International Journal of Innovative Computing, Information and Control
Volume **8**, Number **6**, June **2012**

ICIC International ©2012 ISSN 1349-4198
pp. 4333–4345

A REDUCED OFFSET BASED METHOD FOR FAST COMPUTATION OF THE PRIME IMPLICANTS COVERING A GIVEN CUBE

Fatih Başçiftçi[1], Sirzat Kahramanli[2] and Murat Selek[3]

[1]Department of Electronic and Computer Education
Technical Education Faculty
[3]Technical Vocational School of Higher Education
Selcuk University
Konya 42003, Turkey
{ basciftci; mselek }@selcuk.edu.tr

[2]Department of Computer Engineering
Engineering Faculty
Mevlana University
Konya 42250, Turkey
skahramanli@mevlana.edu.tr

Received March 2011; revised August 2011

Abstract. *In order to generate prime implicants for a cube of a logic function, most logic minimization methods expand this cube by one at a time removing the literals from it. However, there is an intractable problem of determining the order of literals to be removed from the cube and checking whether a tentative literal removal is acceptable. In order to avoid this problem, the reduced offset method was developed. This method uses the positional-cube notation where every reduced off-cube of an n-variable function is represented by two n-bit strings. However, unfortunately, the conversion of such reduced cubes to the associated prime implicants has the time complexity worse than exponential. To avoid this problem, in this study, the method representing every reduced cube by a single n-bit string and a set of bitwise operations to be performed on such strings are proposed. The theoretical and experimental estimations show that this approach can significantly improve the quality of results and reduce the space and time complexities of the logic minimization process 2 times and up to 3.5 times, respectively.*

Keywords: Logic minimization, Prime implicant, Reduced offset, Cube notation

1. Introduction.

Sum-of-products (SOP) minimization is a basic problem in logic synthesis [1-3]. It is used for optimization the logic-networks [3,4], for optimizing the test generators [4,5], for obtaining the shortest paths between source and target nodes in hypercube configured systems [6-8] and for attribute reduction [9-17] and rule generation in information systems [10,11,13,18-20]. Unfortunately, due to the exponential nature of the exact SOP minimization, the state-of-the-art algorithms can typically handle functions with up to hundred product terms (cubes) in the minimum SOP [1]. Therefore, most computer aided design tools rely on *direct-cover heuristic minimization methods* [1,5,21-24]. A function f to be minimized by such a method is represented by the *onset, offset* and *do not care set* that are the sets of minterms (cubes) making the function f equal to 1, equal to 0 and unspecified, respectively. We denote these sets by S_{ON}, S_{OFF} and S_{DC}, and their sizes (cardinalities) by $|S_{ON}|$, $|S_{OFF}|$ and $|S_{DC}|$, respectively. In general, the size of a set X will be denoted by $|X|$. Every element of the sets S_{ON}, S_{OFF} and S_{DC} is called an *on-cube*, *off-cube* and *do not care cube*, respectively.

A typical direct-cover heuristic minimization method realizes the SOP minimization of a function f by the following algorithm [1,25-27].

```
Direct_Cover ($S_{ON}, S_{OFF}$) //The output is to be a minimal form of the function
$W(f) = \emptyset$    // The set of selected PIs
Until $S_{ON} = \emptyset$ {
  1) An on-cube $P \in S_{ON}$ to be covered is chosen;
  2) The set of $S_{PI}(P)$ of PIs covering the cube $P$ is generated;
  3) From the set $S_{PI}(P)$, the essential prime implicant $E(P)$ is identified;
  4) The set $S_{ON}$ is covered by $E(P)$ and its rest is considered as $S_{ON}$ to be processed
     in the next iteration; $W(f) = \emptyset = W(f) \cup E(P)$. }
```

FIGURE 1. A typical direct-cover heuristic minimization algorithm

In the algorithm *Direct_Cover* (Figure 1), the most time-consuming step is the second one where the PIs are usually generated by the *implicant expansion* (reduction) method [3,5,19,20,28,29]. In [5,28-30], it is stated that this step is of polynomial complexity in the *number of Boolean variables* n. However, our estimations given below show that this step is of exponential complexity in n.

Recall that an implicant of a function F is such a *product term* that covers at least one cube from S_{ON} and does not cover any cube from S_{OFF} [5,25,27]. Therefore, an implicant being expanded by removing any *literal* (variable or its complement) from it should be intersected with all the elements of the set S_{OFF} to determine whether a tentative literal removal is acceptable [5,25,27]. Since always $|S_{ON}| = k \times 2^n$, where $k < 1$, the *time complexity* of a PI construction process should be specified *as exponential* in n.

Note that the computational efficiency of the *expand operator* and the quality of the result (the size of the final cover) generated by it depend on two factors [5,25]:

1) The order in which the implicants are expanded,
2) The order in which the literals are removed from the implicant being expanded.

The rationale strategy for the first factor is to expand firstly those implicants that are unlikely to be covered by other ones [5,21-23,25,26]. There are also several strategies for the second factor such as *Sequential Search*, *Multiple Sequential Search*, *Distributed Multiple Sequential Search*, and *Distributed Exhaustive Implicant Expansion* among which usually the *Sequential Search* strategy is preferred [5]. Note that even some differences in implementation of the *Expand operator* would lead to different covers with different sizes [2,5,25,26]. Therefore, to improve the quality of results, the programs such as *MINI, PRESTO* and *Espresso* manipulate the results of the *Expand* operator by the additional operators such as *Reduce* and *Reshape* [5,25,27]. Unfortunately, none of the heuristic algorithms is consistently better than the others for all logic functions [25,26]. In order to avoid the problems specific to the implicant expansion concept, A. A. Malik, R. K. Brayton, A. R. Newton and A. Sagniovanni-Vincentelli developed a *reduced offset-based minimization method* [3,31]. An algorithm realizing this method consists of three steps one that intractable due to its exponentional complexity. However, this complexity may be reduced in a large scale if to reducing the reduced off-cubes specification space from $\{0, 1, x\}^n$ into $\{0, 1\}^n$. We show that this is possible by using a data structure named as a *difference indicator* and using a few logic operations developed especially for it (Section 3). This approach allows us to narrow the reduced off-cubes search space by a factor of $3^n/2^n = 1.5^n \times 2^n/2^n = 1.5^n$ that increases rapidly with the increase of n. For instance, it takes the values 58, 3325, 191751 and 11057331 for n = 10, 20, 30 and 40 respectively. Moreover, we show that usually a very large part of implicants generated during the conversion of *product of sums* (POS) form of a function to its SOP form is redundant (Section

3). Therefore, to reduce the space and time complexities of POS to SOP conversion, we developed a method that prevents the generation of redundant implicants. This method allows us to reduce POS to SOP conversion process on a very large scale. For instance, for $n = 20$ and $n = 40$, the space complexity of the process is reduced 10^5 and 10^{11} times, respectively (see Table 1). This property of our method allows us to generate the exact PIs instead of ones heuristically generated by other methods. Due to these advantageous, for most tested functions including benchmark ones, our method has generated the results significantly better than those generated by other methods.

