Ain Shams Engineering Journal 12 (2021) 729–737

Contents lists available at ScienceDirect

Ain Shams Engineering Journal

journal homepage: www.sciencedirect.com

Electrical Engineering

# Smart minterm ordering and accumulation approach for insignificant function minimization

Fatih Başçiftçi [a], Hakan Akar [b,*]

[a] Department of Computer Engineering, Technology Faculty, Selçuk University, Turkey
[b] Institude of Graduate Education, Konya Technical University, Konya, TURKEY

ARTICLE INFO

*Article history:*
Received 16 March 2019
Revised 23 December 2019
Accepted 26 April 2020
Available online 19 June 2020

*Keywords:*
Insignificant logic functions
Logic
Logic minimization
Logic synthesis
Two-level logic simplification
Direct cover
Minterm ordering

ABSTRACT

Previously finding prime implicants based on off-cubes was explored as an approach to minimize insignificant logic functions which include minterms both easy and difficult to cover. Off-cube based function minimization falls short in certain functions and may not yield/produce the accurate results. In this study, a new method of minimizing insignificant logic functions that includes smart minterm ordering according to their contiguity is proposed. In the proposed method, minterms are ordered from easy to difficult in terms of covering. This kind of a smart ordering helps minimization algorithms to quickly cover easy minterms and decrease the complexity of remaining function. A new accumulation approach is also developed and employed for the minimization of complicated functions. The use of the new accumulation approach in the study made it possible to reach more precise results. When it is impossible to determine exact prime implicants, the developed algorithm accumulates minterm and its corresponding implicants in a suspended state (SS) and reconsiders covering them later. Both the theory and practice of accumulation approach for the minimization of minterms is presented. Standard MCNC benchmarks are simplified with both the proposed method and with the two level simplification program ESPRESSO. The comparative analysis of the results revealed that the proposed method finds exact minimum results using less time and memory than ESPRESSO.


## 1. Introduction

Digital circuits consist of logic gates such as AND, OR, and their derivatives. These digital circuits are represented in SOP formed logic functions. One of the simplification methods for digital circuits is two level simplification. Better simplification of logic functions enables smaller and faster integrated circuits. Two level simplification of logic functions is a classical approach but still very useful in circuit design synthesis, feature reduction, control systems, biological circuits, and memristive devices [1–6]. It is also useful in hierarchic design of networks for optimization [7,8]. Moreover, SOP simplification known as two level simplification, is also very handy for optimization of test generators [8,9]. In addition, obtaining prime cubes including source and destination nodes and finding shortcut between nodes in hyper cube designed systems may have benefits of logic simplification [10]. The fact that using the solution of an NP-Hard problem (logic minimization) to solve another NP-Hard problems is possible, lies on the basis of this generalization. As some common optimization techniques are similar to logic minimization, improvements in one of them may benefit others [11,12]. As it can be used in many recent areas, simplification of logic functions is a quite old subject but not dead.

One of the most convenient methods for computer implementation is Quine-McCluskey method [13,14]. But it is a time and memory consuming method for many minterms [15]. Since exact simplification of SOP functions has exponential nature, even super algorithms have difficulties for huge functions having more than one hundred product terms [6]. Thus, heuristic simplification is used for most of the practical applications [6,16]. Heuristic algorithms are slower than exact ones only for functions with many product terms. Widely used acceleration technique for logic simplification is basically performing only one simplification loop. In these cases, simplification quality is low and runtime problem still remains unsolved [6]. There are many heuristic algorithms but

* Corresponding author at: Mollayusuf mh. 1448 sk. No:7E/2 Konyaaltı, Antalya, Turkey.
*E-mail address:* maviakar@gmail.com (H. Akar).
Peer review under responsibility of Ain Shams University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.asej.2020.04.003
2090-4479/

none of these algorithms is always better than others for all classes of functions [17]. Most of the direct-covering heuristic simplification approaches generate prime implicants (PIs) covering Selected Minterm (SM) by implicant expansion technique [18]. The function $f$ consists of *OFF*, *ON* and *DC* minterm groups. $G_{OFF}$, $G_{ON}$ *and* $G_{DC}$ stand for minterm groups which determine function $f$ equal to 0, 1 and unspecified, respectively.

Simplification methods using Direct-Cover Principle (DCP) totally masks $G_{ON}$ by repeating two steps: Generate PIs and selects essential ones among them [19,20]. One of the major obstacles in two level logic simplification is Backtracking Technique (BT) used in second step [21]. Backtracking is a systematic method for finding all possible subsets. Backtracking constructs a tree of partial solutions, where each branch represents a partial solution [22]. As it considers every possible combination, it is only preferable for compelling situations [23]. When SM could be covered by multiple PIs, it takes pretty much time to process a decision rule. This step of the algorithms is important for time and memory consumption and simplification level of the result.

There could be two different approaches to overcome these difficulties. We can order minterms by their adjacency factor before processing them. It's easy to process isolated minterms, because there is only one or a couple of potential implicants covering these minterms. It will be easy to implement decision rule or even no need to use it. After simplifying isolated minterms, decision process for remaining function will be easier. This approach is proposed and discussed in section 3.

Another way to get better results is improving current algorithms and methods. Decision rule is explained by R.E. Miller [24] and modified by researchers [25–31]. Previously finding prime implicants based on off-cubes was explored as an approach to minimize insignificant logic functions [31]. This paper presents new EPI decision rules and algorithms for accumulation approach. Processing minimization algorithm for many PIs consumes much time and increase the probability of finding unnecessary implicants. Thus, we proposed to process easy minterms and accumulate complicated minterms which has many potential PIs.

The rest of the paper is arranged as follows. In section 2, we summarized abbreviations and notations used in this paper. Section 3 briefly explains related work including ESPRESSO. Theory and practice of a new smart minterm ordering method is presented in section 4. Proposed accumulation approach for function minimization, choosing essential prime implicants with accumulation approach and processing accumulated states are presented in section 5. Implementation of proposed method is described in section 6. In section 7, we explained some of our experimental results for proposed method and famous ESPRESSO program and compare results of two algorithms. Finally we conclude the study in section 8.

