THE ITERATIVE BOOLEAN MINIMIZER FC-MIN

Petr Fišer
Supervisor: Hana Kubátová
Czech Technical University
Dept. of Computer Science & Engineering
Karlovo nám. 13, Prague 2, Czech Rep.
fiserp@fel.cvut.cz

Abstract. *A novel two-level Boolean minimization method is presented here. In contrast to classical methods the cover of the on-set is computed first, whilst no implicants are known to this phase. The implicants are being derived from the source terms by their expansion directed by the cover. This allows us to generate group implicants directly, avoiding the time-consuming implicant expansions and reductions. The method is especially efficient for problems with many input and output variables, where other minimization tools (ESPRESSO) are extremely slow.*

Keywords. *Boolean minimization, FC-Min, Cover, Implicant generation*

1 Introduction

The problem of two-level Boolean minimization occurs in almost every area of the logic design. It has been studied for many decades and plenty of minimization methods and algorithmic minimizers were developed. In 50's the classical Quine-McMluskey's method [1] was proposed and determined the two major Boolean minimization phases: *the generation of implicants* and the subsequent *solution of a covering problem (CP)*. Many following algorithms were based on this paradigm, or the two phases were combined together somehow. MINI [2], ESPRESSO [3] and its modifications [4] were proposed, later Scherzo [5] with its improved CP solution algorithm was introduced. Lately we have developed a Boolean minimizer BOOM [6], which is able to handle functions with an extremely large number of input variables.

The general disadvantage of all these above-mentioned methods is their limited applicability to functions with a large number of *output variables*. In this case their results are either far from optimum, or the runtimes are extremely long. In the classical methods some set of prime implicants (PIs) of all the output functions is found at the beginning, then they are being reduced into group implicants (and possibly again expanded), and at the end of the process the covering problem (CP) is solved. In this phase only the necessary set of implicants is selected. All these phases are rather time and memory consuming.

In [7] we have proposed a method solving this problem efficiently. Instead of producing a vast number of implicants (both the prime and group ones) at the beginning of the minimization process and then selecting only few of them to cover the on-set in the CP solution phase, we proceed backwards. We compute an irredundant cover of the on-set first. This cover determines the number of implicants in the final solution and the properties of the implicants. For each element of the cover its implicant is generated by expansion of the appropriate on-set terms. Thus, no redundant implicants, i.e., implicants that are not a part of the solution are generated. This property makes the algorithm extremely fast even for very large problems. Since the main part of this algorithm is the Find Coverage phase, it was named *FC-Min*.

In this paper we propose an enhancement to this method which enables us to reach better results in terms of the quality of the solution. An *Iterative FC-Min* repeats the implicant generation several times, while all the implicants found are being collected. After that the CP is solved. The principles and properties of this method are discussed and experimentally evaluated here.

Let us note that the principles of FC-Min are exploited in the Coverage-Directed Assignment (CD-A) method used in BIST design [13, 14]. Particularly, the Find Coverage phase remains unchanged here, while the implicant generation phase is a generalization of the implicant generation phase used in FC-Min.

The paper has the following structure: after the Introduction and the Problem Statement in Section 2 the principles of FC-Min are described in Section 3, Section 4 introduces the Iterative FC-Min. Experimental results are shown in Section 5, Section 6 concludes the paper.

2 Problem Statement

Let us have a set of *m* Boolean functions of *n* input variables $\boldsymbol{F}_1(x_1, x_2, \ldots x_n)$, $\boldsymbol{F}_2(x_1, x_2, \ldots x_n)$, … $\boldsymbol{F}_m(x_1, x_2, \ldots x_n)$; the output values of the care terms are defined by a truth table. Thus, each function is specified by its on-set $F_i(x_1, x_2, \ldots x_n)$ and off-set $R_i(x_1, x_2, \ldots x_n)$. To the minterms that are not present in the truth table are implicitly assigned don't care values. The part of a truth table representing the terms will be denoted as an *input matrix* ***I***, the rows of the input matrix will be denoted as *input vectors*. The part defining the output values of the functions will be called an *output matrix* ***O;*** similarly, the rows of this matrix *output vectors*. The number of **I** matrix columns correspond to the number of input variables *n*, the number of **O** matrix columns is equal to the number of output variables *m*, the number of **I** and **O** matrix rows will be denoted as *p*.

