A Novel Essential Prime Implicant Identification Method for Exact Direct Cover Logic Minimization

Sirzat Kahramanli and Suleyman Tosun
Computer Engineering Department
Selcuk University
Konya, Turkey

Abstract - *Most of the direct-cover Boolean minimization techniques use a four step cyclic algorithm. First, the algorithm chooses an On-minterm; second, it generates the set of prime implicants that covers the chosen minterm; third, it identifies the essential prime implicant; and fourth, it performs a covering operation. In this study, we focus on the third step and propose a new essential prime implicant identification method. In this method, when the identification of the essential prime implicant is impossible, we postpone dealing with current On-minterm and save a status word for it. Eventually, we retrieve the status words whenever a new essential prime implicant is identified. We compared the proposed minimization method with ESPRESSO-EXACT. The results show that our method obtains exact results faster than other ones.*

Keywords: Logic minimization, prime implicant, branch and bound technique, cover.

1 Introduction

Two-level logic minimization is a basic problem in logic synthesis [1]. Due to the exponential nature of this problem, state-of-the-art algorithms can typically handle functions with up to hundred products [2]. Therefore, most of the practical applications rely on heuristic minimization methods [2,3] with a complexity which is roughly quadratic in the number of products. There are also methods that produce the optimal (exact) solutions [1,2,3,4]. However, they can only be used for consistently small SOP realizations since they take long run time (CPU time) to find the final result. Heuristic algorithms are noticeably faster than the exact ones; however, they are still noticeably slow considering the exponential nature of the problem and they do not always obtain the optimal solutions [2]. Furthermore, the heuristic algorithms display diversity in realizations. That is, no single heuristic algorithm is consistently better than the others for all logic functions. There are classes of functions where one heuristic algorithm is better than the others [3]. Our studies show that the diversities in realizations of heuristic algorithms arise from the imperfect rules used for the identification of the essential prime implicant (EPI).

In this study, we propose a new EPI identification method that obtains exact minimal solutions. Our minimization algorithm randomly chooses an On-minterm to be handled (named as target minterm (TM)) from the On-set of the function being minimized. It obtains the set of prime implicants that include TM. If this set consists of a single PI this our algorithm selects this PI as EPI to cover the On-set. If there is more than one PI then, to find the EPI, we apply a rule, which selects the PI that covers all On-minterms included by other PIs. If such a PI does not exist in the set then we form a status word (MSW) for the current TM and for the PIs that include this TM. We use MSWs to obtain a new EPI. This procedure is repeated until all of the On-minterms are covered.

Our algorithm has the following differences than the existing ones. First, it is invariant to PIs generating method. Second, the minterm handling order can have a little effect on runtime, but it does not affect the final result. Third, it works in only one loop. Therefore, it is as fast as any heuristic one but generates only exact results.

The rest of the paper is organized as follows: The next section lists the abbreviations and notations used in this paper. Section 3 discusses branch-and-bound technique based EPI identification rules and their problems. Section 4 presents our approach. Section 5 gives the experimental data. Finally, Section 6 concludes this paper.

2 Abbreviations and Notations

In this section, we list the abbreviations and notations used in this paper to make the paper easy to follow.

Table 1: The abbreviations used in the paper.

The term	Abbreviations
Branch and Bound Technique	BBT
Direct Cover Principle	DCP
Essential Prime Implicant	EPI
Minterm Status Word	MSW
On-Minterm	OM
Prime Implicant	PI
Partial Ordering Operation	POO
Target Minterm	TM
Task Status Word	TSW

Table 2: The notations used in the paper.