The rest of the study is organized as follows. In Section 2, the complexity of the reduced offset based algorithm is estimated. In Section 3, the method of representation of the reduced off-cubes by n-bit strings and the method generating PIs by using these strings are explained. In Section 4, the results of experiments performed on 45 standard single-output MCNC benchmarks are given. The paper is concluded by Section 5.

2. The Reduced Offset Based Logic Minimization Method.

Recall that the reduced offset method was developed to speed up the second step of the algorithm *Direct_Cover* given in Figure 1. According to this method, the reduced offset $S_R(P)$ is generated first by special handling the elements of S_{OFF} on the chosen on-cube P and second by minimizing the set $S_R(P)$ into the minimal set $S_{RM}(P)$ of the reduced cubes (RCs). Then the set $S_{RM}(P)$ is transformed into the set $S_{PI}(P)$ that contains all PIs covering the on-cube P. This method can be realized by the following three procedures [3].

2.1. The procedure *Reduce_S_{OFF}*.

This procedure transforms each cube $Z = z_{n-1}z_{n-2} \ldots z_0 \in S_{OFF}$ into the corresponding reduced cube $Z^r = c_{n-1}c_{n-2} \ldots c_0$ as follows [3,30]:

$$\text{If } p_i = \bar{z}_i \text{ then } c_i = z_i \text{ else } c_i = x, \quad \forall i \in \{0, 1, \ldots, n-1\} \tag{1}$$

where $P = p_{n-1}p_{n-2} \ldots p_0$ is an on-cube on which the set S_{OFF} is reduced and a literal x appearing in the position of any variable means that this variable is *do not care* for Z^r. In the other words, the procedure *Reduce_S_{OFF}* transforms the set S_{OFF} into a set $S_R(P)$ by removing all literals from the cubes of S_{OFF} except those that are complements of the corresponding literals in P [3]. We estimated the complexity of the procedure *Reduce_S_{OFF}* via the complexity of an assembly-based subprocedure reducing the cubes by Formula (1) and performing 6 bitwise operations per cube. Since this subprocedure must be repeated for all off-cubes, the time complexity of the procedure *Reduce_S_{OFF}* is to be $6 \times |S_{OFF}|$ computer's instruction cycles. Since $|S_{OFF}| = k \times 2^n$ where $k < 1$, the *worst-case time complexity* (WCTC) of the procedure *Reduce_S_{OFF}* is to be *exponential* in n, i.e., *W(Reduce_S_{OFF})* $= O(2^n)$.

2.2. The procedure *Minimize_$S_R(P)$*.

This procedure removes from $S_R(P)$ all cubes absorbed by other ones present in $S_R(P)$ [3,16,17,32]. The work of this procedure may be formally expressed as follows:
For $\forall i, j$: $i \in \{1, 2, \ldots, |S_{OFF}| - 1\}$ and $j \in \{2, \ldots, |S_{OFF}|\}$

$$(Z_i^r, Z_j^r) = \begin{cases} Z_j^r, \text{ if } Z_i^r \cap Z_j^r = Z_i^r \\ Z_i^r, \text{ if } Z_i^r \cap Z_j^r = Z_j^r \\ (Z_i^r, Z_j^r), \text{ if } Z_i^r \cap Z_j^r \notin \{Z_i^r, Z_j^r\} \end{cases} \tag{2}$$

We estimated the time complexity of this procedure via a subprocedure *Minimize_$S_R(P)$* performing 3 bitwise, 2 conditional and 3 return instructions per comparison of type (2). Since in worst case, the subprocedure *Minimize_$S_R(P)$* must be applied $|S_{OFF}| - j$ times for each $Z_j^r \in S_R(P)$, the WCTC of this procedure is to be quadratic in $|S_{OFF}|$. But since

$|S_{OFF}| = k \times 2^n$, $W(Minimize\_S_R(P)) = O(|S_{OFF}|^2) = O((k \times 2^n)^2) = O(k \times 2^{2n})$. That is, the WCTC of the procedure $Minimize\_S_R(P)$ is even worse than exponentional.

2.3. **The procedure *Generate_S*$_{PI}$(*P*).** The function realized by this procedure may be expressed as follows [6,33,34].

$$S_{PI}(P) = \{x\}^n \# S_{RM}(P) = \bar{S}_{RM}(P) = \bar{Z}_1^r \& \bar{Z}_2^r \& \ldots \& \bar{Z}_m^r \tag{3}$$

where $\{x\}^n$ is the n-dimensional Boolean cube, # is the *sharp-product operation* sign, and $m = |S_{RM}(P)|$. Formula (3) is realized as follows:

1. The set $S_{RM}(P)$ is represented in a POS form,
2. The obtained POS is converted to a SOP form.

Example 2.1. *Let* $P = 001$ *and* $S_{OFF} = \{000, 100, 111\}$. *Find* S_{PI} (001). *In the solution of this example given below, the conversions performed by Formulas (1), (2), and (3) are denoted by* T_1, T_2 *and* T_3, *respectively.*
$T_1 : (P, S_{OFF}) \to S_R(001) = \{xx0, 1x0, 11x\}$;
$T_2 : S_R(001) \to S_{RM}(001) = \{xx0, 11x\}$; // $1x0$ *was removed due to* $1x0 \subset xx0$
$T_3 : S_{RM}(001) \to S_{PI}(001) = \bar{S}_{RM}(001) = \{\overline{xx0}\}\&\{\overline{11x}\} = \{xx1\}\&\{0xx, x0x\} = \{0x1, x01\}$

Our experiments over a lot of functions have shown that usually the size of $S_{RM}(P)$ does not exceed $2.5n$. Namely, usually the maximum number of clauses forming a POS is limited above by $2.5n$. Since usually each clause contains at least two literals, the expected size of non-minimized SOP may not be less than $O(2^{2.5n})$. In our opinion, this is one of the main reasons making the reduced offset method time-consuming when S_{OFF} is unreasonable large and there are many on-cubes to be handled [3].

