PALMINI - FAST BOOLEAN MINIMIZER FOR PERSONAL COMPUTERS.

Loc Bao Nguyen +
Marek A. Perkowski + +
Nahum B. Goldstein + + +

+ HF2-13, Intel Corporation, 5200 N.E. Elam Young Pkwy, Hillsboro, OR. 97123,
+ + Portland State University, Department of Electrical Engineering,
+ + + Department of Engineering, Harvey Mudd College, Claremont, CA 91711.

ABSTRACT

This paper describes a fast and efficient method for minimization of two level single output Boolean functions. The minimization problem is reduced to that of coloring of the graph of incompatibility of implicants. The program permits also to remove static hazards and allows inversion of output's polarity which proves to be very convenient when designing with PAL's. It gives solutions within a very reasonable amount of time. On small industrial examples its speed is slightly better than Espresso and it occupies 6 times less memory.

1. INTRODUCTION.

There has been recently an interest in programs for optimization of PLAs and PALs: Presto [7], Espresso [4], Espresso-mv [26], Presol-II [1], Mini [14], [2], [3], [8]. Two approaches are currently known: algorithms that look for the minimum solution: [10], [27], [28], [6], [12], [25], [4], [30], [9], [16] and approximate algorithms [4], [7], [14]. The most advanced programs for minimum solution are Espresso-Exact [27], [28] and McBoole [11]. Detailed comparison of these two programs is given in [27]. All algorithms which search for the minimal solutions include two stages: generation of prime implicants and minimum covering of minterms with prime implicants. The number of prime implicants increases rapidly with the number of minterms, especially for functions with many don't cares. The set of prime implicants can become too large to enumerate even if it is possible to represent the function in two-level form [19]. This result limits the application of algorithms based on generating all prime implicants. The covering problem is NP-hard [12]. Some functions that lead to extremely hard to solve covering problems have been constructed [16]. It results then that there are two reasons why the current approaches to exact minimization will meet limited success.

We have developed a new approach which produces minimum solutions without generating prime implicants and without solving the covering problem at all.

Our approach is general and can be applied to many covering problems that occur in Boolean minimization. **We reduce the covering problems to the coloring problems.** In this paper we will present only one application: minimization of completely specified single-output functions in 2-valued algebra. (Our general method has some special properties for completely specified functions, very strongly unspecified functions, multi-output and multi-valued functions).

Instead of solving **the covering problem** with prime implicants we solve **the coloring problem** for a graph whose nodes correspond to minterms or some implicants of a new type. Therefore, we solve one NP-hard problem (graph coloring) instead of two NP-hard problems (generation of implicants and covering).

Graph coloring can be solved approximately or exactly. We have written several graph coloring algorithms of both types. In this paper the coloring algorithms will be not discussed and the results will be given for an approximate coloring algorithm. Replacing this algorithm with the optimal graph coloring algorithm will produce minimum solutions [24].

The graph for coloring is created with any on-cubes of the function as nodes. These can be minterms, arbitrary cubes (product implicants), minimal product implicants of the function or disjoint minimum implicants proposed below. Minimal implicant for a minterm M is a product of all prime implicants covering M. The number of such implicants never exceeds the number of minterms or the number of prime implicants.

2. SOME BASIC DEFINITIONS.

Boolean functions are implemented as arrays (sets) of cubes. The following notation will be used: ON[f] is a set of ON-cubes of function f. OFF[f] is a set of OFF-cubes of function f. DC[f] is a set of Don't-Care cubes of function f. Cube Ci is a string of 0's, 1's, and X's; it represents a product of literals of function f. An implicant of a function is an arbitrary subset of its on-cubes and dc-cubes. A product implicant is an implicant being a cube. A prime implicant is a product implicant which is not included into any other product implicant of that function. ϵ denotes belongs to a set. $\supseteq$ means inclusion of arrays of cubes or cubes (S $\supseteq$ T if the set of all minterms of T is included in the set of all minterms of S). $\cap$ denotes an intersection of arrays of cubes (S $\cap$ T is some set of cubes that includes all minterms that are included both in S and in T).

Example:
{01X,0X1} $\cap$ {X10,0X1} = (01X $\cap$ X10) $\cup$ (01X $\cap$ 0X1) $\cup$ (0X1 $\cap$ X10) $\cup$ (0X1 $\cap$ 0X1) = 010 $\cup$ 011 $\cup$ 011 = 010 $\cup$ 011 = 01X.

[P]* is a set of minterms included in function (product implicant) P. # denotes a sharp operator (S # T contains all of the minterms of S which are not contained by T).

Example:
0XX # 01X = 00X.

3. COMPATIBLE IMPLICANTS AND COMPATIBLE SETS.

The goal of this section is to discuss some properties of product implicants, which are essential for the reduction method that will be presented in section 4.

First, we introduce the matching operator, which is a main logic operation in our system.