The term	Notations
Boolean variables specification space	{0,1,x}, *where* x *denotes a non essential value*
Boolean functions specification space	{0,1,d}, *where d denotes an unspecified value*
The number of variables of a function	n
Sharp product (coordinate subtraction) operation	#
Intersection operation	$\cap$
Uniting operation	$\cup$
Current state of the set of On-minterms	S_{ON}
Set of the Off-minterms	S_{OFF}
Set of don't care mintems	S_{DC}
PI covering given TM	PI(TM)
Set of all PIs covering given TM	S_{PI}(TM)
Set of EPIs	S_{EPI}
Set of MSWs	S_{SW}
The size of a set S	|S|

3 BBT-based EPI Identification

In this section, we present an EPI identification method using BBT. We also demonstrate the shortcomings of this method on a small example.

3.1 EPI Identification Rules

DCP based heuristic methods are generally realized similar to the following algorithm [3,5-11].

```
DCM (S_ON, S_OFF). Direct cover minimization algorithm.

  while ($S_{ON}=\emptyset$) do
  1.        TM=X∈ $S_{ON}$;
  2.        $S_{PI}$(TM)_generation(TM, $S_{OFF}$);
  3.        EPI_identification($S_{PI}$(TM));
  4.        $S_{ON}$= $S_{ON}$#EPI;
```

First, the algorithm chooses the first element of the set SON as the minterm to be handled to identify its EPI (step 1 in DCM algorithm). This minterm is called the target minterm (TM). Second, it determines the set of PIs (SPI) that covers TM (step 2). Third, it identifies EPI among the elements of the set SPI (step 3). Fourth, it subtracts EPI from the set SON (step 4). This process is repeated until the set SON is empty.

For many functions, the major difficulty of the DCP based minimization lies in BBT [3,5-10] that employs a decision rule used in the third line of DCM algorithm when the TM is covered by more than one PI. BBT based on POO over the set SPI(TM) has been explained in [12] and later was improved by different researchers [3,5-11]. However, to the best of our knowledge, there is no POO that guaranties exact identification of EPIs in the literature. We discuss this problematic concept in the following subsections and propose a new method guarantying exact identification of EPIs in Section 4.

3.2 Choosing the EPI Randomly

The main idea of DCP based minimization methods is to choose the EPI once the $S_{PI}(TM)$ is obtained by using the following rule [3, 9]:

$$PI_k(TM) > PI_i(TM) \leftrightarrow \\ |S_{ON} \cap PI_k(TM)| > |S_{ON} \cap PI_i(TM)| \\ \forall (i,k / i \neq k) \tag{1}$$

where $\{PI_i(TM)\}_{i=1}^{k} = S_{PI}(TM)$ is the set of PIs covering the given TM. Then the PI$\in S_{PI}(TM)$ that covers the largest number of OMs is selected as EPI. If there are more than one such PI then one of them is selected randomly. However, this EPI selecting procedure does not always give the intended results since the exact EPI may not be PI(TM) that covers the largest number of OMs, but it is the one which covers all OMs included totally by all other PI(TM)s. We demonstrate this problem using the following example.

Example 1: Minimize the following completely specified function using Rule 1.

S_{ON}= {0011, 0100, 0101, 0111, 1001, 1101, 1110, 1111}

Figure 1 shows the initial Karnaugh map for the given function. In this figure, we have five PIs (numbered from one to five in Figure 1) covering eight OMs (starred boxes in Figure 1). Now, we apply Rule 1 to this problem. If we choose the OM 1101 (13)[1] as the first TM we obtain the PIs x1x1 and 1x01. Therefore, we have:

$R(13)_1$=$S_{ON0}\cap$x1x1={0011,0100,0101,0111,1001,1101, 1110,1111}#x1x1={ 0101,0111, 1101, 1111}

$R(13)_2$= $S_{ON0}\cap$1x01={0011,0100, 0101, 0111, 1001, 1101, 1110, 1111}#1x01={1001, 1101}

where $R(13)_1$ and $R(13)_2$ are the sets of intersections of PI x1x1 and PI 1x01 with the set S_{ON}, respectively. Since $|R(13)_1|$=4 and $|R(13)_2|$=2, we select PI_1(1101)=x1x1 as EPI_1. Therefore; S_{EPI} ={x1x1}; S_{ON}= S_{ON}#x1x1= {0011, 0100, 1001, 1110}.