Specifying a Boolean function by its on-set and off-set, rather by its on-set and don't care set, which is commonly preferred, is advantageous especially for functions that have defined values of only few terms. The typical example of the use of such a function can be found, e.g., in the build-in self-test (BIST) design [8, 9], for which the method was originally designed.

Our task is to synthesize a two-level circuit implementing the multi-output Boolean function described by the truth table, whereas the implementation of the circuit should be as small as possible. The result will be in a form of a set of *m* SOP (sum-of-the-product) forms implementing the *m* output functions, while some terms might be reused for more than one output function.

3 Principles of FC-Min

As it was stated in the introduction, the method consists of two major phases: the Find Coverage phase, in which the rectangle cover [10] of the on-set is found, and the Implicant Generation phase producing the very implicants from this cover. These phases may be conducted separately and independently, one after another. However, there might occur one serious problem – when the whole cover is generated not taking into account the possible implicants, it may happen that it wouldn't be possible to produce all the implicants. In other words, there may not be a solution for a particular cover. This problem is discussed more thoroughly in [7].

We have found that the best way to solve this problem is to generate the implicants together with the cover. Thus, immediately after finding one element of the cover the implicant corresponding to it is generated with respect to the previously found implicants. This is being repeated until the whole on-set is covered.

Let us state the problem formally:

Definition 1

Let t_i be an implicant. The *coverage set* $C(t_i)$ of the implicant t_i is a set of vectors (rows) of the **O** matrix, in which at least one "1" value is covered by this implicant. In other words, the coverage set is a set of vectors of the output matrix for which t_i is an implicant for at least one output variable. ■

Definition 2

The *coverage mask* $M(t_i)$ of the implicant t_i is the set of columns of the **O** matrix, in which all vectors included in $C(t_i)$ have one or more "1" value. The coverage mask $M(t_i)$ can also be expressed as a vector in the output matrix corresponding to the term t_i. In the following text we will use both representations of the coverage mask.

■

Definition 3

The *coverage of an implicant* t_i is a pair of the sets $C(t_i)$ and $M(t_i)$ for which the following equation holds:

$$\forall a \in C(t_i), \forall b \in M(t_i): \mathbf{O}[a,b] \neq \text{"0"}$$

■

Definition 4

The *coverage of the matrix* $\boldsymbol{O}$ is a set of implicant coverages $\{C(t_i), M(t_i)\}$ such that

$$\forall a < p, \forall b < p, \mathbf{O}[a,b] = \text{"1"}: \{a,b\} \in \bigcup_{\forall i} C(t_i) \times M(t_i)$$

■

The implicant coverages will be also denoted as *coverage elements* with respect to the coverage of the matrix.

An example of such a coverage of **O** matrix is shown in Figure 1. Here all the "1"s are covered by six implicants t_1-t_6. Their respective coverage sets and masks are shown in Table 1.

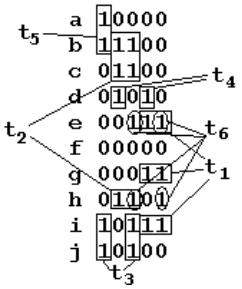

Figure 1: Coverage of the output matrix

Table 1: Coverage sets and masks for Fig. 1

Implicant	$C(t_i)$	$M(t_i)$
t_1	{*e*, *g*, *i*}	{0, 0, 0, 1, 1}
t_2	{*b*, *c*, *h*}	{0, 1, 1, 0, 0}
t_3	{*i*, *j*}	{1, 0, 1, 0, 0}
t_4	{*d*}	{0, 1, 0, 1, 0}
t_5	{*a*, *b*}	{1, 1, 0, 0, 0}
t_6	{*e*, *h*}	{0, 0, 1, 0, 1}

Finding an optimum rectangle cover is a NP-hard problem, thus some heuristic must be used. There exist many efficient algorithms finding a cover consisting of the minimum of elements [10]. However, such a cover need not be always optimal for solving the minimization problem. The solution will be then consisted of the minimum of product terms, however, it needs not be optimal with respect to the number of literals.

