Multiple–Valued Boolean Minimization Based on Graph Coloring

Maciej J. Ciesielski Saeyang Yang
Dept. of Electrical & Computer Engineering
University of Massachusetts
Amherst, MA 01003

Marek A. Perkowski
Department of Electrical Engineering
Portland State University
Portland, OR 97207

Abstract

A new method for the minimization of multiple–valued input Boolean functions is presented. The method is based on the reduction of logic minimization problem to graph coloring, applied to the graph of incompatibility of implicants. In this approach, two NP-complete problems encountered in the minimization of Boolean functions, i.e., the generation of prime implicants and the covering problem, are reduced to a single, and better understood, graph coloring problem. A special type of implicants, called Minimally Split Product Implicants, are generated from an arbitrary set of input cubes that allow optimum results to be obtained. An important result of this method is that it is analytical, rather than heuristic, and gives more insight into a larger class of logic synthesis problems, such as input encoding and Boolean decomposition.

1. Introduction

Minimization of multiple-valued input Boolean functions is of great practical importance in modern synthesis of digital VLSI circuits. Its applications include minimization of PLAs with input decoders [Sas81, Sas84], symbolic minimization of logic functions [DeM86], and Boolean decomposition of PLAs [Dev88, Yan89a, Yan89b].

The logic minimization programs available today, such as MINI [Hon74] and Espresso–MV [Rud87], are based on heuristic iterative improvement techniques, and, in general, give suboptimum solution with no information on how far it is from the global minimum. Furthermore, if the input is a set of arbitrary cubes, instead of minterms, the solution depends on the input description. The minimization technique presented in this paper is based on the reduction of logic minimization problem to graph coloring [Ngu87], extended to multiple–valued input, incompletely specified logic functions. A special set of implicants is generated from an arbitrary set of input cubes. The relations among the implicants in this initial cover are represented by a graph of incompatibility of implicants. By coloring the graph nodes with minimum number of colors the minimum cover of the Boolean function is obtained. This method can be applied to single and multiple output functions, both completely and incompletely specified.

2. Basic Definitions

Let X_i be a multiple–valued variable, and $P_i = \{0, 1, ..., p_i - 1\}$ be a set of values it may assume. A multiple–valued Boolean function with n inputs is defined as:

$$f(X_1, ..., X_n) : P_1 \times P_2 \times \cdots \times P_n \rightarrow B,$$

where $B = \{0, 1, *\}$ represents the binary value of the function. Let $S_i \subseteq P_i$. Then $X_i^{S_i} = 1$ if $X_i \in S_i$, otherwise it is 0. $X_i^{S_i}$ is called a *literal* of variable X_i. Boolean product of literals is called a *product term* or a *cube*.

Example 1: Cube $C = X_1^{\{0,1,3\}} X_2^{\{0,2\}} X_3^{\{1\}}$ represents a product term of three four–valued variables, X_1, X_2, X_3. It evaluates to 1 if ($X_1 = 0$ or $X_1 = 1$ or $X_1 = 3$) and ($X_2 = 0$ or $X_2 = 2$) and ($X_3 = 1$); otherwise it evaluates to 0. In the positional cube notation this cube can be written as {1101–1010–0100}.

We assume that the reader is familiar with such basic definitions as Boolean cover, the $ON(f), OFF(f)$ and $DC(f)$ sets, prime implicants, essential prime implicants, etc. We also assume readers familiarity with basic operations on cubes and arrays of cubes, such as: union, intersection, disjoint sharp, and consensus. Formal definitions of these terms can be found in [Rud87]. Here we only define those terms which are central to our algorithm.

Definition 1: Let $C_1 = X_1^{S_1} \cdots X_n^{S_n}$ and $C_2 = X_1^{T_1} \cdots X_n^{T_n}$ be cubes, where $S_i, T_i \subseteq P_i$, and n is the number

CH2794-6/89/0000/0262$01.00 © 1989 IEEE

of input variables. The *matching*, or *supercube*, of C_1 and C_2, denoted $C_1\$C_2$, is the cube $C = X_1^{S_1 \cup T_1} \ldots X_n^{S_n \cup T_n}$. It is the smallest cube containing both C_1 and C_2. The matching operation is commutative and associative, and the result is always a cube.

Definition 2: Two product implicants I_1 and I_2 of function f are *compatible* if $I_1\$I_2$ is also an implicant of function f, i.e., if $(I_1\$I_2) \cap OFF(f) = \emptyset$. Product implicants which are not compatible are called *incompatible*. A set of product implicants PI is called a *set of compatible pairs* when

$$\forall\{I_i, I_j\} \subseteq PI \quad (I_i\$I_j) \cap OFF(f) = \emptyset.$$

A set of product implicants PI is called a *compatible set* when

$$(\$_{I \in PI} I) \cap OFF(f) = \emptyset.$$

Notice that any compatible set is also a set of compatible pairs, but the opposite is not necessarily true.