[1] 13 is the decimal equivalent of the binary code 1101.

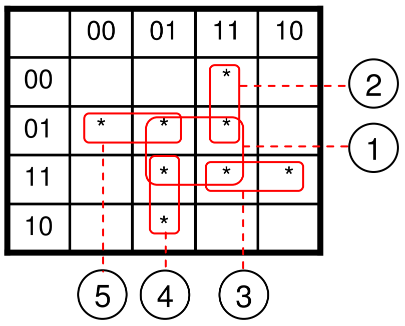

Figure 1: The initial Karnaugh map for Example 1.

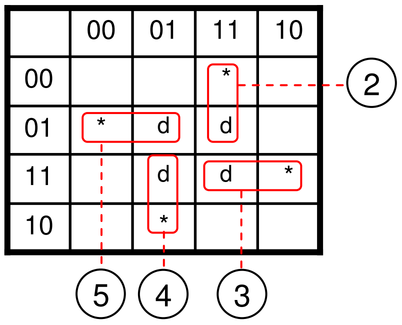

Figure 2: The Karnaugh map of the rest of the function being minimized after PI 1 is removed as EPI.

As we can see from Figure 2, the set S_{DC} = {0101, 0111, 1101, 1111} of don't care minterms is appeared, which is covered by EPI_1. Consequentially, the rest of the function being minimized is the incompletely specified function represented by sets S_{ON} and S_{DC} (see Figure 2).

As Figure 2 shows, all OMs are isolated. Therefore, to avoid further calculations, we identify all EPIs by inspection as follows:

EPI_2=0x11, EPI_3=111x, EPI_4=1x01, EPI_5=010x, i.e.

S_{EPI}={1x1x, 0x11, 111x, 1x01,010x}

Note that we can obtain the same result if we select the OMs 0101, 1101, and 1111 as first four TMs. Table 3 shows the covering diagram for Example 1. Using this table, we can form the Petrick BBT function [1] as follows:

S_{EPI} = BE(A+E)(A+B)D(A+D)C(A+C) = BEDC = BCDE $\Rightarrow$ {0x11, 111x, 1x01, 010x}

This function shows that the PI $(13)_1$=x1x1, which has been identified above as EPI_1, actually is not EPI. Hence, S_{EPI} ={0x11, 111x, 1x01,010x}.

Our analysis of the Example 1 shows that the exact result can be reached only in 4!=24 cases when we choose the isolated OMs; 0011, 0100, 1001 and 1110 as first four TMs. Furthermore, the number of choosing orders leading to non-exact results is 4x4!=96. Namely, the percentage of a fail is 96/(96+24)=80%. The general estimation of ratios of the good and worst results generating by non-exact minimization methods is given in [1,3].

Table 3: Covering diagram for Example 1

PIs	OMs	0011	0100	0101	0111	1001	1101	1110	1111
A	x1x1			+	+		+		+
B	0x11	+			+				
C	111x							+	+
D	1x01					+	+		
E	010x		+	+					

3.3 Minterm Ordering Based on Adjacency Factors

To avoid the possible faulty results like the one we demonstrated in the previous subsection, the improved DCP method has been proposed [3,6]. In this method, for each PI, the efficiency factor, which is equal to the number of OMs that the PI covers divided by its cost, is calculated and the PI with the largest efficiency factor is chosen as EPI. If there are more than one PI with the same largest efficiency factors, one of them is chosen randomly. However, this method is also problematic and may not always give the exact minimal results. One may not identify the EPI among the PIs with the same efficiency factor. Moreover, the calculations of the efficiency factors can take very long time for the large On-sets. If $|S_{ON}|=N=k2^n$ then, the calculations of the efficiency factors requires comparisons (each of which consist of one EXOR operation and several others to detect result containing 1 only in one bit position and 0's in all others) the count of which is expressed by the following formula.

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

That is, this ordering has the exponential time-complexity which is undesirable [9].