## 2. Theoretical background

In this section, we describe some notations and abbreviations used on this paper. Boolean values {0, 1} set is denoted as $\mathbb{B}$ and incomplete Boolean values {0, 1, *d*} set is denoted as $\mathbb{B}_+$. "*d*" is an insignificant value like Don't Care(DC) (see Figure 1). A *completely specified logic function* is a mapping of $\mathcal{F}$: $\mathbb{B}^i \rightarrow \mathbb{B}$ and an *insignificant logic function* is a mapping of $\mathcal{F}$: $\mathbb{B}_+ i \rightarrow \mathbb{B}$. "*i*" represents the input values as *i-bit binary variable* in the form $x_1x_2x_3...xi$. Expression $\mathcal{F}(x) \equiv \mathcal{F}(x_1x_2x_3...xi)$ denote the logic function having $i$ number of *literals*. A *literal* is a *Boolean variable* or its complement. *On-set* and *off-set* of a *completely specified logic function* can be defined as $\mathcal{F}^{on} = \{\in \mathbb{B}i \mid \mathcal{F}(x) = 1\}$ and $\mathcal{F}^{off} = \{x \in \mathbb{B}i \mid \mathcal{F}(x) = 0\}$ respectively. In addition to $\mathcal{F}^{on}$ and $\mathcal{F}^{off}$, *incompletely specified logic functions* also have *dc-set* defined as $\mathcal{F}^{dc} = \{\in \mathbb{B}^i \mid \mathcal{F}(x) = d\}$.

*Prime Implicant* is defined as PI and *Essential Prime Implicant* is defined as EPI. SM, PM and AM are abbreviations for *Selected Minterm*, *Processing Minterm* and *Accumulated Minterm*. *Inclusion Function* is represented as IF and Backtracking Technique is represented as BT. FPP stands for *Fragmented Processing Procedure* and SS stands for *Suspended State*. DCP represents *Direct Covering Principle* and SOP represents *Sum of Product* form.

A relation on *mutual condition* and *cube subtract operation* is represented by ↔ and # symbols respectively. $G_{ONi}$ represents uncovered *on-set minterm group* where $G_{OFF}$, $G_{DC}$ represents *off-set minterm group* and *dc-set minterm group*, respectively. $X$ defines *on-set minterm* selected to be masked and $PI_i(x)$ defines a PI masking that *minterm X*. All PIs masking the *minterm X* is defined as $G_{PI}(x)$. EPI masking the *minterm X* is represented as $EPI(x)$ and EPIs series is defined as $G_{EPI}$. Series of SS is denoted as $G_{SS}$ and size of a series is $|G|$. Finally, an inclusion function is defined as $IF_i$, a term of $IF_i$ is defined as $L_{im}$ and number of neighbours of the SM $X$ is defined as $N(x)$.

## 3. Related work

Two level logic minimization, one of the most famous minimization approach, is presented by Quine [13]. Then McCluskey developed this algorithm for a fully automated minimization version [14]. This is why two level logic minimization is generally called Quine-McCluskey procedure. At first, this procedure generates all possible PIs using on-set of a function, secondly finds minimum EPIs that covers all on-minterms. Generating PIs is relatively easy and can be efficiently done [32]. But finding the minimum subset of EPIs is known as an NP-Complete problem. As a result, using this technique is not practical for large functions.

A group of scientists developed heuristic algorithms for efficiently minimize logic functions at IBM laboratories. Development of this algorithm, Espresso, is managed by Robert Brayton. Final Espresso version 2.3 is released in 1988. Although it has been a long time after it's developed, it is one of the most famous and creative solution for logic function simplification. Espresso has very effective method and use advanced heuristics for two level minimization. Espresso iterates these four operations until no improvement:

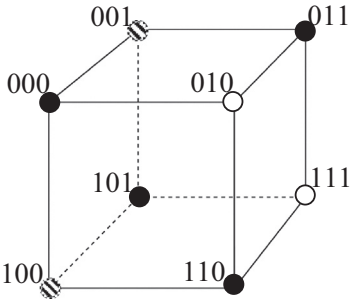
1. **Expand** each implicant to Prime
2. Extract **Essential Primes**
3. Find simplest **Irredundant Cover**
4. **Reduce** to simplest Essential Implicant

Instead of dividing function into cubes, this algoritm produces ON, OFF and DC cubes. This algorithm does not guarantee the simplest solution but runs really fast for complicated files. Espresso algorithm is given in Fig. 2.

Espresso is more efficient than other methods in terms of memory usage and computation times. There is no restriction to the number of variables and output functions. This algorithm can minimize tens of functions with tens of variables in a reasonable time. In addition to original Espresso software, Minilog and Logic Friday software are also using Espresso algorithms.

## 4. Minterm ordering

Logic functions consist of minterms. If we change the sequence of minterms, minimization algorithms may find more or less simplified solutions. Minimization quality is effected by the sequence of minterms. In this section, we investigate random minterm selection and present smart minterm ordering by their contiguity.



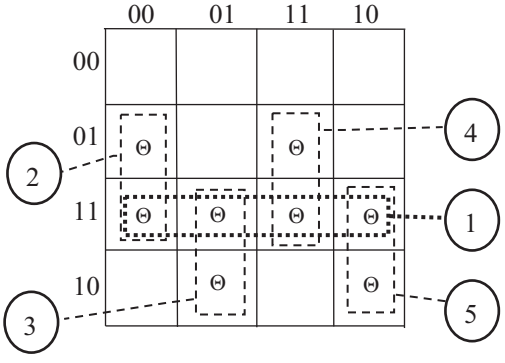

| Input | | | Output |
|---|---|---|---|
| $x_1$ | $x_2$ | $x_3$ | $y$ |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | *dc* |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | *dc* |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**Fig. 1.** 3 dimensional space representation of a Boolean function and its truth table.

Algorithm ESPRESSO

(1) **Inputs**: ON is on minterm series, DC is dc minterm series, OFF is off minterm series
(2) **Outputs**: ON as minimized function
(3) Get the complement of ON and DC to find OFF
(4) Expand ON using OFF
(5) Perform irredundant cover with ON and DC
(6) Find essential primes with ON and DC
(7) Remove essential primes from ON
(8) Add essential primes to D
(9) WHILE Cost(ON) keeps decreasing DO
(10) Perform reduction using ON and DC
(11) Perform expansion for ON using OFF
(12) Perform irredundant cover using ON and DC
(13) ENDWHILE;
(14) Remove essential primes from D
(15) Add essential primes to ON
(16) Make ON sparse using ON, OFF, DC
(17) RETURN ON
(18) END

**Fig. 2.** Pseudo-code of Espresso Algorithm.




| $G_{min}$ | G(*x*) | *PI(x)*'s |
|---|---|---|
| 0000 | 0 | |
| 0001 | 0 | |
| 0010 | 0 | |
| 0011 | 0 | |
| 0100 | 1 | 2 |
| 0101 | 0 | |
| 0110 | 0 | |
| 0111 | 1 | 4 |
| 1000 | 0 | |
| 1001 | 1 | 3 |
| 1010 | 1 | 5 |
| 1011 | 0 | |
| 1100 | 1 | 2,1 |
| 1101 | 1 | 3,1 |
| 1110 | 1 | 5,1 |
| 1111 | 1 | 4,1 |

**Fig. 3.** Karnaugh map and truth table of function *G*.

Proposed minterm ordering method do not change minterms or functions, this method only change minterms' sequence.