Definition 1. Let C1 = ($C1^1$, ..., $C1^n$) and C2 = ($C2^1$, ..., $C2^n$) be cubes, where n is the number of input variables. The matching operator is defined as follows:

$C1,2 = (C1,2^1, ..., C1,2^n) = C1 \$ C2 = (C1^1 \$ C2^1, ..., C1^n \$ C2^n)$ where the operation \$ on single bit is defined as follows:

$$C1^i \$ C2^i = \begin{cases} C1^i & \text{when } C1^i = C2^i \\ X & \text{otherwise} \end{cases}$$

When the positional cube notation is used this operator corresponds to the component-wise Boolean OR of the two cubes. This holds true for any m-valued algebra (see [28]). This definition permits for application of our method basically with no modifications to Boolean algebras with multiple-valued inputs [29], [26]. The definition of matching of C1 and C2 is the same as the definition of the supercube of C1 and C2 from [27]. For ease of notation and consistency with our previus papers [22] we will however continue to use the name of matching and symbol \$. The operation \$ is commutative and associative and the result of its operation is always a cube.

Theorem 1. Let PI be a prime implicant of a completely or a partially specified Boolean function f. Then, for each set SM of product implicants (paricularly minterms) of the function f which are covered by PI:

$SM = \{m_1, m_2, ..., m_r\} \subseteq [PI]^* \subseteq ON(f) \bigcup DC(f)$

the following relation holds

$$\$_{m_i \in SM \subseteq [PI]^*} m_i \subseteq \Pi$$

i.e., a cube resulting from the matching of all minterms included in any subset of minterms of a prime implicant of a Boolean function is a product implicant of this function. (It is not necessarily a prime implicant).

The proofs are omitted. They can be found in [24].

Definition 2. We say that product implicants MIi and MIj are **compatible** when MIi \$ MIj is an implicant of function f, i.e. when there exists no maxterm $Z \in OFF(f)$ such that MIi \$ MIj $\supseteq$ Z. Product implicants MIi and MIj which are not compatible will be called **incompatible.** A set of product implicants CM will be called **a compatible set** when

$$\$_{MIi \in CM} MIi \bigcap OFF(f) = \phi$$

In other words a compatible set is a set of product implicants whose matching is disjoint with the OFF-set of the function.

A set of product implicants CP will be called **a set of compatible pairs** when

$$(\forall \{MI_i, MI_j\} \subseteq CP)\ [(MI_i \$ MI_j) \bigcap OFF(f) = \phi]$$

In other words a set of compatible pairs is one in which a matching of any two product implicants is disjoint with the OFF-set.

Any subset of the set of product implicants included in a prime implicant is then compatible and the matching of any compatible set of product implicants is a product implicant of the function. Any compatible set is also a set of compatible pairs. The opposite statement is however not true, as shown in the following example.

Example 1. The Karnaugh map for function f is given in Fig. 1, where M1 = X00, M2 = 1X0, M3 = 10X, and Z = 011.

We have that M1 \$ M2 = XX0 $\not\supseteq$ Z = 011,
M1 \$ M3 = X0X $\not\supseteq$ Z = 011,
M2 \$ M3 = 1XX $\not\supseteq$ Z = 011,
But M1 \$ M2 \$ M3 = XXX $\supseteq$ Z = 011.

Hence, set {M1, M2, M3} is not a compatible set.

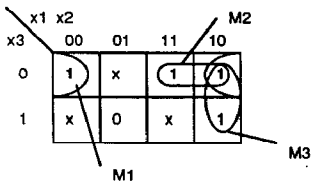


Fig. 1.

The following theorems were used to create the discussed below algorithms.

Theorem 2. Let CPR be a set of cubes covering all minterms and don't cares included in a product implicant PR. Then

$$\$_{C_i \in CPR} = PR$$

This means that matching of all cells of Karnaugh map included in a cube produces this cube.

Theorem 3. Let C be a set of cubes covering cells (0-cubes) with minterms and don't cares. If

$(\forall C_i, C_j \in C)[(C_i \$ C_j) \bigcap OFF(f) = \phi]$

then

1) $PR = \$_{C_r \in C} C_r$ is a product implicant

and

2) $(\forall C_i \in C)[PR \subseteq C_i]$

Theorem 4. If the function f is completely specified, then a set of compatible pairs is also a compatible set.

This theorem is important since it simplifies and speeds up the synthesis of the completely specified functions. This is because for such functions if each pair of product implicants from set S is compatible then matching of all implicants from S is a product implicant of a function.

4. REDUCTION OF TWO LEVEL SINGLE OUTPUT BOOLEAN FUNCTION MINIMIZATION TO THE MINIMAL GRAPH-COLORING PROBLEM.

The purpose of this section is to discuss how the minimization of a single-output Boolean function can be reformulated as a Graph Coloring problem.

Let us create the non-ordered graph GIM = (SMI, RS), where SMI is the set of nodes of the graph corresponding to the cubes of function f and RS is the set of non-oriented edges such that

e = (MI1, MI2) $\in$ RS if and only if MI1 is incompatible with MI2.