4 Our EPI Identification Method

4.1 Main Minimization Algorithm

We give the pseudo code of the main algorithm for the DCP based minimization of the single-output two-level Boolean functions as follows:

MAIN (S_{ON}, S_{OFF}). *Main algorithm.*

```
    $S_{EPI}=\varnothing$;
M1:  if ($S_{ON} =\varnothing$) then go to M3;
     TM = $S_{ON}[m_1]$;
     CALL $S_{PI}$(TM, $S_{OFF}$);
M2:  if (CF=$\varnothing$) then go to M1;
     CALL CP(CF, $S_{ON}$);
     if ($S_{SW} =\varnothing$) then go to M1;
     CALL SWH($S_{ON}$, CF, N);
     Go to M2;
M3: if ($S_{SW} =\varnothing$) then go to M4;
     CALL PP($S_{SW}$);
M4: end
```

Our algorithm starts by selecting a minterm as TM from the set S_{ON} then calls the procedure S_{PI}(TM, S_{OFF}) to find the set S_{PI}(TM) and to identify EPI from this set. If this procedure cannot find any EPI the algorithm selects a new minterm to continue. In the meantime, the S_{PI}(TM,S_{OFF}) procedure saves a status word (MSW) for the minterm of which we could not find any EPI. This procedure will be explained in the next subsection in detail.

After finding the EPI, we call the covering procedure CP (CF, S_{ON}). If we do not have any MSW saved, we continue selecting a new minterm and try to find its correspondent EPI. Otherwise, (i.e. we have some status words saved), we call the MSWs handling procedure SWH (S_{ON}, CF, N). This procedure looks MSWs one at a time to identify and use for covering PIs that have become EPIs. This part of the algorithm is repeated until the On-set is empty. The process is completed if there are no saved MSWs left. Otherwise, we call Petrick procedure and obtain a minimal subset of PIs covering the TM parts of saved MSWs

4.2 Our TM Handling Method

In this subsection, we explain the procedure S_{PI}(TM, S_{OFF}) that identifies the EPI to cover the given TM. We give the pseudo code of this procedure as follows:

S_{PI}(TM,S_{OFF}). *The procedure for calculating the set S_{PI}(TM), identifying EPIs and forming MSWs.*

```
     $P_E:S_{OFF} \longrightarrow S_E$ ; $P_{EO}:S_E \longrightarrow S_{EP}$;
     R=xx…x# $S_{EP}$;  $P_{EO}$:R $\longrightarrow S_{PI}$(TM);
PI1:  if ($|S_{PI}(TM)| > 1$) then go to PI2;
     CF= $S_{PI}$(TM); Go to PI3;
PI2: CF=$\varnothing$;
     SW(TM)={TM {SPI(TM)}};
     $S_{SW} =S_{SW} \cup$ SW(TM);
PI3: return
```

In this procedure, first we expand the minterms in the set S_{OFF} by using current TM and remove the non-prime cubes from this set. Then, we subtract the result from the n-cube and again remove the non-prime cubes from the set S_{PI}. If the TM at hand is covered more than one PI, we apply the following POO to identify the EPI among the PIs (PI1 in the above pseudo code).

$$PI_k(YM) > PI_i(TM) \leftrightarrow (S_{ON} \cap PI_k(TM) \supseteq (S_{ON} \cap PI_i(TM)), \forall(i,k / i \neq k) \qquad (2)$$

The principal difference between Rule (1) and Rule (2) is that the former compares the size of the sets $R_i(TM)=S_{ON}\cap PI_i(TM)$ and $R_j(TM)= S_{ON}\cap PI_j(TM)$ whereas the latter compares the contents of these sets; therefore, Rule (2) leads to the exact minimal results. That is, the PI_k(TM) is EPI if and only if it covers all OMs included by all others PI(TM)s.