### A. The Random Minterm Choose Principle

Key point for using FPP in DCP based simplification approaches is that an *EPI(x)* is selected at that moment when $G_{PI}(x)$ has been defined. Here is the problem. Assume that a series of $G_{PI}(x) = \{PI(x)\}_{i=1}^{K}$ for SM *X* has been generated. Applying FPP rule, then

$$PI_k(x) > PI_i(x) \leftrightarrow \left|G_{ON} \bigcap PI_k(x)\right| > \left|G_{ON} \bigcap PI_i(x)\right| \quad forall \ \ i \neq k \qquad (1)$$

This approach selects a *PI(x)* which masks more On-minterms than other PIs. On the contrary, superior *PI(x)* is the one which masks all On-minterms with all other *PI(x)*s. Thus, (1) cannot find the best solutions in all cases. Example 1 reveals how (1) miscalculates EPIs and may cause worst (non exact) outcomes [28].

**Example 1.** *Simplify function below by using random minterm choose approach.*

$G_{ON}$ = {0100, 0111, 1001, 1010, 1100, 1101, 1110, 1111}
$G_{OFF}$ = {0000, 0001, 0010, 0011, 0101, 0110, 1000, 1011}
Consider choosing the *ON*-minterm 1100 as first SM, then the following prime implicants $PI_1$ (1100) = 11xx and $PI_2$ (1100) = x100 will be obtained (see Figure 3). Therefore,

C(11xx) = $G_{ON}$ ∩ 11xx = {0100, 0111, 1100, 1101, 1111, 1110, 1001, 1010} ∩ 11xx
= {1100, 1101, 1111, 1110}
C(x100) = $G_{ON}$ ∩ x100 = {0100, 0111, 1100, 1101, 1111, 1110, 1001, 1010} ∩ x100
= {1100, 0100}

Since |C(x100)| = 2 and |C(11xx)| = 4, $PI_1$(1100) = 11xx is the $EPI_1$.
Therefore: $G_{EPI}$ = {11xx};
$G_{ON}$ = $G_{ON}$ # 11xx = {0100, 0111, 1001, 1010}.
As a result, *DC* minterm series is defined as $G_{DC}$ = {1100, 1101, 1111, 1110} consisting of 1 s masked by the $EPI_1$. Thus, the rest of the *ON* minterm series to be simplified is defined by $G_{ON}$ = {0100, 0111, 1001, 1010} and $G_{DC}$ = {1100, 1101, 1111, 1110}. Fig. 4 represents the current state of the function *G*. *DC* and *ON* minterms are coded by "DC" and "Θ" respectively.
Fig. 4 reveals that, rest of *ON*-minterm series are isolated minterms which enables us to easily define all *EPIs*. $EPI_2$ = x100, $EPI_3$ = 1-x01, $EPI_4$ = x111, $EPI_5$ = 1x10. As a result,

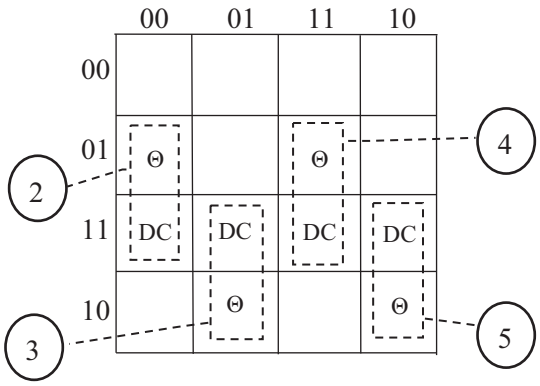



**Fig. 4.** Karnaugh map for current state of function *G*.

$G_{ON}$ = $G_{ON}$ # {$EPI_2$, $EPI_3$, $EPI_4$, $EPI_5$}
= {0100, 1001, 0111, 1010} # {x100, 1x01, x111, 1x10} = ∅
$G_{EPI}$ = {11xx, x100, 1x01, x111, 1x10}

Consider that choosing first SM among 1101, 1111 and 1110 would result in same solution. Because these minterms also leads us to the same unnecessary implicant. On the other hand, Petrick BT lead us to a different solution [5,29].
Results formed in the Table 1 shows that the $PI_1$(1011) = 11xx, formerly determined as $EPI_1$ above, is not an EPI at all. Hence, the exact solution should be as follows
$G_{EPI}$ = {x100, 1x01, x111, 1x10}.
Actually, solution for function *G* could be found in 4! = 24 iterations. We need to give priority to four most isolated SM which are 0100, 0111, 1001 and 1010. If non-isolated minterms are picked, solution set will lead to worst case 4 × 4! = 96. In other words, there will be 96–24 = 72 unnecessary iterations which means 96/24 = 4 times slower runtime (75% loss).

### B. Smart Minterm Ordering by Their Contiguity

Priority of *ON* Minterms which will be masked, is very important to get best results. Section 3 briefly explains worst case. We proposed an improved DCP approach that repeats the following procedure until $G_{ON}$ =∅.

(1) For each element of GON calculate the weight equal to the number of its neighbours
(2) Chose as the SM X that one which with the smallest weight and generate GPI(x)
(3) Choose an EPI among the PI(X)s
(4) Subtract the EPI from the rest (last state) of GON

**Example 2.** *The function and its initial Karnaugh Map are stated in Example* 1. *Simplify this function with minterm ordering based on adjacency factors.*

As seen from the Figure 3; M(0100) = M(0111) = M(1001) = M(1010) = 1; M(1100) = M(1101) = M(1111) = M(1110) = 3. Consequentially, the group of minterms with the minimal weight is $G_F$ = {0100, 1001, 0111, 1010}

(1) Let as first SM to choose the minterm 0100 ∈ GF. There the EPI1 = x100 covering the SM 0100. Consequentially, GEPI = {x100}; GON = GON # x100 = {0100, 0111, 1100, 1101, 1111, 1110, 1001, 1010} # x100 = {0111, 1101, 1111, 1110, 1001, 1010}. The state of Karnaugh map after this covering is shown in Fig. 5a.
(2) Let as second SM to choose 1001 ∈ $G_F$. There the $EPI_2$ = 1x01 covering the minterm SM 1001. Consequentially, $G_{EPI}$ = {x100, 1x01}. $G_{ON}$ = $G_{ON}$ # 1x01 = {0111, 1101, 1111, 1110, 1001, 1010} # 1x01 = {0111, 1111, 1110, 1010}. The state of Karnaugh map after this covering is shown in Fig. 5b.
(3) Let as third SM to choose the minterm 0111 ∈ $G_F$. There the $EPI_3$ = x111 covering the SM 1111. Consequentially, $G_{EPI}$ = {x100, 1x01, x111}. $G_{ON}$ = $G_{ON}$ # x111 = {0111, 1111, 1110, 1010} # x111 = {1110, 1010}. The state of Karnaugh map after this covering is shown in Fig. 5c.
(4) As the fourth SM compulsory is chosen the minterm 1010 ∈ $G_F$. There the $EPI_4$ = 1x10 covering the SM 1010. Consequentially, $G_{EPI}$ = {x100, 1x01, x111, 1x10}. $G_{ON}$ = $G_{ON}$ # 1x10 = {1110, 1010} # 1x10 = ∅. The state of Karnaugh map after this covering is shown in Fig. 5d.