3. Reducing the Space Complexity of the Reduced Offset Method and Generating the Prime Implicants.

3.1. **Representation of a reduced cube by a difference indicator.** In order to represent the cubes, the reduced offset method uses *positional-cube notation* as well as most other minimization methods. According to this notation, an uncomplemented variable y_i, a complemented variable $\bar{y}_i$ and a '*do not care variable*' (missing variable in a product term) are represented by bit-pairs 01, 10 and 11, respectively [3,5,25,27]. If to denote the left and right bits of each bit-pair by L and R, respectively, then a cube $Z = z_n z_{n-1} \ldots z_1$ may be represented by the pair $Z = (Z_L, Z_R) = (L_n, L_{n-1}, \ldots, L_1, R_n, R_{n-1}, \ldots, R_1)$ that is the most suitable representation of cubes for computers [25,35]. For example, according to this representation, the cube $Z = 0x1x0$ is to be represented as $(Z_L, Z_R) = (11011, 01110)$. Namely, in positional-cube notation, each cube is represented by two n-bit strings. But our studies show that each RC may be represented by a single n-bit string that allows us to significantly reduce the space complexity (the amount of memory required) of the logic minimization problem. As it is stated in [16], the time complexity of the POS (named also as CNF) to SOP (named also as DNF) conversion is a square of its space complexity. Consequently, by reducing the space complexity of this problem by a factor of g, at the same time we can reduce its time complexity by a factor of g^2. Our starting point is as follows.

As it has been stated above, $|S_R(P)| = |S_{OFF}| = O(2^n)$, $\forall P \in S_{ON}$. But usually most of the RCs are absorbed by a small number of others RCs in $S_R(P)$ [16,17,25,27,32]. Our experiments performed on a lot of functions show that usually $|S_{RM}(P)| \leq 2.5n$, $\forall P \in S_{ON}$. Namely, usually there are approximately $O(2^n) - 2.5n$ RCs in a $S_R(P)$ that should be removed from it. Hence, in order to simplify the generation of $S_{RM}(P)$ as much as possible, we represent an RC in such a form that allows us to easily detect the redundant RCs. For this aim we use the following relation that may be between any two cubes Z_i and Z_j. The cube Z_i is absorbed by the cube Z_j if:

1) The set of do not care literals present in Z_i is a subset of ones present in Z_j,

2) The same literals appearing in Z_i and Z_j have the same values.

Notice that the second condition is always satisfied, due to Formula (1), for all pairs of RCs. Therefore, it does not need to be checked. Thus, we may use only one *n-bit string* per RC instead of two ones. Such a bit-string (BS) contains 0s and 1s in positions of appearing and do not care literals of an RC represented by this BS, respectively. We call such a BS a literally *Difference Indicator* (DI). According to this approach, the DI for a cube $Z_j \in S_{OFF}$ can be generated by the following simple procedure.

```
**Generate_**$D(P)_j(P, Z_j)$
    $D(P)_j = P^\wedge Z_j$
    *Return* $(D(P)_j)$
```

FIGURE 2. The algorithm converting a Z_j to the corresponding $D(P)_j$

In this algorithm, P is an on-cube for which $D(P)_j$ is generated, $Z_j \in S_{OFF}$ is an off-cube to be converted to the appropriate DI and $^\wedge$ is the bitwise XOR operation sign. We prove the correctness of the algorithm *Generate_D(P)*$_j$ via the procedure *Derive_*Z_j^r that unambiguously converts any $D(P)_j \in S_D(P)$ to the corresponding $Z_j^r \in S_R(P)$.

```
**Derive_**$Z_j^r(P, D(P)_j)$
    $Z_{jL}^r = P \mid \sim D(P)_j$; $Z_{jR}^r = \sim P \mid \sim D(P)_j$
    *Return* $(Z_{jL}^r, Z_{jR}^r)$
```

FIGURE 3. The algorithm converting a $D(P)_j \in S_D(P)$ to the corresponding $Z_j^r \in S_R(P)$

In this algorithm, $\mid$ is the bitwise OR operation sign, and Z_{jL}^r and Z_{jR}^r are the strings of the left and right bits of the reduced cube Z_j^r.

3.2. **Generating the minimal set of difference indicators.** Recall that, according to reduced offset method, in order to generate the set $S_{PI}(P)$, first the set of reduced cubes $S_R(P)$ is generated, and second the set $S_R(P)$ is reduced into the set $S_{RM}(P)$ by removing all redundant cubes from it. It is obvious that the WCTC of $S_R(P)$ to $S_{RM}(P)$ reduction is to be quadratic in the size of $S_R(P)$. Since in the DIs approach $|S_D(P)| = |S_R(P)|$ and $|S_{DM}(P)| = |S_{RM}(P)|$, the WCTC of $S_D(P)$ into $S_{DM}(P)$ reduction is also to be quadratic in the size of $S_D(P)$. Unfortunately, the size of $S_D(P)$ is exponential in n as well as the size of $S_R(P)$. In order to avoid this negativity we form the set $S_{DM}(P)$ directly from the set S_{OFF} without any need to the set $S_D(P)$. For this aim we developed a Formula (2)-based procedure comparing each new generated $D(P)_j$ to those that already in $S_{DM}(P)$. This comparison is continued until $D(P)_j$ is absorbed by any cube in $S_{DM}(P)$ or until all elements of $S_{DM}(P)$ are handled. In order to generate the set $S_{DM}(P)$ completely, the procedure *Generate_*$D(P)_j$ must be applied to all elements of S_{OFF} [3,30,32]. This work is done by the procedure *Generate_*$S_{DM}(P)$ (Figure 4).

Example 3.1. *A single-output function* $f(x_1, x_2, x_3, x_4, x_5)$ *is represented by* $S_{ON} = \{00000, 00010, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110\}$ *and* $S_{OFF} = \{00110, 01010, 10011, 10100, 10101, 10110, 11001\}$. *Find the set* $S_{DM}(P)$ *for the on-cube* $P = 11010 \in S_{ON}$. *The initial content of* $S_{DM}(P)$ *is* $\{1\}^n$. *The* $D(P)_j$ *and the associated content of* $S_{DM}(P)$ *are to be computed by the statements 1 and 2 of*

```
$\boldsymbol{Generate\_S_{DM}(P)}(P, S_{OFF})$
$S_{DM}(P) = \{1\}^n$, $|S_{DM}(P)| = 1$        // Setting the initial content and size of $S_{DM}(P)$
For $j = 1$ to $|S_{OFF}|$ Do
    1. $D(P)_j = P^{\wedge} Z_j$                // The body of the procedure Generate_$D(P)_j$
    2. $S_{DM}(P) = S_{DM}(P) \cup D(P)_j$
    3. Minimize $S_{DM}(P)$ by removing redundant cubes from it by Formula (2)
Return $(S_{DM}(P))$
```