If the condition expressed by Rule (2) is not satisfied by any of the PIs, the problem of choosing the EPI is revealed. To solve this problem, we postpone the handling of the TM at hand until one of the PI_k(TM)s satisfies the condition. To resume handling of the postponed TM we use the following information called a Minterm Status Word (MSW).

$$MSW(TM_i)=\{TM_i, S_{PI}(TM_i)\}$$

where $S_{PI}(TM_i)=\{PI_2(TM_i), \ldots, PI_m(TM_i)\}$ is the set of PIs including TM_i.

Handling of TM for which there is no PI satisfying the condition expressed by Rule (2) may be postponed by stopping of handling of TM, forming a MSW for it and including this MSW into the set S_{SW}. In addition, the existence of different postponed TMs and appropriate MSWs in different loops of the algorithm request to use the information about the last state of the task being solved. This information is called a *Task State Word* (TSW) and has the following structure:

$$TSW =\{S_{ON}, S_{EPI}, S_{SW}\}$$

where, S_{ON} is the current (last) state of the set of On-minterms, S_{EPI} is the current state of the set of EPIs, S_{SW} is the current state of the set of MSWs.

Example 2: Let us illustrate how MSW is used in our method. To do this, assume we have S_{ON}= {001, 010, 011, 110}, TM=011(3) and S_{PI}(011)= {01x, 0x1}. Then, $R(3)_1$={001, 010, 011,110}$\cap$01x ={010, 011}; $R(3)_2$={001, 010, 011, 110}$\cap$0x1={001, 011}. Since $R(3)_1 \not\subset R(3)_2$ and $R(3)_2 \not\subset R(3)_1$, handling of TM=011 may be postponed as follows:

```
TMP(TM, $S_{PI}$(TM))
        Stop handling of TM =011,
        Form MSW as set {011,{01x, 0x1}},
        Include this MSW into the set $S_{SW}$.
```

4.3 The Remaining Procedures in the Main Algorithm

In this subsection, we present the other procedures included in our main algorithm. These are the direct covering (CP(CF, S_{ON}), MSWs handling (SWH(S_{ON}, CF, N), and the Petrick (PP(S_{SW})) procedures.

CP(CF, S_{ON}). *Direct covering procedure.*

```
    $S_{ON} = S_{ON}\# CF$; $S_{EPI} = S_{EPI} \cup CF$; $CF = \varnothing$;
    return
```

Direct covering procedure subtracts the found EPI that presents in the connection field CF from the set S_{ON} and includes this EPI into the set S_{EPI}. After that, it empties the CF.

SWH(S_{ON},CF,N). Minterm *status words handling procedure.*

```
        i=1;
SW1:    select MSW$_i$;
        $TM_C = MSW_i[TM]$; $TM_C = TM_C\#CF$;
        if ($TM_C = \varnothing$) then go to SW2;
        $S_{PIO}$ = POO: $S_{PI}$;
        if ($|S_{PIO}| > 1$) then go to SW3;
        CF = $S_{PIO}$; MSW$_i$[TM]=$\varnothing$; Go to SW4;
SW2:    MSW$_i$[TM]=$\varnothing$;
SW3:    i=i+1;
        if ($i \le N$) then go to SW1;
        CF =$\varnothing$;
SW4:    Delete all MSW with MSW[TM]=$\varnothing$;
        Update the value of N;
        return;
```

Minterm status word handling procedure first selects the first MSW from the set S_{SW}. Then, it picks the TM part (TM_C) of this MSW and subtracts the last founded EPI from TM_C. If the result is empty then the procedure removes the content of TM part of MSW at hand and selects the next MSW. Otherwise, the procedure partially orders the S_{PI} part of the current MSW to identify EPI. If the EPI is found (the content of S_{PIO} is the new EPI if $|S_{PIO}|=1$) then the procedure sends it into the connection filed CF, and removes the content of TM part of MSW at hand. When the TM part of MSW is not covered by the last founded EPI and the procedure does not identify the new EPI, the procedure handles the second MSW in the same way. This part of the procedure is repeated until the new EPI is found or all of MSWs are handled. Then the procedure deletes all of MWSs the TM parts of which are empty and returns.