The simplification process is completed, because current state of the $G_{ON}$ is empty. Therefore, $G_{EPI}$ = {x100, 1x01, x111, 1x10}.

**Table 1**
Covering table for Example 1.

| PIs | | Minterms | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 0100 | 0111 | 1100 | 1101 | 1111 | 1110 | 1001 | 1010 |
| 1 | 11xx | | | + | + | + | + | | |
| 2 | x100 | + | | + | | | | | |
| 3 | 1x01 | | | | + | | | + | |
| 4 | x111 | | + | | | + | | | |
| 5 | 1x10 | | | | | | + | | + |

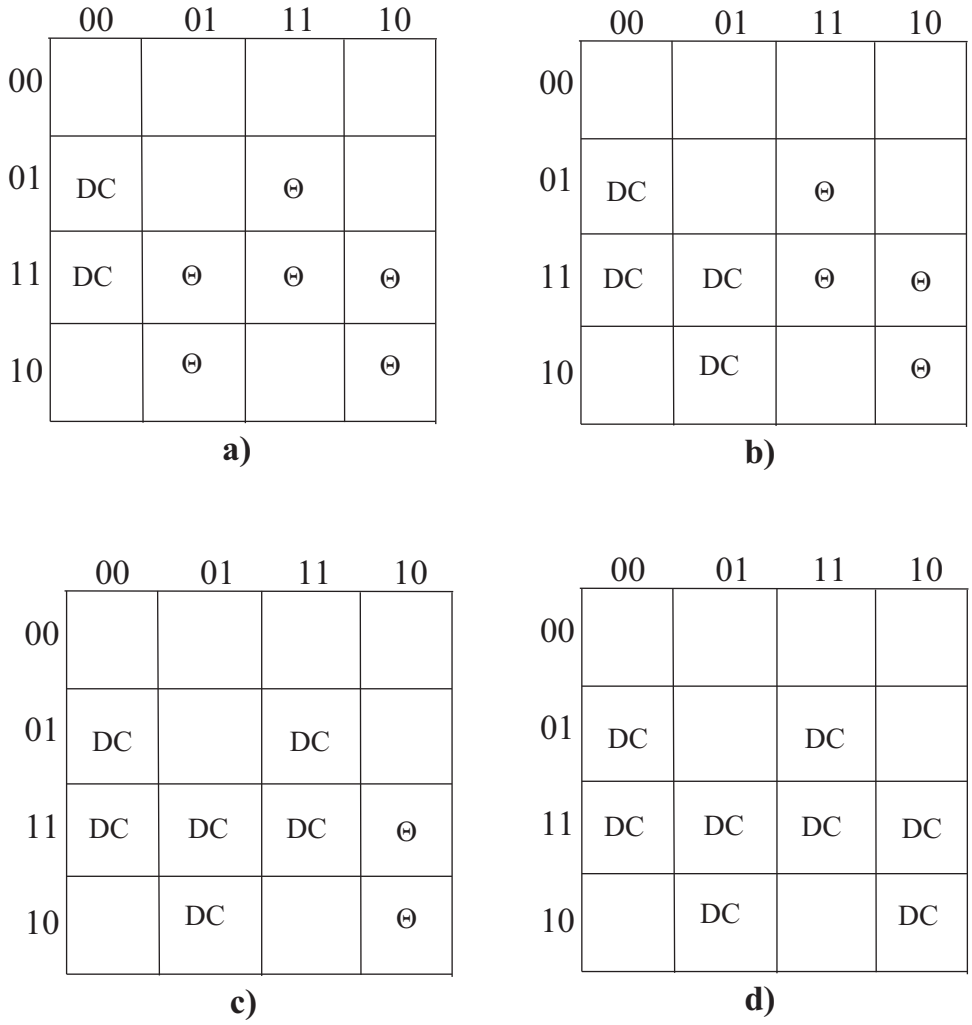



**Fig. 5.** Four states of function G after covered by x100 (a), 1x01 (b), x111 (c) and 1x10 (d).

But the step 1 of the demonstrated method may take long time for simplifying *ON* Minterm Series. Because, if there $|G_{ON}| = N = k2^n$, where $k < 1$, then

$$WC_1 = \sum_{i=1}^{N-1} i = \frac{1+N-1}{2} \times (N-1) = \frac{N(N-1)}{2} \cong \frac{N^2}{2} = \frac{(k\times 2^n)^2}{2} = \frac{k^2}{2} \times 2^n \Rightarrow O(2^{2n}) \quad (2)$$

iterations are required for performing this step (Eqn 2). Every *ON* minterm requires EXOR operation and each operation needs bit by bit comparison with *OFF* minterms. So this step has the exponential time-complexity that is undesired generally. On the other hand, this approach leads to better results than the random minterm choose principle, still cannot assure exact minimal results. So there is indefiniteness with the determination of an EPI from PIs having similar efficiency factors.

Smart minterm ordering lead us to more simplified results. Random minterm choose principle requires no time but cannot find the most simplified result in most cases. On the other hand, smart ordering by their contiguity takes time but finds more simplified results.

## 5. Accumulation approach

In this section, we define some decision rules for EPIs and explain the accumulation approach. Formation of SS and how to process SS are also explained. 3 algorithms for proposed method is explained step by step.