This graph will be called "Graph of Incompatibility of Implicants" (GIM). This graph is next colored. A number will be then assigned to each product implicant of the function f. We will call these numbers **the colors of the implicants.** Hence, there exists a coloring function: COLF(SMI(f)) $\rightarrow$ N such that each node of the graph is assigned with a specific color. N is the set of natural numbers.

The property of this function ensures that any two incompatible product implicants will be given two different colors. We will call this the property of "Proper Coloring". Hence,

If MI1 $\in$ SMI(f) & MI2 $\in$ SMI(f) & (MI1,MI2) $\in$ RS
then COLF(MI1) $\neq$ COLF(MI2)

So a Proper Coloring will be defined as one in which different values of the function COLF are assigned to any pair of nodes which are connected by an edge (MI1, MI2) $\in$ RS.

Definition 3. A *Compatible Coloring* is a proper coloring in which each set of nodes of the graph having the same color is a compatible set of implicants of the function.

By finding the compatible coloring of the graph with the minimum number of colors, we minimize the number of compatible sets of implicants, and then the number of prime implicants in the cover.

Theorem 5. The minimal number of compatible sets of product implicants is the same as the number of prime implicants in the minimal cover of the function.

The conclusion of the Theorems 4 and 5 is that for the completely specified functions the proper coloring is the compatible coloring as well, thus the minimum solution to the graph-coloring gives also the *minimum* solution to the single-output optimization problem. Single-output optimization is useful for PALs and EPLDs, since in these structures the products are not shared among sums.

Each output function can be thus minimized separately.

After completing the compatible coloring of graph GIM all the nodes of this graph are assigned to a specific color. If function f is a completely specified function, then by matching those implicants which have the same color we obtain product implicants of the minimal solution. The treatment for incompletely specified functions is slightly different [24] and is not a concern of this paper.

A completely specified function is described by sets ON(f) and OFF(f) alone. One of them is enough and the other one can be found from complementation. Such functions are often met in the current engineering practice at the system design level.

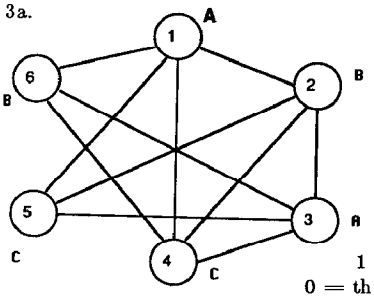
Example 2. Fig. 2 shows the Karnaugh map for a given function: ON (f(X1, X2, X3, X4)) = {0000, 0011, 0100, 1101, 1111, 1011}.

Node 1 = 0000
Node 2 = 0011
Node 3 = 0100
Node 4 = 1101
Node 5 = 1111
Node 6 = 1011

x1 x2 \ X3 x4	00	01	11	10
00	1	0	1	0
01	1	0	0	0
11	0	1	1	0
10	0	0	1	0

Fig. 2.

By matching each pairs of nodes, we create graph GIM from Fig. 3a.



Node	1	2	3	4	5	6
1	0	1	0	1	1	1
2	1	0	1	1	1	0
3	0	1	0	1	1	1
4	1	1	1	0	0	1
5	1	1	1	0	0	0
6	1	0	1	1	0	0

1 = an edge between two nodes.
0 = there is no edge between two nodes.

Fig. 3a. Fig. 3b.

This graph can be represented by a **incompatibility matrix** from Fig. 3b.

The minimum number of colors needed for this graph is three. The coloring with colors A, B and C is shown in Fig. 3a. According to Theorem 5 this means that we can realize the minimal solution for this function with three product implicants. By matching minterms with color A, we get 0000 $ 0100 = 0X00. Similarly, by matching minterms with color B, we get 0011 $ 1011 = X011. Finally, by matching minterms with color C we get 1101 $ 1111 = 11X1.

So, f(X1, X2, X3, X4) = {0X00, X011, 11X1}
or f = $\overline{X1}\cdot\overline{X3}\cdot\overline{X4} + \overline{X2}\cdot X3\cdot X4 + X1\cdot X2\cdot X4$

Discussion:

In the above example, we used minterms to create graph GIM. There are three reasons for it.

1. Graph coloring gives an absolute minimum solution if we start out with minterms. (Graph coloring tries all the combinational ways to match two cubes in the graph. This guarantees that we have searched all possible way of grouping a given cube to the rest of the graph).
2. For larger functions it is impractical to create the graph with minterms. Some of the reasons are: the numbers of minterms are too large, of the order of two to power of N inputs; the graph will then be too large to construct. This suggests that some thing must be done to the input functions before invoking the graph coloring routine.
3. All the terms in the above example are disjoint. In fact, the performance of the Graph Coloring algorithm will be much degraded if the input cubes were non-disjoint.