PP(S_{SW}). *Petrick procedure.*

```
    i=1;
    while ($S_{SW} \ne \varnothing$)
        Select SW$_i$;
        $S_{PIi} = SW_i[S_{PI}(TM)]$;
        $IF_i = \bigcup_{i=1}^{m} L_{im}$ ; Delete SW$_i$; i=i+1;
    end while
        $PF = \{ \bigcap_{i=1}^{K} IF_i \}$;
        PR = min{L(PT$\in$ PF)};
        $L_{im} \leftarrow PI_{im}$;
        $PC = \bigcup_{im} P_{im}$ ; $RMC = S_{EPI} \cup PC$;
    return
```

Petrick procedure selects each MSWs and employs the inclusion function (IF) for its PIs part, where $m=|S_{PIi}|$ and $L_{im}=PI_{im}\in S_{PIi}$. Then it takes the product of these inclusion functions. Afterwards, it selects the -product term (PT) formed by minimum count of literals. If there is more than one such PT then it selects one of them randomly. After selecting PT, it replaces each literal with appropriate PI.

Example 3: Minimize the function given in the Example 1 using Rule (2).

Let us first explain a task state word by ordered set TSW ={S_{ON}; S_{EPI}; S_{SW}}. According to this format the initial state of the given task is:

TSW0= {{0011, 0100, 0101,0111,1001,1101, 1110, 1111}; $\varnothing$;$\varnothing$}

In the following enumerations such as 1.1., first number represents the iteration number and the second one represents the current step of the iteration.

1.1. Let us, as in Example 1, first choose TM =1101 for which S_{PI}(1101)= {x1x1,1x01}. In other words, this minterm is included by the PIs x1x1 and 1x01. Since $|S_{PI}(1101)|>1$, it is necessary to subject the set S_{PI}(1101) to POO.

1.2. $R_{13,1}=S_{ON}\cap$x1x1={0101,0111,1101, 1111}; $R_{13,2}= S_{ON}\cap$1x01={1001, 1101}.

Since $R_{13,1} \not\subset R_{13,2}$ and $R_{13,2} \not\subset R_{13,1}$, none of the PIs can be identified as EPI. Therefore, it is necessary to form an MSW for TM as MSW_1 ={1101,{x1x1,1x01}} which changes TSW0 into:

TSW1= {{0011, 0100, 0101, 0111, 1001, <u>1101</u>, 1110, 1111}; ∅; {{1101,{x1x1,1x01}}}

2.1. Now, let us choose TM =1110 for which S_{PI} contains single cube 111x that is EPI_1 (see Figure 1) and must be used for covering the set S_{ON}.

2.2. S_{ON}= S_{ON}#111x={0011, 0100, 0101, 0111, 1001, 1101}. Therefore,

TSW2= {{0011, 0100, 0101, 0111, 1001, <u>1101</u>}; {111x}; {{1101, {x1x1, 1x01}}}

2.3. Since the new EPI is identified, it is necessary to handle MSWs that have been saved.

S_{SW}={SW_1}={{1101,{x1x1,1x01}};
TM_C=MSW_1[TM]#EPI_1=1101#111x=1101≠∅

Since the TM of MSW_1 is uncovered by EPI_1, it is necessary to subject to POO the PIs part of MSW_1.

$R_{13,1}$=S_{ON}∩x1x1={0101,0111,101};
$R_{13,2}$=S_{ON}∩1x01={1001, 1101}

Since $R_{13,1} \not\subset R_{13,2}$ and $R_{13,2} \not\subset R_{13,1}$, none of PIs can be identified as EPI. Therefore, there is no change in MSW_1 and in other parts of TSW2.

3.1. Let us choose the next TM as TM=1001, for which S_{PI} contains single cube 1x01 that is EPI_2 (see Figure 1) and must be used for covering the set S_{ON}.