FIGURE 4. The algorithm generating the minimized set of difference indicators

the algorithm Generate_$S_{DM}(P)$*, respectively. Note that the reason of choosing the cube* $P = 11010 \in S_{ON}$ *for this example is that it causes appearance of all possible absorption relations that may be between two cubes. In the solution given below the scratched cubes are those that are redundant (absorbed by other cubes in the same set).*

$D(P)_1 = 11100$; $S_{DM}(P) = \{\cancel{11111}, 11100\}$
$D(P)_2 = 10000$; $S_{DM}(P) = \{\cancel{11100}, 10000\}$
$D(P)_3 = 01001$; $S_{DM}(P) = \{10000, 01001\}$
$D(P)_4 = 01110$; $S_{DM}(P) = \{10000, 01001, 01110\}$
$D(P)_5 = 01111$; $S_{DM}(P) = \{10000, 01001, 01110, \cancel{01111}\}$
$D(P)_6 = 01100$; $S_{DM}(P) = \{10000, 01001, \cancel{01110}, 01100\}$
$D(P)_7 = 00011$; $S_{DM}(P) = \{10000, 01001, 01100, 00011\}$

If we had solved this example by Formula (1), we would generate a set $S_R(P)$ of the size 7, which has to be minimized additionally. As it is seen from Figure 4 and Example 3.1, the time complexity of the algorithm *Generate_*$S_{DM}(P)$ is linear in $|S_{OFF}|$.

3.3. **The expansion of a DI into a set of unit clauses.** In the DI approach, the set $S_{PI}(P)$ is generated by processing the set $S_{DM}(P)$. But to do this, the elements of $S_{DM}(P)$ must be expanded into clauses to be processed by *Nelson's theorem.* The theorem states that the SOP form of a function can be obtained by multiplying out all clauses of the POS form of the same function and removing the redundant implicates from the result [3]. Let us first introduce some definitions:

Definition 3.1. *A clause consisting of a single literal is called a unit clause [13].*

Definition 3.2. *The projection of a DI* $D = d_{n-1}d_{n-2}\ldots d_i\ldots d_0$ *on the coordinate* i *is expressed as follows [16,17,36]:*

$$D(P)[i] = \underbrace{00\ldots0}_{n-i-1} d_i \underbrace{0\ldots00}_{i}, \quad d_i \in \{0,1\} \tag{4}$$

In order to represent a $D(P)_j \in S_{DM}(P)$ as a corresponding clause, we have to decompose it into its own *unit clauses* by the following formula [16].

$$C(P)_j = \{D(P)_j[i] : d_i = 1\}, \quad i = 1, 2, \ldots, n \tag{5}$$

Example 3.2. *Convert the result* $S_{DM}(P) = \{10000, 01001, 01100, 00011\}$ *of Example 3.1 to the sets of unit clauses by using Formulas (4) and (5).*

1. According to Formula (4):
1.1) $D(P)_1[5] = 10000$*;*
1.2) $D(P)_2[1] = 00001$*,* $D(P)_2[4] = 01000$*;*
1.3) $D(P)_3[3] = 00100$*,* $D(P)_3[4] = 01000$*;*
1.4) $D(P)_4[1] = 00001$*,* $D(P)_4[2] = 00010$.
2. According to Formula (5):

2.1) $C(P)_1 = \{D(P)_1[5]\} = \{10000\}$;
2.2) $C(P)_2 = \{D(P)_2[1],\ D(P)_2[4]\} = \{00001, 01000\}$;
2.3) $C(P)_3 = \{D(P)_3[3],\ D(P)_3[4]\} = \{00100, 01000\}$;
2.4) $C(P)_4 = \{D(P)_1[1],\ D(P)_4[2]\} = \{00001, 00010\}$.

The following algorithm implements Formulas (4) and (5) for all $j = 1$ to $|S_{DM}(P)|$.

```
*Generate_C(P)*($S_{DM}(P), n$)
$C(P) = \emptyset$;
For $j = 1$ to $|S_{DM}(P)|$
    $C(P)_j = \emptyset$; $B_0 = D(P)_j$,
    While $B_0 \neq 0$ Do {
        $B_1 = D(P)_j - 1$; $B_0 = B_1$ & $D(P)_j$; $B_2 = B_0 \wedge D(P)_j$; $C(P)_j = C(P)_j \cup B_2$}
    $C(P) = C(P) \cup C(P)_j$
Return ($C(P)$)
```

FIGURE 5. The algorithm generating the sets of unit clauses

As it is easy to see from Figure 5, the time complexity of the procedure *Generate_C(P)* is to be linear in n. But it must be applied to each element of $S_{DM}(P)$ the number of which can be as high as $2.5n$. Hence, the WCTC of application of this procedure is to be quadratic in n, i.e., W(Generate_$C(P)) = O(n^2)$.

3.4. **Generating the set of incompletely specified prime implicants.** We will use the sets $C(P)_1$, $C(P)_2$, …, $C(P)_r$ where $r = |S_{DM}(P)|$, for generating the *incompletely specified* PIs covering the on-cube P. We denote such a PI by $I(P)$. In difference from an exact PI, an $I(P)$ indicates only the positions of the literals present in the PI it represents but does not specify the states (complemented or uncomplemented) of these literals. For example, a $I(P) = 0101$ for the function $f(x_1, x_2, x_{3,}x_4)$ indicates that there is a PI containing literals x_2 and x_4 with no specified states. As will be seen below, these states are clarified by bitwise conjunction of each $I(P)$ with the bitwise negation of P.

Formula (3) states that in order to generate $I(P)s$, it is sufficient to process the sets $C(P)_1$, $C(P)_2$, …, $C(P)_r$ where $r = |S_{DM}(P)|$, by the following iterative formula.

$$I(P) = I(P)\mid C(D)_j, \quad \forall j \in \{1, 2, 3, \ldots, |S_{DM}(P)|\} \tag{6}$$

where instead of $\bar{Z}_j^r$ used in Formula (3) the corresponding $C(D)_j$ is used and the bitwise AND operation is replaced by bitwise OR operation ($\mid$) due to *De Morgan's duality law*. Note that the initial state of $I(P)$ is to be $\{0\}^n$. The bitwise OR operation on $I(P)$ and $C(P)_j$ may be performed as follows:

$$I(P)\mid C(P)_j = \{e_k \mid v_g : e_k \in I(P)) \text{ and } v_g \in C(D)_j\}, \tag{7}$$

where $k_{\max} = |I(P)|$, $g_{\max} = |C(P)_j|$.