The nodes of the GIM graph can correspond to minterms, disjoint product implicants, disjoint minimum implicants or minimal product implicants. The advantage of using minterms is a warranty of optimum solution, the disadvantage is the size of the graph. The advantage of using arbitrary disjoint product implicants is the execution speed and simplicity of the algorithm. The advantage of using minimal product implicants [24] is, like for minterms, a warranty of the optimum solution, but the number of such implicants is usually smaller than the number of minterms. For many industrial functions it is more likely to approximate the number of products in the solution. Disjoint minimum implicants also produce minimum solution. In our experiments such implicants produced fastest solutions, but we suspect that with a new procedure to generate minimal product implicants the synthesis based on them will be faster. In this paper disjoint minimum implicants will be used.

Several variants of the algorithm can be created by parametrization of the minimization procedures.

The graph can be created from:
- minterms,
- disjoint product implicants,
- minimal product implicants,
- disjoint minimum implicants.

Function can be realized as:
- sum of product,
- product of sums.

The graph can be colored by using:
- heuristic, non backtracking algorithm for approximate minimization,
- optimal backtracking algorithm for solution with minimum number of implicants.

The solution can be:
- hazardous,
- hazardfree.

The implicants from the solution can be:
- prime implicants (to minimize power, increase reliability and foldability),
- arbitrary product implicants.

These lead to the algorithm PALMINI discussed in the next section.

5. PALMINI.

Description of PALMINI:

- input: cubes (product implicants) of completely specified functions in terms of sum of products. The input cubes can be overlapping.
- output: a minimized version of the function.
- features as options:
  1. Form of input cubes for Graph Coloring.
  2. Optimal and quasi-optimal Graph-coloring algorithms.
  3. Invert the polarity of the output.
  4. Check for Static Hazards for combinatorial outputs.
  5. Minimize the number of literals in each term of the function.

5.1. Main procedures of PALMINI.

procedure COMPL(SMI);

This procedure returns the complementation of the input function contained in SMI. The Disjoint Sharp method is currently employed. At the end of each loop, the list OFF is passed to procedure ABSORBE to delete redundant terms.

procedure CREATEDISJOINT(SMI);

This procedure receives inputs from input sets SMI. It then returns a set of disjont cubes back into set SMI. The algorithm is as follows:

```
for i = 1 to (last cube in SMI - 1)
begin
 for j = i + 1 to last cube in SMI
 begin
  if cubei intersects cubej then
  begin
   list D = cubei # cubej;
   cubej is deleted from SMI;
   list D is added to SMI;
  end;
 end;
end;
```

procedure CREATEMINIMAL(SMI);

This procedure is used to create disjoint minimum product implicants. In general, only implicants of this type (or ones included in them, like minterms) assure the minimum solution if the solution to graph coloring problem is also optimal.

1. Find all consensuses of cubes from SMI and add them to the set SMI.
2. Find all products of pairs, pairs of pairs, pairs of pairs of pairs, ... etc. of cubes from SMI; remembering for each new product cube the product cubes that it originates from. This is done in the form of the (directed, acyclic) graph. An arrow points from cube1 to cube2 if cube2 originates from cube1.
3. Remove from the tree all cubes, that are cube unions of other cubes from the graph. This is done from top to bottom of the graph (starting from largest cubes).
4. Remove from the tree all cubes that are included into single cube only.
   The remaining cubes in the tree are the disjoint minimum implicants. Return them as a value of CREATEMINIMAL.

Example 3. For function f(a, b, c, d) = {0X01, X1X1, 011X, 1100, 1011} the consensuses are {110X, 1X11, 01X1}. The products of cubes are {0101, 0111, 1101, 1111}. After removal of products being unions of other products the set SMI is = {1100, 1011, 011X, 0X01, 0101, 0111, 1111, 1101}. After removing of cubes that are included into only one cube the set SMI = {1100, 1101, 1111, 1011, 011X, 0X01}. This set is used to create graph GIM.

procedure GRAPH(SMI, OFF, GIM);

This procedure will construct graph GIM from disjoint set SMI and set OFF which contains the complementation of the input function.

The algorithm is as follows:

```
for i = 1 to (last cube in SMI - 1)
begin
 for j = 1 to last cube in SMI
 begin
  if (cubei $ cubej) ∩ OFF ≠ φ then
   GIM(i,j) = GIM (j,i) = 1
      {an edge exists between node i and node j}
  else GIM(i, j) = GIM(j, i) = 0;
      {no edge exists between node i and node j}
 end;
end;
```

procedure COLOR(GIM, cost1);

This paper presents only a non-backtracking, approximate algorithm. For optimal graph-coloring algorithms see [24], [15], [18].