**Algorithm 1**: Accumulation Approach for Exact Minimization

(1) Pick a member of $G_{ON}$ series and define it as SM $X$
(2) Expand the members of $G_{OFF}$ series by SM $X$
(3) Extract non maximal cubes from results
(4) Remove step 3 from $n$-cube ($G_{PI}(x)$ = (xx…xx) # (step 3))
(5) If there is only one PI, write it to communication field
(6) If there are many PIs, create a SS and add it to series of $G_{SS}$
**(7)** Return step 1 unless $G_{ON}$ is empty
(8) If $|G_{PI}(x)|$ =1 then go to step 11
(9) Call Algorithm 2: Processing Accumulated States
(10) Subtract the result $G_{ON} = G_{ON} \# X$
(11) Send the EPI to communication field
(12) Get the EPI from communication field
(13) Subtract the EPI from current *ON* Minterm Series, $G_{ON} = G_{ON} \#$ EPI
(14) if $G_{SS}$ series is null, continue to step 18
(15) Clear EPI from communication field
(16) Call Algorithm 2, Processing Accumulated States
(17) If an EPI exists in the communication field then go to step 13
(18) Jump to step 1, unless $G_{ON} = \varnothing$
(19) If the set $G_{SS} = \varnothing$ then go to END
(20) Call Algorithm 3, Petrick's Algorithm
(21) **End**

**Fig. 6.** Accumulation approach for exact minimization.

**Algorithm 2**: Processing Accumulated States

(1) Mark all SSs as unprocessed
(2) Pick a SS and read its AM $X$ part
(3) Remove EPI(x) from AM
(4) If current result is null, delete the current SS and jump to step 8
(5) Apply FPP for $G_{PI}(X)$ section of current SS
(6) If there exists a $PI_k(x)$ satisfying Eq (3), jump to step 9
(7) Mark the SS as processed
(8) If there exists an unprocessed SS then jump to step 2
(9) Write $PI_k(x)$ as an EPI(x) to the communication field and clear SS worked on
(10) Return

**Fig. 7.** Processing accumulated states.

**Algorithm 3**: Petrick's Algorithm

(1) Put i = 1
(2) Peak out the first SS from the group $G_{SS}$
(3) Select the $G_{PI}(x)$ part of this SS
(4) Mark the PIs by $L_{i1}, L_{i2}, \ldots, L_{im}$, where $m = |G_{PI}(x)|$
(5) Form an inclusion function as the sum $IF_i = \bigcup_{i=1}^{m} L_{im}$
(6) Delete this SS
(7) $i = i+1$ and If $G_{SS} \neq \varnothing$ then go to 2
(8) Form and expand the presence function PF = $\bigcap_{i=1}^{K} IF_i$
(9) Select the product terms consisting of minimal number of $L_{im}$ literals
(10) Form the subsets $G_{PI}$ of the secondary EPIs by replacing each $L_{im}$ literal with the appropriate PI in each selected product term interpreted as a set consisting of own multiplicands
(11) Form the minimal cover sets with the set $G_{EPI}$.
(12) Return

**Fig. 8.** Petrick's method.

### A. **Advanced FPP Rule and Finding Best EPIs**

We need to pick a minterm to simplify and call it as Processing Minterm (PM). If a PM generates more than one PIs, determination of superior PI as an EPI, can be done by following FPP rule (3).

$$PI_1(x) > PI_2(x) \leftrightarrow (G_{ON} \cap PI_1(x)) \subseteq (G_{ON} \cap PI_2(x)) \tag{3}$$

As we mentioned before, (1) compare two series C1(x) = $G_{ON} \cap PI_1(x)$ and C2(x) = $G_{ON} \cap PI_2(x)$ according to unmasked minterm numbers. On the contrary, (3) compares two series by

their contents which lead to better results. It is obvious that the (3) can be enlarged by number of PIs.

$$PI_k(x) > \forall PI_{i\neq k}(x) \leftrightarrow (G_{ON} \cap PI_k(x)) \supset (G_{ON} \cap PI_{i\neq k}(x)) \quad (4)$$

(4) is looking for a $PI_k(x)$ which totally masks all *ON*-minterms masked by other *PI(x)s*. If such a $PI_k(x)$ exists, it is the superior one and will be picked as an EPI. However, if none of *PI(x)s* provide conditions for (4) then result is indefinite. We suggest to accumulate such PMs until a $PI_k(x)$ satisfying (4) exists. These PMs are accumulated as AM. AMs consist of PM and their PIs that will be used in the future. This special data form is called Suspended State (SS).

$SS(x) = \{X, G_{PI}(x)\}$ is created to process accumulated SM *X* where the $G_{PI}(x) = \{PI_2(x), \ldots, PI_m(x)\}$ is group of PIs including the SM *Xis given in (*Eqn 5*)*. Consequentially,

$$SS(x) = \{X, G_{PI}(x)\} = \{X, \{PI_2(x), \cdots, PI_m(x)\}\} \quad (5)$$

Handling of a SM, whose $PI_k(x)$ is not satisfying the Eq. (4), can be accumulated in two steps. First handling of a PM should be stopped, then $SS(x)$ will be created and accumulated into the series of $G_{SS}$

For instance, let $G_{ON}(Y) = \{011,110,010,001\}$, 010 is picked as PM and $G_{PI}(x) = \{x10, 01x\}$. The series $C_1(x)$ and $C_2(x)$ will be as follows;

$C_1(x) = \{011,110,010,001\}$ # x10 = {001, 011},

$C_2(x) = \{011,110,010,001\}$ # 01x = {001, 110}.

As $C_1(x) \not\subset C_2(x)$ and $C_2(x) \not\subset C_1(x)$, then SM *X* = 010 is accumulated.

(1) Processing of PM (0 1 0) is suspended.
(2) Regarding the PM, $SS_{(0\ 1\ 0)}$ is created as {010, {x10,01x}}
(3) Newly created SS is added to series of $G_{SS}$, $G_{SS} = G_{SS} \cup SS_{(0\ 1\ 0)}$

After accumulating a PM, an unmasked *ON*-minterm is picked as a new PM.

### B. Advanced Minimization Algorithms

We can summarize the presented method in 3 algorithms. Algorithm 1 shows the accumulation approach for exact minimization (see Figure 6), algorithm 2 shows how to process accumulated states (see Figure 7) and algorithm 3 explains a branch and bound technique, Petricks algorithm (see Figure 8).

Algorithm 1 expands On minterms (line 2), finds PIs (line 4) and use decision rules given in Eq. (3)–(5). According to these equations, some PIs may not cover each other. These PIs and related minterm is accumulated in a special form SS (line 6). If there is only 1 PI, this is called EPI (line 8, 11), otherwise accumulated states are processed (line 9) and new EPIs are subtracted from ON minterm series (line 10, 13). These processes are repeated until handling all minterms in communication field and ON minterm series (lines 14–18). If there exist some unsolved $G_{SS}$, then Petrick algorithm is used for minimization of remaining SSs.

Finding an EPI changes the remaining function. Thus, Algorithm 1 needs to reconsider SSs after finding every new EPIs. Algorithm 2 handles SSs and recalculate minterms and PIs. Lines 2–4 are checking minterms whether they are masked by EPI or not. Lines 5 and 6 are checking PIs if they are satisfying Eq (3). Some PIs may become essential after newly find EPI. Lines 7–10 do some marking for processed SSs and send newly find EPI to communication field.