3.2. S_{ON}= S_{ON}#1x01={0011, 0100, 0101, 0111}. Consequentially,

TSW3.1 = {{0011, 0100, 0101, 0111}; {111x, 1x01}; {{1101, {x1x1, 1x01}}}

3.3. Since the new EPI is identified it is necessary to handle the MSWs that have been saved.

S_{SW}={SW_1}={{1101,{x1x1,1x01}};
TM_C=MSW_1[TM]#EPI_2=1101#1x01=∅

Since the TM of MSW_1 is covered by EPI_2=1x01, it is necessary to delete MSW_1, i.e.

TSW3.2={{0011, 0100, 0101,0111 };{111x, 1x01}; ∅}.

4.1. Now, let as choose TM= 0011. This minterm is included by the single cube 0x11 that is the EPI_3 (see Figure 1). Therefore,

S_{ON}= S_{ON}# 0x11={ 0100, 0101}.

Consequentially,

TSW4={{0100, 0101};{111x, 1x01,0x11}; ∅}.

5.1. Let us now choose TM=0100. This minterm is included by the single cube 010x that is the EPI_4.

5.2. Since S_{ON}=S_{ON}#010x=∅, TSW5={∅; {111x, 1x01, 0x11, 010x}; ∅}. Consequently, the minimization process is completed. The exact minimal result is

S_{EPI}={111x, 1x01,0x11, 010x}

5 Experimental Results

In this section, we evaluate the performance of our method by using the standard MCNC benchmarks [13-15]. We compared our method with ESPRESSO-EXACT [14,15]. The sets of PIs were calculated by the Off-set expanding method [16]. We used the outputs of MCNC benchmarks as the 50 different single-output functions. Table 4 gives the results of our experiments. In this table, first four columns are the name of the benchmark, the number of variables, the size of S_{ON}, and the size of S_{OFF}. Column five and six show the runtime (as milliseconds) of our method and EXPRESSO-EXACT, respectively. The last column of Table 4 gives the runtime ratio of our method and EXPRESSO-EXACT. As we observed, our method takes less time to solve 66% of the benchmarks whereas EXPRESSO-EXACT has better runtimes from our method for 6% of the benchmarks. For the remaining (for 28%) of the benchmarks our method and EXPRESSO-EXACT obtain the exact minimal result at the same time. In general, the relative speed of our method is increased by decreasing the ratio $|S_{OFF}|/|S_{ON}|$. For example, the first output of the benchmark *prom1* depending on 9 variables and having the ratio$|S_{OFF}|/|S_{ON}|$= 14/488=0.03 was solved by our method 408 times faster than ESPRESSO-EXACT. On the other hand, the first output of the benchmark *bcb* depending on 26 variables and having the ratio |SOFF|/|SON|= 289/10=29, was solved by our method 1.5 times slower than ESPRESSO-EXACT.

6 Conclusions

In this study we propose a single-output two-level Boolean function minimization algorithm. The proposed algorithm is as fast as any single loop heuristic minimization algorithm and generates exact result. It is invariant to the minterm choosing order from the final result point of view. However, this order can have a little effect on the runtime.

To overcome this problem, we postpone the handling of minterms and save a status word for them. We use the use status words later when we revisit the postponed minterms. From this point of view, the most complex tasks for this algorithm are the ones that have prime implicants forming cyclic graphs. However, this kind of tasks is rare in practice. Our essential prime implicant identification rule can also be used for multiple-output Boolean functions without any improvement.

7 Acknowledgement

This project is supported by Selcuk University project number 06701088.

8 References

[1] T. Sasao, *"Worst and Best Irredundant Sum-of-Product Expressions"*, IEEE Trans. Comp., Vol. 50, No 9, 2001, pp. 935-947.

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

[3] P.P. Tirumalai and J.T. Butler, "*Minimization Algorithms for Multiple-Valued Programmable Logic Arrays*", IEEE Transactions on Computers, Vol. 40, No2, 1991, pp.167-177.

[4] R.K. Brayton et al., "Logic minimization algorithms for VLSI synthesis", Boston, Kluwer Academic Publications, 1984.