This procedure will use GIM as its input and return cost1 as the number of colors needed to color this graph. The method to assign a new color is as follows: the first node is assigned color 1. For any following node j, we first assign color 1 as a temporary color for that node j. Then starting from node 1, we search for edges between current node j and previous nodes. We only assign the next color to node j if there is an edge between the current node j and a previous node and only if the color of this previous node is the same as that of the current node j. This helps us eliminate the backtracking portion. This procedure is approximate, but gives good results.

procedure DELETELITERAL(SOL, OFF);

This procedure takes each term in SOL and tries to remove as many redundant variables as possible according to the following algorithm:

```
for i = 1 to last cube in SOL
begin
 for j = 1 to max number of input variables
 begin
  temp = cubei[j];
  cubei[j] = X;
  if cubei ∩ OFF ≠ φ then
   cubei[j] = temp;
 end;
end;
```

5.2. Hazardless minimization.

Product implicants PI1 and PI2 are adjacent when they include two minterms, m1 ∈ PI1 and m2 ∈ PI2 such that m1 and m2 differ in a single bit only (are adjacent in a sense of a Gray code). The static hazard in ones occurs in a two-level circuit when there are two ANDs realizing adjacent product implicants but lacking a third product to cover the adjacent minterms of the two products. The result of such hazards is the glitch (short pulse zero) in the output before it reaches the stable state 1.

Example 4. Let us assume a two-level realization of an expression: $f = \bar{a}\cdot\bar{c}\cdot d + a\cdot b\cdot\bar{c} + \bar{a}\cdot b\cdot c + a\cdot c\cdot d$ Assume all the gates have the same delay "tpd". The pair of cells 0101 and 0111 is a pair of adjacent minterms not covered by a single implicant. So are also the pairs: 0111 and 1111, 1111 and 1101, 1101 and 0101. This is then a circuit with four static hazards (this is the same function as in Example 3). Depending on the later stages of the circuitry, these glitches may cause catastrophic failures to the rest of the operation of the circuitry (for instance if hazard occurs in a feedback loop of an asynchronous circuit or if a counter is driven from a circuit with hazard).

By introducing a fifths cube to cover the adjacent 1's between the original product implicants, we effectively eliminate all four hazards. Solution: $f = \bar{a}\cdot\bar{c}\cdot d + a\cdot b\cdot\bar{c} + \bar{a}\cdot b\cdot c + a\cdot c\cdot d + bd$ is then hazardless.

One of the features of PALMINI is the ability to correct all the static hazards that exist in the solution. After the solution is obtained from the Graph Coloring Algorithm and if the hazardless option is selected, PALMINI will compute all the consensuses which exist among the cubes in SOL. Next it will find all mergings (distant-one merge groups $AB + A\bar{B} = A$) of consensuses and of consensuses and product implicants. This operation is repeated until no more groups are created. It will then remove consensuses properly included into mergings. The consensuses and mergings are attached to SOL as a part of the final solution.

Below we will present the algorithmic way to find the all hazard eliminating cubes. Consensus of two cubes A and B is created as follows. First we calculate the bit-by-bit operation star (*) on cubes A and B. Star operation per bit is defined in Table 1.

*	0	1	X
0	0	e	0
1	e	1	1
X	0	1	X

Table 1.

Next if the resultant cube includes exactly one e, it is changed to X. Otherwise the cube is not a consensus of the function.

Example 5. From the example 4 above, we have
cube1 * cube2 = 0X01 * 011X = 01e1 = 01X1.

Note: 01e1 contains only one "e". Therefore, it can be changed to "X". There are four consensuses in this example: 01X1, 11X1, X101 and X111. Merging of 01X1 and 11X1 produces cube X1X1. All consensuses are now removed since they are covered by this cube. This leads to a hazardless solution from Example 4.

procedure HAZARDLESS(SOL);
1. {Find the set of all consensuses $cube_c$ of
cubes from solution SOL}
for i = 1 to (last cube in SOL - 1)
begin
for j = (i + 1) to last cube in SOL
begin
$cube_c = cube_i * cube_j$;
{if there is no result of consensus operation $cube_c$ is an empty set}
if $cube_c$ is not empty and $cube_c \notin$ SOL
then add $cube_c$ to SOL;
end;
end;
2. {Find the set NEW_CUBES of all cubes $cube_m$ being results of
merging operations ($cube_m = cube_i$ m $cube_j$)
of all $cube_i$ and $cube_j$ from SOL}
for i = 1 to (last cube in SOL - 1)
begin
for j = (i + 1) to last cube in SOL
begin
$cube_m = cube_i$ m $cube_j$;
{ m is a merging operator, if cubes do not merge
the result $cube_m$ is an empty set}
if $cube_m$ is not empty and $cube_m \notin$ SOL
then
begin
add $cube_m$ to SOL; add $cube_m$ to NEW_CUBES;
end;
end;
end;
3. NEW_CUBES = MERGINGS(NEW_CUBES, SOL);
if NEW_CUBES = ϕ
then
return SOL = SOL with removed cubes
included in other cubes of SOL;
else
begin
SOL = SOL $\cup$ NEW_CUBES;
go to 3
end;
function MERGINGS(NEW_CUBES, ALL_CUBES);
NEW_CUBES = ϕ;
for i = 1 to last cube in NEW_CUBES
begin
for j = 1 to last cube in ALL_CUBES
begin
$cube_m = cube_i$ m $cube_j$;
if $cube_m$ is not empty and $cube_m \notin$ SOL
then add $cube_m$ to NEW_CUBES;
end;
end;
return NEW_CUBES;

5.3. Flow chart of PALMINI. (variant for a single-output completely specified function).

Get input: set SMI <- sum of products.