Petrick's Algorithm is a famous branch and bound technique used for finding minimum subsets. This algorithm labels all PIs such as $L_{i1}$ in lines 3 and 4. Then lines 5–7 combinates the conditions satisfying function. Line 8 expands these combinations and forms all possible solutions. This algorithm selects minimal number of literals in line 9 and do this repeatedly for all PIs.

## 6. Implementation of proposed method

In this section we shall examine proposed method for insignificant function minimization. The proposed method can easily find exact solutions using rules and algorithms in section 5. Although accumulation approach can easily handle huge functions, it is not handy to explain their simplification procedures step by step. Thus we shall examine proposed method on an easy function.

**Example 2.** *Simplify function F using accumulation approach.*

$F_{ON}$ = {0000, 0001, 0010, 0011, 0101, 0110, 1000, 1011}

$F_{OFF}$ = {0100, 0111, 1001, 1010, 1100, 1101, 1110, 1111}

Let's begin minimization by 0000, SM = 0000. There are 2 PIs masking that minterm, $PI_{(0000)}$ = {00xx, x000}. 00xx masks more minterms than x000. Although 00xx is bigger than x000, we cannot decide it as an EPI. We should check our decision rule (3).

$PI_1(0000) > PI_2(0000) \leftrightarrow (F_{ON} \cap PI_1(0000)) \subseteq (F_{ON} \cap PI_2(0000))$

$PI_1(0000)$ = 00xx and $PI_2(0000)$ = x000

$G_{ON} \cap 00xx$ = {0000, **0001, 0010, 0011**}

$G_{ON} \cap$ x000 = {0000, **1000**}

As neither $PI_1(0000) > PI_2(0000)$ nor $PI_2(0000) > PI_1(0000)$ we cannot decide an EPI. According to Algorithm 1, an SS is created with PM and PIs.

$SS_1$ = {0000, {00xx, x000}} and $G_{SS} = \{SS_1\}$

We shall continue minimization by 0001, SM = 0001. There are 2 PIs masking that minterm, $PI_{(0001)}$ = {00xx, 0x01}. 00xx masks more minterms than 0x01. Although 00xx is bigger than 0x01, we cannot decide it as an EPI. We should check our decision rule (3).

$PI_1(0001) > PI_2(0001) \leftrightarrow (F_{ON} \cap PI_1(0001)) \subseteq (F_{ON} \cap PI_2(0001))$

$PI_1(0001)$ = 00xx and $PI_2(0001)$ = 0x01

$G_{ON} \cap 00xx$ = {**0000**, 0001**, 0010, 0011**}

$G_{ON} \cap$ 0x01 = {**0101**, 0001}

As neither $PI_1(0001) > PI_2(0001)$ nor $PI_2(0001) > PI_1(0001)$ we cannot decide an EPI. According to Algorithm 1, an SS is created with PM and PIs.

$SS_2$ = {0001, {00xx, 0x01}} and $G_{SS} = \{SS_1, SS_2\}$

Simplifying 0011 and 0010 as SM will result in similar situations as $SS_3$ and $SS_4$. These states are accumulated in $G_{SS}$.

$SS_3$ = {0010, {00xx, 0x10}} and $SS_4$ = {0011, {00xx, x011}}

$G_{SS} = \{SS_1, SS_2, SS_3, SS_4\}$

On the other hand, expanding SM = 0101 lead us to only one PI = {0x01}. Therefore, algorithm assigns this PI as an EPI, writes this EPI to communication field and call Algorithm 2 for processing accumulated minterms. Algorithm 2 checks $G_{SS}$ whether they are masked by EPI(0x01) or not. Only one of the suspended states ($SS_2$) is masked by EPI.

PI = {0x01} ⇒ $EPI_1$

$G_{SS} = G_{SS} - EPI_1 = \{SS_1, SS_2, SS_3, SS_4\} - EPI_1 = \{SS_1, SS_3, SS_4\}$

Minimization of SM = 0110 lead us to only one PI = {0x10}. Therefore, algorithm assigns this PI as an EPI, writes this EPI to communication field and call Algorithm 2 for processing accumulated minterms. Algorithm 2 checks $G_{SS}$ whether they are masked by EPI(0x10) or not. Only one of the suspended states($SS_3$) is masked by EPI.

PI = {0x10} ⇒ $EPI_2$

$G_{SS} = G_{SS} - EPI_2 = \{SS_1, SS_3, SS_4\} - EPI_2 = \{SS_1, SS_4\}$

Expanding 1000 and 1011 will create EPIs and these EPIs will be subtracted from $G_{SS}$.

SM = 1000 ⇒ PI = {x000} ⇒ $EPI_3$

$G_{SS} = G_{SS} - EPI_3 = \{SS_1, SS_4\} - EPI_3 = \{SS_4\}$

SM = 1011 ⇒ PI = {x011} ⇒ $EPI_4$

$G_{SS} = G_{SS} - EPI_4 = \{SS_4\} - EPI_4 = \{\varnothing\}$

$G_{SS} = \{\varnothing\}$ and $F_{ON} = \{\varnothing\}$.

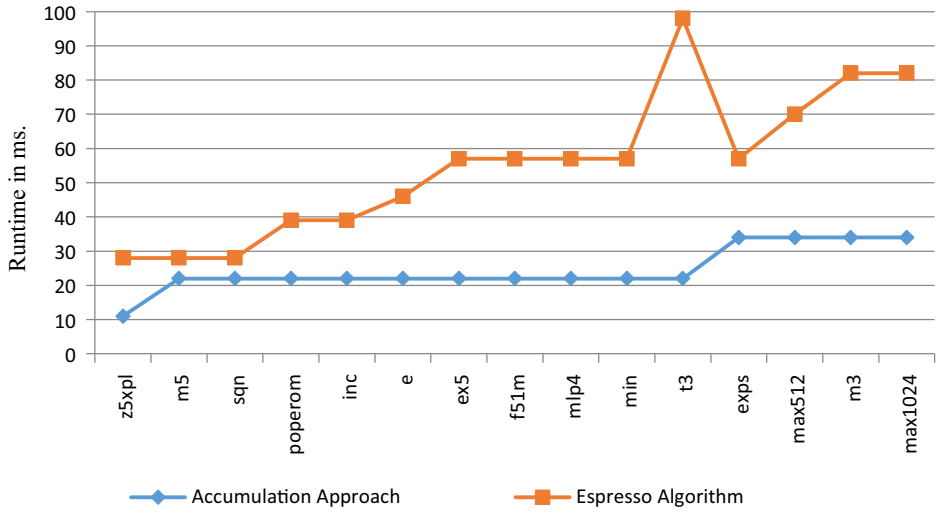



**Fig. 9.** Time Elapsed for simplification of 15 common MCNC benchmarks.

Finally all minterms are minimized, all SS are processed and there is no minterm in communication field. Exact minimized function is EPI = {0x01, 0x10, x000, x011}.