3. Multiple–Valued Minimization

The objective of multiple–valued minimization is to determine a minimum cardinality cover of a given multiple–valued input Boolean function. In this section we formulate this problem as a graph coloring problem.

Let $GI(f) = (PI, E)$ be a graph whose set of nodes PI represents product implicants of function f, and whose set of edges E represents incompatibility relations between the corresponding pairs of cubes, such that $e = (I_1, I_2) \in PI$ iff I_1 is incompatible with I_2. The graph $GI(f)$ is called a *Graph of Incompatibility of Implicants*.

Definition 3: A *proper coloring* of a graph is the coloring of its nodes such that no two adjacent nodes are assigned the same color. A *compatible coloring* of the graph $GI(f)$ is a proper coloring such that each set of nodes with the same color represents a compatible set of product implicants.

Theorem 1: The minimum number of compatible sets of product implicants of function f is equal to the number of prime implicants in the minimal cover of f.

Proofs of the theorems can be found in [Cie89].

An important conclusion from this theorem is that finding a minimum cardinality cover for a given function f is equivalent to finding a compatible coloring of its graph $GI(f)$ with the minimum number of colors. Furthermore, the minimum number of colors obtained by the proper coloring of $GI(f)$ gives lower bound on the minimum solution.

4. Minimally Split Product Implicants

In general, the nodes of graph $GI(f)$ may represent an arbitrary set of input cubes. If the cubes are minterms, then an optimum compatible coloring of $GI(f)$ is also an optimum solution to the logic minimization of function f. The reason for this is that the minterms can be grouped in all possible ways to form other cubes, and the optimum coloring gives the grouping with minimum number of implicants. For large functions, however, it is impractical to create a graph whose nodes represent minterms, as their number can be in the order of 2^n. If the on-set $ON(f)$ consists of arbitrary cubes, rather than minterms, the resulting graph $GI(f)$ will have smaller number of nodes. This decreases the complexity of the minimization problem and speeds up graph coloring. However, the optimality of the resulting solution cannot be guaranteed in this case.

We introduce a new class of product implicants, called *Minimally Split Product Implicants*, MSI. The MSI set is obtained by splitting an initial set of cubes INC into a smallest possible set of fundamental cubes from which a minimum cover can be obtained. The main idea behind the splitting is to reshape a set of cubes in the initial cover into a new set of cubes from which all prime implicants, and therefore all Boolean covers of the function, can be constructed. As a result, the cardinality of the MSI set is significantly smaller than that of minterms, while it allows to obtain a globally minimum cover.

Definition 4: Consider a cube C and a set of cubes $S = \{S_i\}$. Cube C is called *splittable* with respect to set S, if

$$C = \bigsqcup_{S_i \neq C} S_i.$$

The procedure to create a set of Minimally Split Product Implicants from an arbitrary set of input cubes INC is outlined below. Each cube in the initial cover of f is examined by computing the consensuses ($CONS$) of this cube with all other cubes, and checking if the cube is splittable. The splitting is repeated recursively, until the cubes cannot be split any further.

```
create_msi(INC)
    Initialize: MSI = ∅.
    Make cubes in INC mutually disjoint;
    For each cube CC ∈ INC do:
        Find a set of consensuses, CONS, of CC with INC;
        Find a set of products, PROD, of CC with CONS;
        If CC is splittable w.r.to PROD then
            MSI = MSI ⊔ split_product(PROD)
        else MSI = MSI ⊔ CC.
    end{for each}
    Return (MSI).
```

split_product($PROD$)
  Initialize: $R = \emptyset$.
  For each cube $EC \in PROD$ **do:**
    Find a set of products, $APROD$, of EC with $PROD$;
    If EC is splittable **then**
      $R = R \sqcup$ *split_product*($APROD$)
    else
      if $\exists$ cube C, splittable, s.t. $EC \in C$ **then**
        $R = R \sqcup EC$
  end{for each}
  Return (R).

Example 2: Consider the multiple–valued input function $f = X^{\{0,2,3\}}Y^{\{0\}} + X^{\{0,1,3\}}Y^{\{1\}} + X^{\{1\}}Y^{\{2\}} + X^{\{2\}}Y^{\{3\}}$. Fig. 1(a) shows the input set of cubes, INC, that covers the function. Fig. 1(b) - (d) illustrate the splitting of cube a, and Fig. 1(e) - (g) show the splitting of the remaining cubes, b, c, and d, respectively. The resulting MSI set is shown in Fig. 1(h).