1. Find the complementation from this set:
OFF <- COMPL(SMI).
2. If invert polarity option is selected then begin
SMI <- OFF.
OFF <- SMI. end.
3. If createdisjoint variant selected
then create disjoint set from SMI:
SMI <- CREATEDISJOINT(SMI)
else
create minimal set from SMI:
SMI <- CREATEMINIMAL(SMI).
4. Create graph GIM: GIM <- GRAPH(SMI, OFF, GIM).
5. Color graph GIM to find cost:
cost <- COLOR(GIM, cost1);
6. Find solution and store in array SOL.
7. If the Static Hazardless option is selected
then SOL <- HAZARDLESS(SOL).
8. If the literal delete option is invoked then
begin
delete redundant literals in each term of solution SOL.
SOL <- DELETELITERAL(SOL, OFF).
end.

Solution is now contained in SOL.

6. PERFORMANCE EVALUATION.

We have implemented three versions of PALMINI. The first of them was written in Fortran [32], the next in Turbo Pascal, the last one is in C. The last version has the improved data structure for arrays of cubes. In the Pascal version all bits in the cubes are realized as elements of two dimensional arrays. Thus all the operations operate on arrays. The C version uses computer words (registers) to represent cubes which is a much better approach. We have tried about thirty examples ranging from 4 terms/4 inputs to 20 terms/18 inputs. Solutions were compared to those of LOGMIN and were the same.

A set of 9 examples is used to demonstrate the discussed here variant of algorithm. All tests were done on PC XT compatible machine with 8 MHz clock. We have compared it with minimizers in ALTERA EPLD software, DATA I/O ABEL, which uses PRESTO algorithm, and Espresso, all running on the same machine. The algorithm used in ALTERA sotware is an order of magnitude slower than PALMINI and is not shown here. On the other hand, Presto from ABEL is very reasonable. The version used is 1.1 which is much better than version 1.04. Depending on the types of functions, sometimes, PALMINI is faster than Espresso and sometimes it is not.

In the following table the numbers of terms and input variables are given for each example. Next, the times (in seconds) and numbers of terms in solutions are given for PALMINI, ABEL and Espresso.

Ex#	Function		PALMINI		ABEL 1.1		ESPRESSO	
	Trm	Inp	Time	Term	Time	Term	Time	Term
1	19	6	2	13	12	13	4.5	14
2	8	9	2	4	5	4	2	4
3	15	9	1	4	9	4	2	4
4	10	10	1	10	12	10	3	10
5	10	11	2	9	8	9	3	9
6	12	12	2	10	27	10	3	10
7	13	13	4	10	26	10	5	10
8	11	17	9	10	7	10	4	10
9	20	18	6	17	--	--	8	17

-- no answer in 20 minutes.

The time under the ESPRESSO column is hand calculated. The version of Espresso-mv for PC XT that we have does not permit to measure time. Also, we were not able to use -do exact and -do single_output parameters described in [27]. The above examples come from practical industrial PLAs from Intel. We were not able to compare with functions provided with Espresso since all the functions we dispose have more than one output. When compared on randomly generated large functions with product implicants of many literals the results of PALMINI were often worse than those of Espresso. More testing is necessary on large single-output industrial PLAs.

With the current version of the program:
-- 60 % of the time is to compute the complementation.
-- 20 % of the time is to compute disjoint cubes.
-- 10 % of the time is to compute graph GIM.
-- 5 % of the time is to color the graph.
-- 5 % of the time is to delete the literals.

The fact that 60 % of the time is used to compute the complementation suggests that by having a better algorithm such as the one used in Espresso, the speed of the program can be much further improved. Currently, the Disjoint Sharp method is employed. In fact, the Disjoint Sharp method is not too bad in this case, when applied to PAL's, because PAL can have many inputs, 15 or 16, but the outputs can only have at most eight OR terms except for the case of C types or PLD types.

In general, we will have a very limited number of OR terms per each output when using PAL's or PLD's. So, we will not design many OR terms per each output. This will allow the Disjoint Sharp method to be manageable because the performance of the Sharp operator is proportional to the number of inputs and terms. We are currently working on a different algorithm to find the complementation of a function. Hopefully, it will enhance the program even more. Some of the good approaches to complementation are described in [4], [29].