[5] G. Promper and J. Armstrong, "*Representation of Multivalued functions using the direct cover method*", IEEE Trans. Comp., pp 674-679, 1981.

[6] P.W. Besslich, "*Heuristic Minimization of MUL functions: A direct cover approach"*, IEEE Trans. Comp., pp 134-144, 1986.

[7] G.W. Dueck and D.M. Miller, *"A direct cover MUL minimization using the truncated sum"*, Proc. of the 17th Int. Sem. Multiple-Valued Logic, 1987, pp 221-227.

[8] O. Coudert, *"Two Level Logic Minimization: An Overview",* Integration. The VLSI Journal, 17-2, pp 97-140, 1994.

[9] Ch. Umans, *"The Minimum Equivalent DNF Problem and Shortest Implicants*", *Journal of Computer and System Sciences*, 63, 2001, pp. 597-611.

[10] P. Fiser and J. Hlavicka, *"BOOM- A Heuristic Boolean Minimizer*" , Journal of Computing and Informatics, Vol. 22, pp. 1001-1033, 2003.

[11] S. J. Hong and S. Muroga, "*Absolute Minimization of Complete Specified Switching Functions*", IEEE Trans. Comp., Vol.40 , No 1, pp 53-65, 1991.

[12] R.E. Miller, "*Switching Theory*", Vol. 1, New York, 1965.

[14]ftp://ftp.menc.org/pub/benchmark/Benchmark_dirs/ LGSynth93/testcases/pla/

[14] ftp://ic.eecs.berkeley.edu

[15] http://eda.seodu.co.kr/~ chang//download/espresso/

[16] S. Kahramanlı and N. Allahverdi, "*An Algebraic Approach to the Transformations on Hypercube System*", Mathematical & Computational Applications, 4(1), 1996, pp. 50-59.

Table 3: Runtimes for the standard MCNC benchmarks

Benchmarks	Number of variables	Size of S_{ON}	Size of S_{OFF}	Runtime (msec.)		T_E/ T_P
				Our method	Espresso	
check / wim	04	4/9	9/1	15,6	31,3	2
check1 /check2/check3	04	4/4/8	8/6/5	15,6	15,6	1
p82 / squar	05	11/9	13/23	15,6	15,6	1
m / new2/ sqr	06	27/3/18	5/4/46	15,6	31,3	2
m5 / poperom	06	27/56	5/8	15,6	15,6	1
Inc/ z5xp1	07	12/25	22/103	15,6	15,6	1
linrom	07	65	63	15,6	46,9	3
max128/ max3/ sqn	07	29/12/48	99/116/48	15,6	31,3	2
Dist /ex5	08	53/33	203/223	15,6	31,3	2
e	08	65	128	15,6	46,9	3
exp	08	18	52	15,6	15,6	1
exps / f51m/ m3/ m4	08	65/128/98/223	131/128/30/26	15,6	46,9	3
exp1	08	19	47	15,6	31,3	2
mlp4	08	32	224	31,3	31,3	1
root	08	15	241	15,6	31,3	2
rd84	08	120	136	31,3	46,9	3
apex4/max4/min	09	4/38/82	434/8/51	15,6	31,3	2
max512	09	258	254	15,6	46,9	3
prom1	09	488	14	31,3	12781,3	408,3
prom2	09	142	145	15,6	15,6	1
Max1024	10	516	508	15,6	31,3	2
T10	10	113	201	78,1	93,8	1,2
T11	10	266	371	93,8	109,4	1,17
T12	10	266	758	140,6	78,1	0,55
br11/ br1/ br2/ t3	12	29/29/33/27	5/5/2/121	15,6	31,3	2
pdc	16	24	1891	15,6	46,9	3
spla	16	67	2036	62,5	46,9	0,75
den	18	24	3	15,6	31,3	2
bca /bcc	26	9/3	292/242	31,3/46,9	31,3/46,9	1
bcb	26	10	289	46,9	31,3	0,67