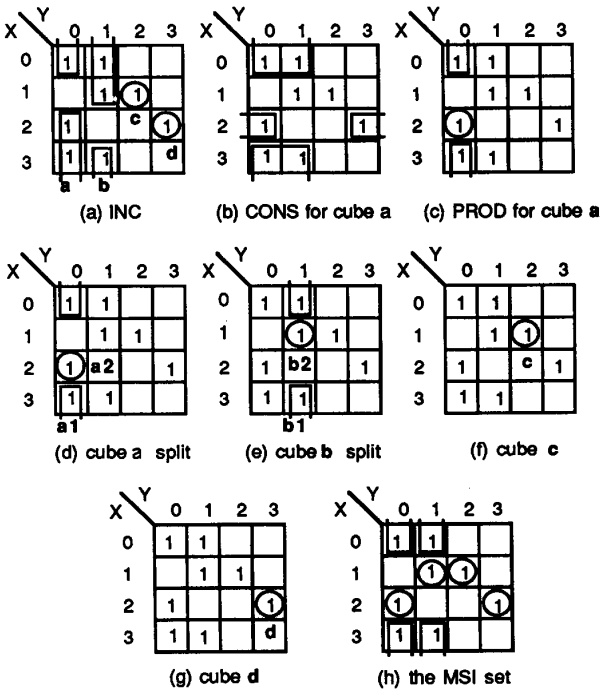

Figure 1: Generation of the MSI set.

In general, the set of cubes created with this algorithm may not be disjoint. As a result, its cardinality may be smaller than that of the Disjoint Minimal Product Implicant set introduced in PALMINI [Ngu87]. This, in turn, leads to a faster graph coloring and, consequently, to a faster logic minimization.

Finally, we have the following theorem which forms the basis for our multiple–valued logic minimization. It holds for both completely and incompletely specified functions.

Theorem 2: An optimum solution to a compatible graph coloring of graph $GI(f)$, whose nodes correspond to a set of Minimally Split Product Implicants of function f, gives a minimum cardinality cover of f.

5. The Algorithm

Input to the algorithm is an arbitrary set of cubes specified by the sets $ON(f)$ and $DC(f)$. The $OFF(f)$ set is then obtained from the two sets by complementation. The cubes in $ON(f)$ are expanded to larger cubes to reduce complexity of the procedure. During the expansion process, the off–cubes, dc–cubes and the expanded on–cubes are used to block further expansion. Optionally, the dc cubes may also be expanded. A set of Minimally Split Product Implicants and the corresponding graph of incompatibility of implicants, $GI(f)$, is constructed. Our graph coloring algorithm of $GI(f)$ preserves compatibility of implicants and allows for multicoloring, whereby each node can be assigned more than one color. As a result, the cubes in the final cover may overlap with each other, which further contributes to the minimization of literal count.

Example 3: Consider again the function f in Example 2. The set of Minimally Split Product Implicants of f is shown in Fig. 2(a); Fig. 2(b) shows the graph $GI(f)$ constructed with this set of implicants. This graph can be colored with three colors, that correspond to the three implicants in the minimum cover of f, shown in Fig. 2(c)

$$f = X^{\{0,3\}}Y^{\{0,1\}} + X^{\{1\}}Y^{\{1,2\}} + X^{\{2\}}Y^{\{0,3\}}.$$

Each implicant is a compatible set of cubes, and the solution is optimum. For comparison, if the function were minimized using the INC set, instead of MSI, the minimum number of colors needed to properly color the $GI(f)$ graph would be four.

6. Results and Conclusions

The algorithm has been implemented as a computer program UMini, written in C. The input to the program is consistent with Espresso-MV. UMini has been tested on a number of benchmark examples, and compared to Espresso-MV. The preliminary results are given in Table I, showing the CPU time on a VAX 11/785 computer. The last column in the table indicates the ratio of the number of Minimally Split Product Implicants to the number of minterms for a given function. Notice that for large functions this ratio is significantly smaller than 1. These results indicate that UMini is faster than

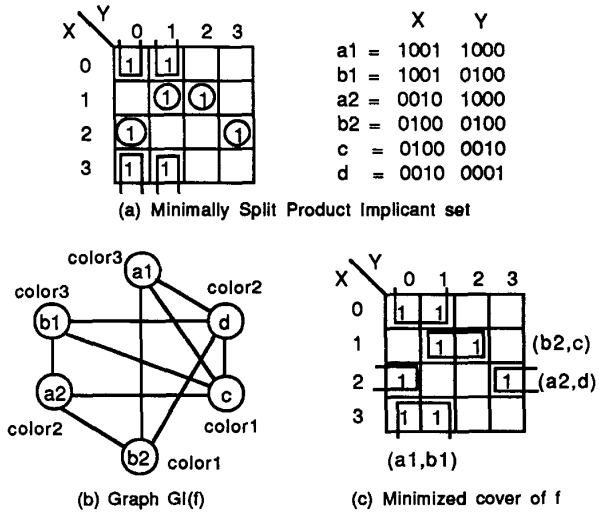

Figure 2: Minimization with the MSI set.