In [16,37], it has been shown that while the maximal possible number of PIs of a function of n variables is $SC_{PI} = \binom{n}{n/2}$, the *worst-case space complexity* of a POS to SOP conversion is $SC_{WS} = (n/2)^{\binom{n}{n/2}}$. In the other words, the number of RIs generated in the computation of Formula (7) may be as large as $N_{RI} = SC_{WS} - SC_{PI} = (n/2)^{\binom{n}{n/2}} - \binom{n}{n/2}$. Since $\binom{n}{n/2} << (n/2)^{\binom{n}{n/2}}$ for $n \geq 6$, this expression can be reduced into $N_{RI} = (n/2)^{\binom{n}{n/2}}$. For instance, $N_{RI} = 1,9 \times 10^{42}$ for $n = 8$. This is to say that even a memory with $2^{36} = 10^{10.8}$ address space theoretically may be overflowed during processing of the datasets with $n \geq 7$. In the theory of *switching functions*, to reduce this complexity, the well known *expand* and *remove* approach is used. According to this approach, the clauses are multiplied one by one and after each multiplication the redundant implicants are removed from the result. But since the maximal possible number of implicants occurring in POS to SOP conversion may be as large as $3^n/n$ [27,31], the 64 GB memory available today may sometimes overflow for $n \geq 23$ even at using the expand and remove procedure. To avoid this negativity, the method preventing the occurrence of the redundant implicants proposed in [17,37] can be used. For this aim, Formula (7) is represented in the following recurrent form [37]:

$$\begin{aligned} &I(P)_0 = \{\{0\}^n\} \\ &I(P)_j = I(P)_{j-1} \mathbin{\dot{|}} C(P_j), \quad \text{for all } j = 1, 2, \ldots, |S_{DM}(P)| \end{aligned} \tag{8}$$

As it is seen from Formula (8), $I(P)_j$ is formed by bitwise *Cartesian summing* $I(P)_{j-1}$ with $C(P_j)$. If to look at each of $I(P)_j$, $I(P)_{j-1}$ and $C(P)_j$ as a set, Formula (8) may be reduced into the following one:

$$I(P)_j = \{x \mathbin{\dot{|}} z : x \in I(P)_{j-1} \text{ and } z \in C(P)_j\} \tag{9}$$

In [17] it has been proved that

$$\forall x \in I(P)_{j-1} : x \& C(P)_j \neq \{0\}^n \rightarrow x \mathbin{\dot{|}} C(P)_j = x \tag{10}$$

Formula (10) states that if there is an implicant $x \in I(P)_{j-1}$ such that $x\&C(P)_j \neq 0$ then it will occur as one of implicants generated by the operation $x \mathbin{\dot{|}} C(P)_j$ and will absorb all other implicants generated by this operation. Therefore, all $x \in I(P)_{j-1}$ satisfying the condition $x\&C(P)_j \neq \{0\}^n$ should be considered as a part V_{j1} of the final result without summing them with $C(P)_j$. That is,

$$V_{j1} = \{x \in I(P)_{j-1} | x \& C(P)_j \neq \{0\}^n\} \tag{11}$$

The rest of the set $I(P)_{j-1}$ is obtained as follows:

$$V_{j2} = I(P)_{j-1} - V_{j1} \tag{12}$$

That is, $C(D)_j$ is to be Cartesian-summed with only the set V_{j2}.

$$T_j = V_{j2} \mathbin{\dot{|}} C(P)_j \tag{13}$$

However, a few redundant implicants may occur during the computation of Formula (13). They may be removed by the following formula.

$$T_j^r = T_j - \forall x \in T_j : x \in x \mathbin{\dot{|}} V_{j1} \tag{14}$$

Finally, the set $I(P)_j$ is formed as

$$I(P)_j = T_j^r \cup V_{j1} \tag{15}$$

In [17,37], it is given an algorithm implementing Formula (8) with preventing the occurrence of redundant implicants by using Formulas (11)-(15). In [37], it is stated that this algorithm generates only the essential (irredundant) implicants the total number of which is $R = 0.5 \times \binom{n}{n/2} = O(10^{(0.3xn)-1})$ times less than the total number of all implicants (redundant and irredundant) generated by an algorithm based on *expand and eliminate* principle. The dependency of the order of R on the number of attributes n is given in Table 1. As it is seen from this table, with every increase of n by 10 the efficiency of the algorithm increases roughly by a factor of 10^3.

Table 1. The dependency of the efficiency of the algorithm on the number of attributes

$\boldsymbol{n}$	10	20	30	40	50	60	70	80	90	100	110	120
$\boldsymbol{R}$	10^2	10^5	10^8	10^{11}	10^{14}	10^{17}	10^{20}	10^{23}	10^{26}	10^{29}	10^{32}	10^{35}

The set $I(P)$ of incompletely specified PIs for an on-cube P can be generated by the algorithm *Generate_I*(P) that is an ordered sequence of Formulas (11)-(15).

```
***Generate_I(P)***$(\{C(P)_j\}_{j=1}^{|S_{DM}(P)|}, n)$
$j = 1$, $I(P)_0 = \{0\}^n$
For $j$=1 to $|S_{DM}(P)|$ Do {
    1) $V_{j1} = \{x \in I(P)_{j-1} | x \& C(P)_j \neq \{0\}^n\}$; $V_{j2} = I(P)_{j-1} - V_{j1}$ // *Formulas (11) and (12)*
    2) $T_j = V_{j2}¦C(P)_j$                                // *Formula (13)*
    3) $T_j^r = T_j - \forall x \in T_j$: $x \in x¦V_{j1}$; $I(P)_j = T_j^r \cup V_{j1}\}$     // *Formulas (14) and (15)*
Return $(I(P) = I(P)_j)$
```

FIGURE 6. The algorithm generating the set of incompletely specified prime implicants

Example 3.3. *Generate the set $I(P)$ for results of Example 3.2 by using the algorithm Generate_I(P). The results of Example 3.2 for $P = 11010$ are $C(P)_1 = \{10000\}$; $C(P)_2 = \{00100, 01000\}$; $C(P)_3 = \{00001\}$; $C(P)_4 = \{00001, 00010\}$ and the initial content of $I(P)$ is $I(P)_0 = \{0\}^5$. Since in all iterations the same formulas are computed, they will be referred to only in the first iteration.*

The iteration 1. $I(P)_0 = \{00000\}$; $C(P)_1 = \{10000\}$.