Proposed algorithm is coded in C language. Huge functions are minimized with both proposed algorithm and ESPRESSO. We shall describe some experimental results in the next section.

## 7. The experimental results

20 MCNC benchmarks were tested to determine the results quality and runtime of proposed method. The first output digits are used as output function. The qualities of results are calculated according to numbers of EPIs for each function. Same benchmarks were simplified by ESPRESSO-EXACT and results were compared by proposed method.

Both ESPRESSO and accumulation approach algorithm find equal number of EPIs for 15 of benchmarks. But concerning these benchmarks, accumulation approach algorithm is 2.24 times faster in average as seen from Fig. 9. Fastest and slowest performances of Espresso are 28 ms for "z5xpl" function and 98 ms for t3 functions respectively. Espresso calculated 15 functions in 825 ms in total. Fastest and slowest performances of proposed accumulation approach are 11 ms for "z5xpl" function and 34 ms for "max1024" functions respectively. Accumulation approach calculated 15 functions in 367 ms in total.

Remaining 5 extreme benchmarks reveals the deficient side of the algorithms. Fig. 10 shows that ESPRESSO is 1.75 times faster for only one benchmark (m4) which consists of many OFF minterms. This is an expected result as proposed algorithm is simplifying by using OFF minterms. Also sequence of ON minterms play an important role for algorithms [33,38,39]. They may result in worst or best case complexity for algorithms. Accumulation approach algorithm is 1.47 times faster for remaining 4 difficult benchmarks. Proposed algorithm calculated 4 extreme benchmarks in 235 ms while its 345 ms for Espresso. In addition, proposed algorithm created more simplified solutions as seen from Fig. 11. Regarding the 5 extreme benchmarks, accumulation approach found 56 EPIs, whereas ESPRESSO found 63 EPIs in total.

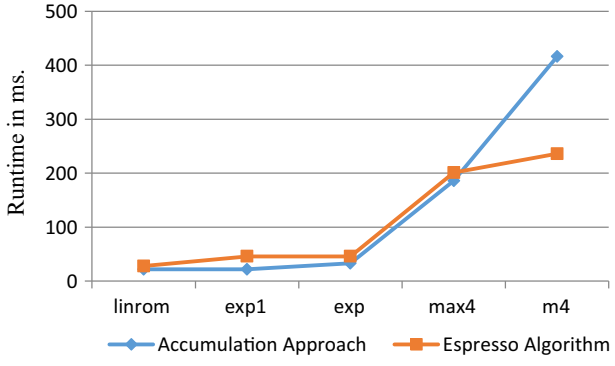



**Fig. 10.** Time elapsed for 5 extreme MCNC benchmarks.

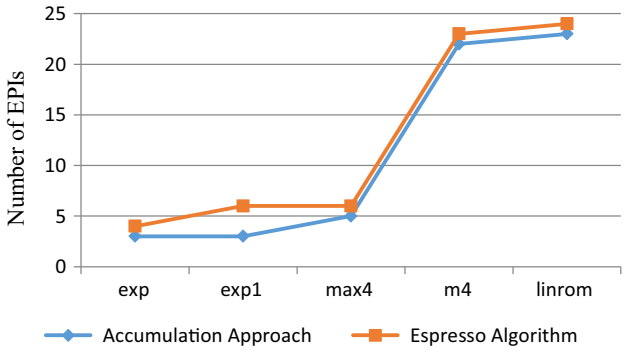



**Fig. 11.** Number of EPIs for 5 extreme MCNC benchmarks.

## 8. Conclusion

Although heuristic algorithms produce acceptable solutions to a complex problem in a reasonable time, researchers are looking for most simplified solutions in some cases [34–35]. Today's developing technology requires smallest integrated circuits. This could be done by decreasing number of input/output variables or number of circuit elements. Two level logic simplification, which we investigated in this paper, enables us to decrease input variables and circuit element numbers. To examplate, "m5" function has 27 *ON* minterms that represents 27 different input states of a circuit. But this function or circuit can be simplified to 4 input states. Thus, simplification process decreases production cost and energy consumption of circuits.

In this study, we investigated how minterm contiguity effects simplification results and proposed a smart minterm ordering approach. Ordering minterms by their contiguity could find more simplified functions but it has got exponential time complexity, which is undesired for simplification process. Secondly, a new

accumulation approach for function minimization is presented in this paper. Proposed simplification method makes use of direct masking approach to simplify logic functions given in SOP form. This method accumulates implicants which requires advanced decision procedure, thus avoids unnecessary EPIs. Proposed method is tested on 20 benchmarks and results show that our method generated better results than ESPRESSO. Proposed method is also faster than ESPRESSO in most cases (19). It is relatively slower for solving one function which includes significantly greater $G_{OFF}$ series than $G_{ON}$ series. As proposed logic simplification method includes DC conditions, some principles could be applied for synthesizing quantum gate circuits [36]. Proposed logic simplification method can also be used for feature reduction [1]. As Toffoli networks could be represented in SOP formed logic functions, this method could be developed to synthesis these networks [37]. Parallel computing may run algorithms faster [40,41]. In the future, proposed algorithms can be parallelized to be compatible with today's computer architecture.

## Acknowledgement

This study is supported by TUBITAK #1059B141500323 and the Coordinatorship of Selcuk University's Scientific Research Projects.

