL and ROM) represent two–level combinational circuits. A PLA whith n input lines, m internal lines and p output lines is represented in Fig.2.

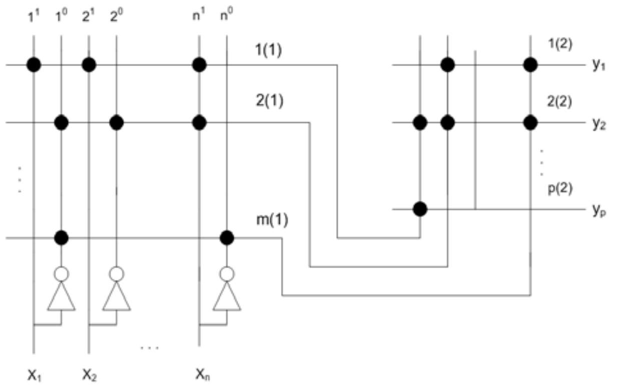

Fig. 2 Programmable logic array

Programmable elements are denoted by symbol "•". We apply the following way of marking programmable points at PLA:

(i,j^1) - cross point of internal line i and a bit line j^1 in AND array (i=1,2,...,m, j=1,2,...,n)

(i,j^0) - cross point of internal line i and a bit line j^0 in AND array (i=1,2,...,m, j=1,2,...,n)

(i,j) -cross point between the lines i and j in OR array (i=1,2,...,p, j=1,2,...,m)

We propose the following procedure for determining 0– or 1–cover of a given output line i(2)=1,2,...,p.

1. We determine the set of test cubes $Q_2(i)$, i=1,2,…,p, which yields 0– or 1–cover of the line i(2)=1,2,...,p, when applied on input lines of the OR array 1,2,...,m. We have i(2)=0 or i(2)=1, depending on whether 0– or 1–cover is determined. When determining the set $Q_2(i)$, we assign the coordinate X ($X\in\{0,1\}$) to input lines of the OR array 1,2,...,m which do not have cross points with the line i(2).

2. The values from the set $Q_2(i)$, i=1,2,…,p, obtained within step 1, are assigned using backtracking to output lines of the AND array i(1), i=1,2,…,m (when backtracking the signal complementation may occur). On the basis of signal values on the output lines of the AND array, the set of test cubes $Q_1(i)$, i=1,2,...,m, is directly determined.

Using the cut $\varnothing$ of cubes we have:

$$Q= Q_1(1)\varnothing\, Q_1(2)\varnothing \dots \varnothing Q_1(m) \qquad (4)$$

Expanding the set of test cubes Q we obtain the input vectors representing 0– or 1–cover of line i(2).

Example 2. Determine 1–cover of line 1(2) for the PAL of Fig. 3.

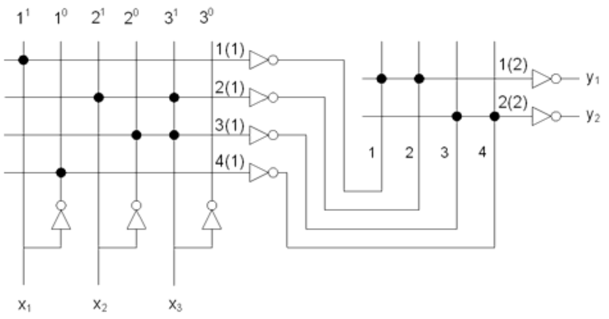

Fig. 3 Programmable array logic

We assign signal values 1 to the points (1,1) and (1,2). We get test cubes set

$$Q_2(1)=\{11XX\}$$

We go back towards output lines of the AND array and assign to these lines the values from $Q_2(1)$, as presented in Fig4.

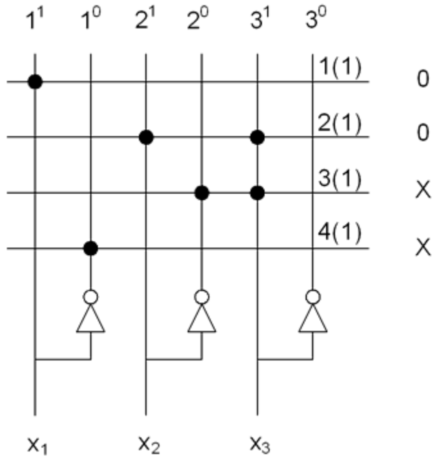

Fig. 4 Signals at output lines of the AND array

For lines 1(1) and 2(1) we determine:

$$Q_1(1)=\{0XX\} \qquad Q_1(2)=\{X00, X01, X10\}.$$

We apply the cut of cubes:

$$Q = Q_1(1)\varnothing\, Q_1(2)=\{000, 001, 010\}.$$

Vectors 000, 001 and 010 represent a 1–cover of line 1(2).

4. Conclusion

The described procedure for determining covers is basically a method for the analysis of combinational circuits. When determining a cover we do not need an analitycal expression; only the circuit structure is used. In particular, we do not need disjunctive normal forms, what is of some theoretical and practical importance. The simplification is in the fact that the cover is determined by moving from inputs towards output lines of the combinational circuit using only the operation of the cut of cubes. In addition, we present a procedure for determining covers of output lines for programmable logic devices. Within proposed procedures the cube theory plays an essential role.

References

[1] P. Clote, and E. Kranakis, Boolean Functions and Computation Models, Berlin Heidelberg: Springer Verlag, 2002.

[2] P. Fišer, P. Rucký, and I. Váňová, ”Fast Boolean Minimizer for Completely Specified Functions”, Proc. 11th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS ’08), Bratislava, 2008, pp. 122-127.

[3] Petr Fišer, and David Toman, ”A Fast SOP Minimizer for Logic Functions Described by ManyProduct Terms”, Proceedings of 12th Euromicro Conference on Digital System Design (DSD’09), Patras, 2009, pp. 757-764.

[4] M. Friedel, S. Nikolajewa, and T. Wilhelm, ”The Decomposition Tree for analyses of Boolean functions”, Math. Struct. in Comp. Science, vol. 18, 2008, pp. 411–426.

[5] E.J McCluskey, ”Minimization of Boolean functions”, The Bell System Technical Journal, 35, No. 5, Nov. 1956, pp. 1417-1444.

[6] A. Mishchenco, and T. Sasao, ”Large-Scale SOP minimization Using Decomposition and Functional Properties”, DAC 2003, pp. 149-154.

[7] P.W. Chandana Prasad, and Azam Beg, and Ashutosh Kumar Singh, ”Effect of Quine-McCluskey Simplification on Boolean Space Complexity”, IEEE Conference on Innovative Technologies in Intelligent Systems & Industrial Applications, Bandar Sunway, 2009.

[8] Y. Wang, and C. McCrosky, and X. Song, ”Single-faced Boolean Functions and their Minimization”, Computer Journal 44 (4), 2001, pp. 280–291.

[9] I. Wegener, Branching Programs and Binary Decision Diagrams – Theory and Application, SIAM Monographs on Discrete Mathematics and Applications, Society for Industrial & Applied, 2000.

Ljubomir Cvetković: received the PhD degree from the University of Belgrade (Faculty of Electrical Engineering) in 2005. He is currently a professor in Teacher Training College in Sremska Mitrovica in Serbia. His major research interests include Digital VLSI architecture, Fault-tolerance and Fault detection. He has published about 20 publications in journals and international conferences and has written 3 books.

Darko Dražić: received the BSc degree from the University of Belgrade (Faculty of Organizational Sciences) in 2005. He is currently a PhD student on software engineering at Faculty of Organizational Sciences. His research interests include Computer architecture, Information system, Audio and video processing.