7. CONCLUSION AND FUTURE WORK.

The discussed above examples were taken from examples at work. PALMINI has shown us that it indeed gives good solutions within an acceptable time frame. Besides the fact that its speed on functions of small size is comparable or better than ESPRESSO and many times faster than ABEL (Presto), it has an useful feature which other low cost minimizers do not have. That is Static Hazard correction. PALMINI is easily recompiled to be run on various personal and home computers which support standard PASCAL or C like IBM PC, APPLE, Commodore, and etc. The compiled code is small. It can easily fit into 64K of memory. This includes all the code and data areas, which permits to use this program together with other memory-resident programs. Executable code of PALMINI is only 30K, versus 177K of Espresso. The limit of the described here version 2.0 are as follows:
- up to 64 input variables,
- up to 60 product terms (due to memory size limits),

A variants to design multi-output functions and incompletely specified functions have been already implemented. We are working [24] on various extensions to PALMINI like:
- multi-valued logic (including application to FSM assignment method from [17],
- more product terms,
- multioutput, multilevel functions (PLA and PAL partitioning),
- improving complementation algorithm by replacing current disjoint sharp menthod with a new algorithm,

Graph coloring can be approximated by a maximum clique problem and the Maximum Independent Set Problem can be used for finding the approximate solution [12]. Although the exact algorithm does not search for prime implicants, it is still able to find the optimum solution for small and medium size functions. For medium size functions we are able to evaluate how far is our solution from the minimum one, even without generating this solution. This is done by approximating of graph coloring by Maximum Independent Set Problem. In some cases the optimum solution can be proven with limited search.

The minimal implicants is the only known to us canonical representation of Boolean function other then sum of minterms or sum of all prime implicants. They also generalize the concept of essential implicants and permit to simplify the function before graph coloring. Unfortunately the tested by us algorithms for generation of minimal implicants were slow. However, recently we found a new method to generate such implicants whose implementation is hoped to be faster.
The advantage of reducing to graph coloring is also that the graph coloring problem has been thoroughly investigated by computer scientists and very fast approximate and optimum algorithms for it have been created. Our ultimate goal is to design parallel algorithms and special hardware for Boolean minimization [23]. We find the reduction to graph coloring a useful approach, since parallel and parallel/probabilistic algorithms for this problem are known and special hardware has been proposed.

We can summarize the limitations of our method as follows:

1. The reduction and coloring algorithms are fast. The weakest part is the complementation. Two improvements are possible:
   1) better complementation algorithm,
   2) avoid complementation and check inclusion of matchings into ON instead of checking intersection of matchings with OFF while creating the GIM graph.
   The time to create a graph grows with the second order of number of implicants used as nodes.
2. We feel that many limitations of the program result more from the ways of implementation then from the method itself. The quality of future versions of PALMINI will depend on the quality of graph coloring algorithms.

Both the complementation and the graph coloring can be solved with tree-searching algorithms. When depth-search is used small computer memory is sufficient. Also, these methods permit for simple parallelization. Implementation on Intel's hypercube computer is planned.

The basic idea of our approach - reducing the covering type problems to graph coloring seems to be very general in logic design. Using slightly different rules for compatibility of implicants the approach can be also used for minimization of three-level networks (TANT) [21], two-level networks from negative gates and multi-level circuits realized from PLAs and PALs.

8. LITERATURE.

[1] **Bartholomeus, M., Man, H.D.,** : "Prestol-II: Yet Another Logic Minimizer for Programmed Logic Arrays", Proc. Int. Symp. Circ. Syst., June 1985, p. 58.

[2] **Besslich, P.W., Pichlbauer, P.** : "Fast Transform Procedure for the Generation of Near-Minimal Covers of Boolean Functions", IEE Proc. Vol. 128, PTE, No.6, Nov. 1981, pp.250-254.

[3] **Biswas, N.N.** : "Computer-Aided Minimization Procedure for Boolean Functions", Proc. 21st Design Automation Conf. pp. 699-702, June 1984.

[4] **Brayton, R. K., Hachtel, G. D., McMullen, C. T., and Sangiovanni Vincentelli, A. L.** : "Logic Minimization Algorithms for VLSI Synthesis." Kluwer Academic Publishers, 1984.

[5] **Breitbart, Y., Vairavan, K.** : "The Computational Complexity of a Class of Minimization Algorithms for Switching Functions", IEEE TC, Vol. C-20, No. 12, December 1979, pp. 941-943.

[6] **Breuer, M.A. (ed)** : "Design Automation of Digital Systems", Vol. 1, Prentice Hall, Englewood Cliffs, New Jersey, 1972.

[7] **Brown, D. W.** : "A State Machine Synthesizer-SMS", 18th Design Automation Conference, p.301, June 1981.

[8] **Caruso, G.** : "A Local Selection Algorithm for Switching Function Minimization", IEEE TC., Vol. C-33, Jan. 1984, pp. 91-96.

[9] **Dietmeyer, D. L.** : "Logic Design of Digital Systems", Allyn and Bacon, Boston, 1971.

[10] **McCluskey, E. J.** : "Minimization of Boolean Functions", Bell Syst. Tech. J., Vol.35, p.1417, April 1956.

[11] **Dagenais, M.R., Agarwal, V.K., Rumin, N.C.** : "McBoole: A New Procedure for Exact Logic Minimization", IEEE Trans. on CAD, Jan. 1986, pp. 229 - 238.