			UMini		Espresso-MV		
	i/o	init. terms	final terms	time sec	final terms	time sec	ratio
dec	5/1	19	9	0.2	10	0.2	0.684
pla1	2/2	8	5	0.3	6	0.3	0.600
pla2	8/10	43	33	7.0	33	67.9	0.025
pla3	8/3	19	12	1.1	12	19.9	0.009
fsm1	1/3	14	10	0.1	10	0.3	0.714
fsm2	2/4	28	13	1.1	13	1.5	0.058
fsm3	1/2	12	10	0.1	10	0.3	0.833
fsm4	2/2	28	20	2.1	20	2.1	0.213
fsm5	2/5	40	32	1.9	32	2.2	0.800
pla4	10/1	197	197	11.8	197	17.2	1.000
exor6	6/1	31	31	0.4	31	0.8	1.000
newbyte	5/8	8	8	0.1	8	0.2	1.000
co14	14/1	14	14	0.4	14	0.8	1.000
bbtas	2/2	24	2	0.1	2	0.2	0.500
dk14	3/5	56	27	2.2	27	2.4	0.297
s8	4/1	20	17	0.3	17	0.5	0.850
dk512	1/3	30	12	0.2	12	0.3	0.923
bbara	4/2	60	6	0.2	6	0.2	0.222
ex6	5/8	34	25	2.6	25	2.6	0.058
wim	4/7	10	9	1.5	9	1.2	0.584
dc1	4/7	15	10	1.5	9	0.8	0.638
rd73	7/3	147	127	64.2	127	17.2	0.740
t3	12/8	148	33	19.7	33	3.9	0.017

Table 1: Results

Espresso-MV for sparse functions, but slower for dense functions. We also observed that UMini was faster than Espresso-MV on functions with multiple-valued, rather than binary, inputs.

We are currently working on a new version of UMini. In this new program essential and secondary essential prime implicants are extracted before generating graph $GI(f)$, and a set of input cubes is allowed to be nondisjoint.

Our approach to multiple-valued minimization, presented in this paper, is quite general and can be used for a larger class of logic synthesis problems, such as state encoding, boolean decomposition, and multi-level minimization Existence of parallel and distributed algorithms for graph coloring makes this approach particularly attractive.

References

[Cie89] M.J. Ciesielski, S. Yang and M.A. Perkowski, "Multiple-Valued Minimization Based on Graph Coloring", Technical Report, TR-89-CSE-4, Dept. of Electrical and Computer Engineering, University of Massachusetts, Amherst, 1989.

[DeM86] G. De Micheli, "Symbolic Design of Combinational and Sequential Logic Circuits Implemented by Two-Level Logic Macros", IEEE Transactions on Computer-Aided Design, Vol. CAD-5, No. 4, Oct. 1986, pp. 597–616.

[Dev88] S. Devadas, A.R. Wang, A.R. Newton, and A. Sangiovanni-Vincentelli, IEEE International Conference on Computer-Aided Design, Digest of Technical Papers, 1988, pp. 290-293.

[Hon74] S.J. Hong, R.G. Cain and D.L. Ostapko, "MINI: A Heuristic Approach for Logic Minimization", IBM Journal of Research and Development, Sept. 1974, pp. 443–458.

[Ngu87] L. Nguyen, M.A. Perkowski and N.B. Goldstein, "PALMINI - Fast Boolean Minimizer for Personal Computers", Proc. 24th. ACM/IEEE Design Automation Conference, 1987, pp. 615–621.

[Rud87] R.L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-Valued Minimization for PLA Optimization", IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 5, Sept. 1987, pp. 727–750.

[Sas81] T. Sasao, "Multiple-Valued Decomposition of Generalized Boolean Functions and the Complexity of Programmable Logic Arrays", IEEE Transactions on Computers, Vol. C-30, No. 9, Sept. 1981, pp. 635–643.

[Sas84] T. Sasao, "Input Variable Assignment and Output Phase Optimization of PLA's", IEEE Transactions on Computers, Vol. C-33, No. 10, Oct. 1984, pp. 879–894.

[Yan89a] S. Yang and M.J. Ciesielski, "A Generalized PLA Decomposition with Programmable Encoders", Proc. International Workshop on Logic Synthesis, MCNC, May 1989.

[Yan89b] S. Yang and M.J. Ciesielski, "On the Relationship between Input Encoding and Logic Minimization", Technical Report, TR-89-CSE-14, Dept. of Electrical and Computer Engineering, University of Massachusetts, Amherst, 1989.