1.1) According to Formulas (11) and (12), $V_{11} = \{x \in I(P)_0 | x \& C(P)_1 \neq \{0\}^n\} = \emptyset$; $V_{12} = I(P)_0 - V_{11} = \{00000\} - \emptyset = \{00000\}$.

1.2) According to Formula (13), $T_1 = V_{12}¦C(P)_1 = \{00000\}¦\{10000\} = \{10000\}$.

1.3) According to Formulas (14) and (15), $T_1^r = \forall x \in T_1$: $x \notin x¦V_{11} = \{10000\}$; $I(P)_1 = T_1^r \cup V_{11} = \{10000\}$.

The iteration 2. $I(P)_1 = \{10000\}$; $C(P)_2 = \{00001, 01000\}$.

2.1) $V_{21} = \{x \in I(P)_1 | x \& C(P)_2 \neq \{0\}^n\} = \emptyset$; $V_{22} = I(P)_1 - V_{21} = \{10000\}$.

2.2) $T_2 = V_{22}¦C(P)_2 = \{10000\}¦\{00001, 01000\} = \{10001, 11000\}$.

2.3) $T_2^r = \forall x \in T_2$: $x \notin x¦V_{21} = \{10001, 11000\}$; $I(P)_2 = T_2^r \cup V_{21} = \{10001, 11000\}$.

The iteration 3. $I(P)_2 = \{10001, 11000\}$; $C(P)_3 = \{00100, 01000\}$.

3.1) $V_{31} = \{x \in I(P)_2 | x \& C(P)_3 \neq \{0\}^n\} = \{11000\}$, $V_{32} = I(P)_2 - V_{31} = \{10001\}$.

3.2) $T_3 = V_{32}¦C(P)_3 = \{10001\}¦\{00100, 01000\} = \{10101, 11001\}$.

3.3) $T_3^r = \forall x \in T_3$: $x \notin x¦V_{31} = \{10101,$ ~~11001~~$\}$; $I(P)_3 = T_3^r \cup V_{31} = \{10101, 11000\}$.

The iteration 4. $I(P)_3 = \{10101, 11000\}$; $C(P)_4 = \{00001, 00010\}$.

4.1) $V_{41} = \{x \in I(P)_3 | x \& C(P)_4 \neq \{0\}^n\} = \{10101\}$, $V_{42} = I(P)_3 - V_{41} = \{11000\}$.

4.2) $T_4 = V_{42}¦C(P)_4 = \{11000\}¦\{00001, 00010\} = \{11001, 11010\}$.

4.3) $T_4^r = \forall x \in T_4 : x \notin x¦V_{41} = \{11001, 11010\}$; $I(P)_4 = T_4^r \cup V_{41} = \{11001, 11010, 101$ $01\}$.

If we had solved this example by the conventional POS to SOP conversion, we would generate a set $I(P)_4$ of the size 4, which has to be minimized additionally. As it is seen from Figure 6 and Example 3.3, the time complexity of the algorithm *Generate_I*(P) is linear in $|S_{DM}(P)|$, while the algorithm solving this problem by conventional POS to SOP conversion is NP hard [9,11,14].

3.5. Converting a set of incompletely specified PIs to the set of exact PIs and selecting the $\boldsymbol{E(P)}$.

In order to convert an incompletely specified PI $G = g_n g_{n-1} \ldots g_1 \in I(P)$ to the corresponding exact PI $U = u_n u_{n-1} \ldots u_1$ represented in the positional cube notation, it is sufficient to perform the following operation.

$$\text{If } g_i = 0 \text{ then } u_i = x, \text{ else } u_i = p_i, \text{ for all } i = 1, 2, \ldots, n \qquad (16)$$

where p_i is the ith bit of the on-cube P for which the PI U is generated. All the PIs for the given on-cube P can be generated by the following algorithm.

```
$\boldsymbol{Generate\_S_{PI}(P)}(P, I(P))$
$S_{PI}(P) = \emptyset$
$For\ j = 1\ to\ |I(P)|$ {
    $Select\ G_j \in I(P);\ If\ g_i = 0\ then\ u_i = x,\ else\ u_i = p_i,\ for\ all\ i = 1, 2, \ldots, n$
    $S_{PI}(P) \cup U_j;\ j = j + 1$ }    // $g_i$, $u_i$ and $p_i$ are the ith bits of G, U and P, respectively
$Return\ (S_{PI}(P))$
```

FIGURE 7. The algorithm converting a set $I(P)$ to the corresponding set $S_{PI}(P)$

Example 3.4. *Using the algorithm Generate_*$S_{PI}(P)$ *convert the result* $I(P) = \{11001, 11\ 010, 10101\}$ *of Example 3.3 to the set of exact PIs for* $P = 11010$.

In the conversions given below, ω *is used as the operator realizing the* IF *statement of the algorithm Generate_*$S_{PI}(P)$.

1) ω: $(I(P)_1, P) = \omega : (11001, 11010) \rightarrow 11xx0$,

2) ω: $(I(P)_2, P) = \omega : (11010, 11010) \rightarrow 11x1x$;

3) ω: $(I(P)_1, P) = \omega : (10101 \otimes 11010) = 1x0x0$; $S_{PI}(P) = \{11xx0, 11x1x, 1x0x0\}$.

There are many heuristic criteria in the related literature for selecting the $E(P)$ from the set $S_{PI}(P)$ [21-23,26,27] among which the most popular is that selecting as the $E(P)$ a PI covering the maximum number of on-cubes from the yet uncovered part of the set S_{ON}. The comparison of the PIs from the set $S_{PI}(P)$ with the cubes from the set S_{ON} shows that the PIs $11xx0$, $11x1x$ and $1x0x0$ cover the subsets of cubes $\{11000, 11010, 11110\}$, $\{11010, 11110\}$ and $\{10000, 10010, 11000\}$, respectively. Therefore, as the $E(P)$, either $11xx0$ or $1x0x0$ should be preferred.