## References

[1] Borowik G, Luba T, Zydek D. Features reduction using logic simplification techniques. Int J Electron Telecommun 2012;58(1):71–6.
[2] Nwobi-Okoye CC, Okiy S, Igboanugo AC. Performance evaluation of multi-input–single-output (MISO) production process using transfer function and fuzzy logic: Case study of a brewery. *Ain Shams Eng J* 2016;7(3):1001–10. ISSN 2090-4479.
[3] Chuang CH, Lin CL, Chang YC, Jennawasin T, Chen PK. Design of synthetic biological logic circuits based on evolutionary algorithm. IET Syst Biol 2013;7(4):89–105.
[4] Poikonen JH, Lehtonen E, Laiho M. On synthesis of boolean expressions for memristive devices using sequential implication logic. IEEE Trans Comput-Aided Des Integrated Circ Syst 2012;31(7):1129–34.
[5] Sasao T, Butler JT. Worst and best irredundant sum-of–product expressions. IEEE Trans Comput 2001;50(9):935–48.
[6] Mishchenco A, Sasao T. Large-scale sop simplification using decomposition and functional properties. DAC 2003:149–54.
[7] Malik A, Brayton RK, Newton AR, Singiovanni-Vincentelli A. Reduced offsets for simplification of binary-valued functions. IEEE Trans Comput 1993;42(11):1325–42.
[8] Bergamaschi RA, Brand D, Stok L, Berkelaar M, Prakash S. Efficient use of large don't cares in high-level and logic synthesis. Int Conf Comput Aided Des 1995:272–8.
[9] Fiser P, Hlavicka J. Boom - a heuristic Boolean simplifier. J Comput Inform 2003;22:1001–33.
[10] Güneş S, Yılmaz N, Allahverdi N. A fault –tolerant multicast routing algorithm based on cube algebra for hypercube networks. Arab J Sci Eng 2003;28(1B):95–103.
[11] Butti D, Mangipudi SK, Rayapudi SR. Design of robust modified power system stabilizer for dynamic stability improvement using Particle Swarm Optimization technique. Ain Shams Eng J 2019;10(4):769–83.
[12] Hemeida AM, Alkhalaf S, Mady A, Mahmoud EA, Hussein ME, Ayman M, et al. Implementation of nature-inspired optimization algorithms in some data mining tasks. Ain Shams Engi J 2020;11(2):309–18. doi: https://doi.org/10.1016/j.asej.2019.10.003.
[13] Quine WV. A way of simplify truth functions. Am Math Monthly 1955;62:627–31.
[14] McCluskey EJ. Simplification of Boolean functions. Bell Syst Tech J 1956;35(6):1417–44.
[15] Rathore TS. Minimal realization of logic functions using truth table method with distributed simplification. IETE J Ed 2014;55(1):26–32.
[16] Tirumalai PP, Butler JT. Simplification algorithms for multiple-valued programmable logic arrays. IEEE Trans Comput 1991;40(2):167–77.
[17] Saad Y, Schultz MH. Topological properties of hypercubes. IEEE Trans Comput 1988;37(7):867–72.
[18] Dueck GW. Algorithms for the minimization of binary and multiple-valued logic functions. Ph.D. Thesis, Computer Science Department, University of Manitoba; 1988.
[19] Giovanni DM. Synthesis and optimization of digital circuits. New York: Mccraw-Hill; 1994.
[20] Brayton RK, Hachtel GD, Mcmullen CT, Singiovanni-Vincentelli A. Logic simplification algorithms for VLSI synthesis. Boston: Kluwer Academic; 1984.
[21] Veitch EW. A chart method for simplifying truth functions. Proc ACM 1952:127–33.
[22] Skiena SS. The algorithm design manual. New York: Springer; 2008.
[23] Benoit A, Robert Y, Vivien F. *A guide to algorithm design: paradigms, methods, and complexity analysis*. Boca Raton: Chapman & Hall/CRC; 2013.
[24] Miller RE. Switching theory. Vol. 1 Combination circuits, New York, John Wiley and Sons; 1965.
[25] Pomper G, Armstrong JA. Representation of multivalued functions using the direct cover method. IEEE Trans Computer 1981;30(9):674–9.
[26] Besslich PW. Heuristic simplification of MVL functions: a direct cover approach. *IEEE Trans* Comput 1986;C-35(2):134–44.
[27] Dueck GW, Miller DM. A direct cover MVL simplification using the truncated sum. Proc 17th Int Symp Multiple-Valued Logic 1987:221–7.
[28] Başçiftçi F. Local simplification algorithms for switching functions. PhD Thesis, Graduate School of Natural and Applied Sciences, Selcuk University; 2006.
[29] Kahramanlı S, Güneş S, Şahin S, Başçiftçi F. A new method based on cube algebra for the simplification of logic functions. Arab J Sci Eng 2007;32(1b):101–14.
[30] Başçiftçi F, Kahramanlı Ş. An off-cubes expanding approach to the problem of separate determination of the essential prime implicants of the single-output Boolean functions. EUROCON 2007, Warsaw-Poland; 2007. p. 432–8.
[31] Başçiftçi F, Kahramanlı Ş. Fast computation of the prime implicants by exact direct-cover algorithm based on the new partial ordering operation rule. Adv Eng Softw 2011;42:316–21.
[32] Umans Ch. The minimum equivalent DNF problem and shortest implicants. J Comput Syst Sci 2001;63:597–611.
[33] Başçiftçi F, Akar H. Finding isolated minterms in simplification of logic functions. International conference on challenges in IT, engineering and technology (ICCIET 2014) Phuket, Thailand; 2014.
[34] Aggarwal A, Rawat TK, Kumar M, Upadhyay DK. Design of optimal band-stop FIR filter using L1-norm based RCGA. Ain Shams Eng J 2018;9(2):277–89.
[35] Kumar M. Optimal design of fractional delay FIR filter using cuckoo search algorithm. Int J Circ Theor Appl. 2018;46:2364–79.
[36] Große D, Wille R, Dueck GW, Drechsler R. Exact synthesis of elementary quantum gate circuits. J Multi-Valued Logic Soft Comput 2009;15:283–300.
[37] Dueck GW. Challenges and advances in Toffoli network optimization. IET Comput Digital Tech 2014;8(4):172–7.
[38] Akar Hakan, Başçiftçi Fatih. Minterm and Implicant Selection Criteria for Direct Cover Algorithms and Off-Based Minterm Ordering. Malaysian Journal of Computer Science 2020. In press.
[39] Akar Hakan, Başçiftçi Fatih. Parallelised Algorithm of Isolated Minterm Detection for Logic Function Simplification. 4th International Symposium on Innovative Technologies in Engineering and Science 2016:1006–14.
[40] Akar Hakan, Başçiftçi Fatih, Uğuz Harun. Paralel ve Sıralı Brute Force Algoritmasının Karşılaştırılması. Akademik Bilişim Konferansı 2013:923–7.
[41] Ercan Uğur, Akar Hakan, Koçer Abdulkadir. Paralel Programlamada Kullanılan Temel Algoritmalar. Akademik Bilişim Konferansı 2013:861–5.

**Fatih Başçiftçi** is a Professor at Selçuk University (Turkey). He is currently the academic of Computer Engineering Department at the Faculty of Technology. He obtained his Bachelor and Master degree in 1997 and 2000 respectively. He then obtained his PhD degree of Electrical and Electronic Engineering at Selçuk University in 2006.

**Hakan Akar** obtained his Bachelor degree of Computer Education and Instructional Technology in Middle East Technical University in 2004 and Master Degree of Educational Sciences in Gaziosmanpaşa University in 2009. He is currently PhD student of Computer Engineering in Konya Technical University. His website is hakanakar.com.