[12] **Garey, M., Johnson, D.** : "Computers and Intractability. A Guide to the Theory of NP-Completeness.", W.H. Freeman and Company, San Francisco 1979.

[13] **Gimpel, J.F.** : "The Minimization of TANT Networks", IEEE TEC, Vol. EC-16, pp. 18-38, February 1967.

[14] **Hong, S.J., Cain, R.G., Ostapko, D.L.** : "MINI: A Heuristic Approach for Logic Minimization", IBM J. Res. and Dev., Vol.18, No.5, pp. 443-458, September 1974.

[15] **Johnson, D.S.** : "Worst-Case Behavior of Graph Coloring Algorithms". Proceedings of the Fifth Southeastern Conference on Combinatorics, Graph Theory and Computing, pp. 513-528, Winnipeg, Canada, Utilitas Mathematica Publishing, 1974.

[16] **Kerntopf, P., Michalski, A.** : "Selected Problems in Synthesis of Combinatorial Logic Circuits", PWN, Warsaw, 1972 (in Polish).

[17] **Lee, E.B., Perkowski, M.** : "Concurrent Minimization and State Assignment of Finite State Machines". Proceedings of 1984 Intern. Conf. on Systems, Man and Cybernetics, IEEE, Halifax, Nova Scotia, Canada, October 9 - 12, 1984.

[18] **McDiarmid, C.J.H.** : "Determining the Chromatic Number of a Graph", SIAM J. Computing, Vol. 8, pp. 1 - 19, 1979.

[19] **McMullen, C., Shearer, J.** : "Prime Implicants, Minimum Covers, and the Complexity of Logic Simplification", IEEE TC, February 1987.

[20] **Miller, R.E.** : "Switching Theory", Vol. 1 and 2, John Wiley, New York, 1965.

[21] **Perkowski, M.** : "Synthesis of Multioutput Three Level NAND Networks", Proceedings of Seminar on Computer Aided Design, pp. 238 - 265, Budapest, November 3 - 5, 1976.

[22] **Perkowski, M.,** : "A General Method to Solve Combinatorial Problems in the Automatic Design of Digital Systems", Publishers of Warsaw Technical University, Warsaw, Poland, September 1980 (in Polish), 434 pages.

[23] **Perkowski, M.** : "Systolic Architecture for the Logic Design Machine", Proc. Intern. Conf. on Computer Aided Design, pp. 133 - 135, Santa Clara, November 1985.

[24] **Perkowski, M.A., Nguyen, L.B., Goldstein, N.B.** : "An Approach to Minimization, Decomposition and Partitioning of PLA and PAL Circuits Based on Multi-Valued Algebra and Graph Coloring", Technical Report, Dept EE, PSU 1987.

[25] **Petrick, S.R.** : "On the Minimization of Boolean Functions", Proceedings of Symp. on Switching Theory, ICIP, Paris, France, June 1959.

[26] **Rudell, R., Sangiovanni-Vincentelli, A.,** : "Espresso-mv: Algorithms for Multiple-Valued Logic Minimization", Proc. 1985 Custom Integrated Circuits Conference, pp. 230 - 234, Portland, OR, May 1985.

[27] **Rudell, R.** : "Multiple-Valued Logic Minimization for PLA Synthesis", Research report, June 5, 1986, Univ. of California, Berkeley.

[28] **Rudell, R., Sangiovanni-Vincentelli, A.,** : "Exact Minimization of Multiple-Valued Functions for PLA Optimization", Proc. ICCAD-86, pp.352 - 355, Santa Clara, CA, November 1986.

[29] **Sasao, T.** : "An Algorithm to Derive the Complement of a Binary Function with Multiple-Valued Input", IEEE Trans. on Comput., Vol. C-34, No.2, pp.131-140, Febr. 1985.

[30] **Slagle, J.R., Chang, C.L., Lee, R.C.T.** : "A New Algorithm for Generating Prime Implicants", IEEE TEC, Vol. EC-19, pp. 304-311, April 1970.

[31] **Svoboda, A.** : "The Concept of Term Exclusiveness and Its Effect on the Theory of Boolean Functions", Journal of ACM, Vol. 22, No.3, July 1975, pp. 425 - 440.

[32] **Wojutynski, J.** : "A General System to Solve Combinatorial Problems by Reduction to Graph Coloring", M.Sc. Thesis, (M. Perkowski superv.), Institute of Automatic Control, School of Electronic Engineering, Warsaw Technical University, May 1979, (in Polish).

Some commercial minimizers currently running on PC's are:

ABEL: a program from DATA I/O Corp.
CUPL: a program from Assisted Technology.
A+ : a program from ALTERA Corp.

Some minimizers currently running on VAX or mainframe are:

Espresso, Presto, Prestol-II, Mini, Spam, and McBoole.

The authors would like to thank an anonymous referee for important remarks.