3.6. The main procedure.

The main procedure *Generate_*$S_{PI}(P)$ generating all PIs for the given on-cube $P \in S_{ON}$ is formed by sequencing the procedures given in this section.

```
$\boldsymbol{Generate\_S_{PI(P)}}\ (P, S_{OFF})$
  { $Generate\_S_{DM}(P)(P, S_{OFF})$        // Output: $S_{DM}(P)$
    $Generate\_C(P)(S_{DM}(P), n)$        // Output: $C(P)$
    $Generate\_I(P)(M(P))$               // Output: $I(P)$
    $Generate\_S_{PI}(P)(P, I(P))$        // Output: $S_{PI}(P)$
  }
```

4. The Experimental Results.

A lot of experiments were done to evaluate the runtime and quality of the results of the algorithm *Generate_*$S_{PI}(P)$ realizing the proposed method. The computer used was a PC with Intel® Core 2 Quad CPU Q8400 2.66 GHz and 4096 MB RAM. The quality of the results was measured by numbers of PIs forming the minimized functions. Since currently the best logic minimization method is that named as Espresso, we compared the results generated by our method with those generated by Espresso for the same functions. In the experiments, a set of 45 standard single-output MCNC benchmarks was solved by *Espresso-Exact*, *Espresso-Signature* and by the proposed method. Since the last two methods generated the same results and took approximately the same time, here we refer to them simply as *Espresso*. In the experiments, we did not applied any pre-ordering to the *onsets* and used the simplest $E(P)$ identification rule that selects as $E(P)$ such a PI which covers more on-cubes from the set S_{ON} than other ones. The results of processing of 45 benchmarks are shown in Table 2. As seen from this table, for 16 benchmarks (group G1) the results generated by proposed

Table 2. The results of experiments

Benchmarks		n/ $\lvert S_{ON}\rvert$/ $\lvert S_{OFF}\rvert$	Number of result cubes		Time elapsed (ms)		T_E/T_{OM}
			Espresso	Our meth.	Espresso T_E	Our meth. T_{OM}	
G1	bca	26/15/13	6	1	59,39	41,82	1,42
	t10	10/134/189	49	45	55,23	18,62	2,97
	br1	12/25/8	8	4	59,82	14,08	4,25
	br11	12/28/5	8	3	57,51	26,46	2,17
	br2	12/29/6	6	2	58,03	14,42	4,02
	den	18/18/2	14	4	58,84	66,16	0,89
	exp	8/18 /52	4	3	56,53	13,68	4,13
	exp1	8/24/47	6	3	58,23	15,12	3,85
	inc	7/12/22	7	6	57,33	14,99	3,82
	max4	9/37/8	36	6	78,56	16,62	4,73
	min	9/83/51	29	6	57,64	14,64	3,94
	p82	5/11/13	5	4	58,68	15,16	3,87
	pdc	16/29/1891	11	4	59,69	14,58	4,09
	prom2	9/142/145	8	7	58,79	14,81	3,97
	spla	16/67/2036	38	29	58,17	19,26	3,02
	sqn	7/48/48	12	8	58,28	14,48	4,02
G2	m3	8/98/30	14	16	62,67	15,30	4,10
	m4	8/223/26	23	24	57,60	29,40	1,96
G3	apex4	9/4/434	4	4	59,88	14,50	4,13
	check	4/4/9	1	1	55,39	14,32	3,87
	check1	4/4/8	1	1	56,37	14,60	3,86
	check2	4/4/6	1	1	58,61	14,13	4,15
	check3	4/8/5	2	2	58,06	14,19	4,09
	dist	8/53/203	12	12	59,17	14,73	4,02
	ex5	8/33/223	2	2	61,29	14,61	4,20
	exps	8/65/131	20	20	58,93	13,99	4,21
	f51m	8/128/128	23	23	62,52	14,28	4,38
	linrom	7/65/63	24	24	59,88	14,86	4,03
	m	6/27/5	4	4	58,61	14,01	4,18
	m5	6/27/5	4	4	59,61	13,91	4,28
	max1024	10/516/508	4	4	59,07	14,47	4,08
	max128	7/29/99	8	8	61,44	14,29	4,30
	max3	7/12/116	7	7	59,38	14,50	4,10
	max512	9/358/254	10	10	59,85	13,76	4,35
	mlp4	8/32/224	9	9	61,91	14,43	4,29
	new2	6/3/4	2	2	61,98	14,79	4,19
	poperom	6/56/8	7	7	60,50	14,53	4,16
	rd84	8/120/136	84	84	60,25	13,72	4,39
	root	8/15/241	4	4	62,34	14,15	4,41
	sqr	6/18/46	2	2	60,13	14,39	4,18
	squar	5/9/23	2	2	59,38	14,77	4,02
	t3	12/27/121	6	6	60,78	14,09	4,32
	wim	4/9/1	4	4	60,14	14,31	4,20
	z5xp1	7/25/103	3	3	58,84	13,79	4,27
	e	8/65/128	20	20	61,54	14,17	4,34

method are significantly better than those obtained by Espresso. But there are 2 benchmarks $m3$ and $m4$ (group G2) for which our method generated a little worse results than Espresso. For all remaining 27 benchmarks (group G3) both methods obtained the same results. In general, our method generated better, equivalent and worse result for 36%, 60% and 4% of the benchmarks, respectively. Our method has proved faster by a factor of 3.55 for 44 benchmarks on average and a little slower for only one benchmark (den).

5. Conclusion.

In this study we propose a new approach for simultaneously generating all PIs covering a given cube of a function. This approach is based on the reduced offset method proposed in [3]. The main property of our approach is that we represent each reduced off-cube by using only a single n-bit string instead of a $2n$-bit string used in all other methods. Such a representation of the reduced off-cubes allows us to reduce the memory space required for storage the sets of such cubes by a factor of 2. Since the proposed method generates all PIs covering the given on-cube simultaneously the algorithm implementing this method works approximately 3.55 times faster, on average than *Espresso*. The quality of the results generated by our method can be significantly improved by using some convenient pre-ordering the onsets and more sophisticated $E(P)$ identification rules given in [21-24,26,38]. Our approach can also be applied to minimization of multiple-output functions by taking into consideration the well known relations existing between multiple-output PIs [5,27,34,38]. Currently, we work on the logic minimization of a function by partitioning its bit-based conjunctive normal form by the method proposed in [16]. The preliminary estimations show that this approach would increase the efficiency of the algorithm on a large scale.

Acknowledgement. This work is supported by Selçuk and Mevlana Universities Scientific Research Projects Coordinatorships, Konya, Turkey.

REFERENCES

[1] A. Mishchenco and T. Sasao, Large-scale SOP minimization using decomposition and functional properties, *DAC*, pp.149-154, 2003.

[2] T. Sasao, Worst and best irredundant sum-of-product expressions, *IEEE Trans. on Comput.*, vol.50, pp.935-947, 2001.

[3] A. Malik, R. K. Brayton, A. R. Newton and A. Singiovanni-Vincentelli, Reduced offsets for minimization of binary-valued functions, *IEEE Trans. Comput.*, vol.42, pp.1325-1342, 1993.

[4] R. A. Bergamaschi, D. Brand, L. Stok, M. Berkelaar and S. Prakash, Efficient use of large don't cares in high-level and logic synthesis, *International Con. on Comp.-Aided Design*, pp.272-278, 1995.

[5] P. Fiser and J. Hlavicka, Boom – A heuristic boolean minimizer, *Journal of Computing and Informatics*, vol.22, pp.1001-1033, 2003.

[6] N. Allahverdi, S. Kahramanli and K. Erciyeş, A fault tolerant routing algorithm based on cube algebra for hypercube system, *Journal of System Architecture*, vol.46, pp.201-205, 2000.

[7] P. Dündar and E. Kılıç, Finding a fault-tolerant routing on neighbor – Faulty hypercube, *International Journal of Computer Mathematics*, vol.81, pp.1043-1049, 2004.

[8] S. Güneş, N. Yılmaz and N. Allahverdi, A fault – Tolerant multicast routing algorithm based on cube algebra for hypercube networks, *The Arabian Journal for Science and Engineering*, vol.28, pp.95-103, 2003.

[9] R. W. Swiniarski and A. Skowron, Rough set methods in feature selection and recognition, *Pattern Recognition Letters*, vol.24, pp.833-849, 2003.

[10] Y. Matsumoto and J. Watada, Knowledge acquisition from time series data through rough sets analysis, *International Journal of Innovative Computing, Information and Control*, vol.5, no.12(B), pp.4885-4897, 2009.

[11] J. Komorowski, L. Polkowski and A. Skowron, *Rough set: A tutorial*, http://folli.loria.fr/cds/1999/library/pdf/skowron.pdf, 1999.

[12] M. Inuiguchi and M. Tsurumi, Measures based on upper approximations of rough sets for analysis of attribute importance and interaction, *International Journal of Innovative Computing, Information and Control*, vol.2, no.1, pp.1-12, 2006.
[13] R. Jensen and Q. Shen, *Rough Set Based Feature Selection: A Review*, htp//cadair.aber.ac.uk/dspace/ handle/2160/490, 2007.
[14] A. Skowron, The rough sets theory and evidence theory, *Fundamenta Informaticae*, vol.13, pp.245-262, 1990.
[15] W. Chen, S. Tseng and T. Hong, An efficient bit-based feature selection method, *Expert Systems with Applications*, vol.34, no.4, pp.2858-2869, 2008.
[16] S. Kahramanli, M. Hacibeyoglu and A. Arslan, Attribute reduction by partitioning the minimized discernibility function, *International Journal of Innovative Computing, Information and Control*, vol.7, no.5(A), pp.2167-2186, 2011.
[17] S. Kahramanli, M. Hacibeyoglu and A. Arslan, A boolean function approach to feature selection in consistent decision information systems, *Expert Systems with Applications*, vol.38, no.7, pp.8229-8239, 2011.
[18] H. Sakai and M. Nakata, On rough sets based rule generation from tables, *International Journal of Innovative Computing, Information and Control*, vol.2, no.1, pp.13-31, 2006.
[19] S. J. Hong, R-MINI: An iterative approach for generating minimal rules from examples, *IEEE Trans. Knowledge and Data Eng.*, vol.9, pp.709-717, 1997.
[20] M. Muselli and D. Liberati, Binary rule generation via hamming clustering, *IEEE Trans. on Knowledge and Data Engineering*, vol.14, no.6, 2002.
[21] G. Promper and J. Armstrong, Representation of multivalued functions using the direct cover method, *IEEE Trans. Comp.*, vol.30, no.9, pp.674-679, 1981.
[22] G. W. Dueck and D. M. Miller, A direct cover MUL minimization using the truncated sum, *Proc. of the 17th Int. Sem. MV Logic*, pp.221-227, 1987.
[23] P. W. Besslich, Heuristic minimization of MUL functions: A direct cover approach, *IEEE Trans. Comp.*, pp.134-144, 1986.
[24] O. Coudert, Two level logic minimization: An overview, *Integration the VLSI Journal*, vol.17, pp.97-140, 1994.
[25] D. M. Giovanni, *Synthesis and Optimization of Digital Circuits*, Mccraw-Hill, New York, 1994.
[26] P. P. Tirumalai and J. T. Butler, Minimization algorithms for multiple-valued PLAs, *IEEE Trans. on Comput.*, vol.40, pp.167-177, 1991.
[27] R. K. Brayton, G. D. Hachtel, C. T. Mcmullen and A. Singiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic, Boston, 1984.
[28] C. Umans, T. Villa and A. Sangiovanni-Vincentelli, Complexity of two-level logic minimization, *IEEE Tran. on Comp.-Aided Design of Integrated Circuits and Sys.*, vol.25, pp.1230-1246, 2006.
[29] C. Umans, The minimum equivalent DNF problem and shortest implicants, *Journal of Computer and System Sciences*, vol.63, pp.597-611, 2001.
[30] Ş. Kahramanlı, S .Güneş, S. Şahin and F. Başçiftçi, A new method based on cube algebra for the simplification of logic functions, *The Arabian Journal for Science and Engineering*, vol.32, pp.101-114, 2007.
[31] A. Malik, R. K. Brayton, A. R. Newton and A. Singiovanni-Vincentelli, Two-level minimization of multivalued functions with large offsets, *IEEE Trans. on Computer-Aided Design*, vol.10, pp.413-424, 1991.
[32] S. Kahramanli and F. Başçiftçi, Boolean functions simplification algorithm of O(N) complexity, *Journal of Mathematical and Computational Applications*, vol.8, pp.271-278, 2003.
[33] D. L. Dietmeyer, *Logical Design of Digital Systems*, 2nd Edition, Boston, 1978.
[34] R. E. Miller, *Switching Theory*, Moscow, Mir, 1970 (in Russian).
[35] J. F. Wakerly, *Digital Design Principles and Practices*, http://www.ddpp.com/DDPP3_mkt/c04samp1.pdf.
[36] D. Brand, R. A. Bergamaschi and L. Stok, Don't cares in synthesis: Theoretical pitfalls and practical solutions, *IEEE Tran. on Comp.-Aided Design of Integrated Circuits and Sys.*, vol.17, pp.285-304, 1988.
[37] M. Hacıbeyoğlu, F. Başçiftçi and Ş. Kahramanlı, A logic method for efficient reduction of the space complexity of the attribute reduction problem, *Turkish Journal of Electrical Engineering & Computer Sciences*, vol.19, no.4, pp.643-656, 2011.
[38] R. Sharon and T. Rhyne, An algorithm for identifying and selecting the PI's of a multiple-output boolean function, *IEEE Trans. Computer-Aided Design*, vol.7, pp.1215-1218, 1988.