Our heuristic is based on a gradual search for a coverage consisting of the maximum number of "1"s. Firstly, the output vector containing the most not yet covered ones is selected as a basis for a new cover - in our example (Fig. 1) it is the *i* row with four ones. Now we continue the search for the next row to add in order to increase the number of the covered ones. When the row *g* is added to *i*, the number of covered ones will remain the same (because the first and the third variable cannot be covered after that). After adding the row *e*, the number of covered ones increases to six.

Finding the coverage consisting of many "1"s in the output matrix is advantageous indeed, however it often means that it contains many vectors. This fact complicates the subsequent phase – finding the structure of a term. A term whose cover consists of fewer vectors is easier to find. Thus, the heuristic algorithm is driven by a *depth factor* DF. Since each of the rectangle covers is being produced by a successive addition of vectors into it, we can decide after every addition whether to extend the cover to more vectors, or to terminate its generation, even if it could grow bigger. The decision is made at random with a probability given by DF. For instance, when DF = 1, there is an equal probability that the search will continue; when DF = 1/5, there is a probability 1:5 for a continual, and thus terms that cover less vectors and more outputs are more likely generated. When the depth factor is low, the runtimes are shorter, while the complexity of the result is slightly higher.

3.1 Generating the Implicants

For each coverage element an implicant has to be produced. To do so, only the knowledge of the particular coverage set together with the **I** matrix is needed. The structure of the **O** matrix, as well as the coverage mask, is insignificant here.

Obviously, when a term (cube) should cover a particular output vector, the corresponding input vector must be contained in this cube, since the input vector implies the output. From this results

that the *minimum term* satisfying the particular cover can be constructed as a *minimum supercube* of all the input vectors corresponding to $C(t_i)$. Moreover, this supercube must not intersect any **I** matrix term that is not included in its coverage set, since it would cover some zeros then.

Let us assume our leading example. Fig. 2 shows both the input and output matrices.

```
a 11010 10000
b 10000 11100
c 01001 01100
d 01111 01010
e 00110 00111
f 01110 00000
g 10110 00011
h 00001 01101
i 10101 10111
j 11100 10100
```

Figure 2. The input and output matrices

```
00110
10110
10101
-01--
```

Figure 3. The implicant t_1

The term t_1 covers vectors *e, g* and *i* – see Fig. 1. Thus, the minimum term that can be a candidate for t_1 must be constructed as a minimum supercube of the terms *e, g* and *i* in **I**, see Fig. 3.

The term `(-01--)` was found as a candidate for an implicant t_1. However, since it has to cover *only* the vectors listed above, the term must not intersect with any of the other terms. We can find that it is a valid implicant by comparing the term with the other terms.

Similarly, we can obtain the minimum implicants t_1-t_6. Figure 4 shows all the minimum implicants obtained by finding the corresponding supercubes of the source terms, together with the output part of the resulting PLA matrix:

```
t1: -01-- 00011    t4: 01111 01010
t2: --00- 01100    t5: 1-0-0 10000
t3: 1-10- 10100    t6: 00--- 00101
```

Figure 4. The minimum implicants

```
t1: -01-- 00011    t4: ---11 01010
t2: --00- 01100    t5: 1-0-- 10000
t3: --10- 10100    t6: 00--- 00101
```

Figure 5. The expanded implicants

The implicants obtained could form the final solution. However, they can be further expanded to improve the quality of the result. The final result is shown in Fig. 5; the literals removed by the expansion are highlighted.

4 Iterative FC-Min

In most cases the FC-Min algorithm is not deterministic – the progress of the Find Coverage phase is controlled by a random number generator, see Section 3. Thus, repeated run of FC-Min could produce different results. The idea of the *Iterative FC-Min* consists in repeating the FC-Min several times, while all the different implicants are put together and stored. At the end the final solution is constructed by solving the covering problem using all the implicants. Even if each of the single FC-Min runs produces a valid solution, a properly selected *combination* of the implicants obtained from different iterations might produce a better solution. Of course, it is paid by a longer runtime.

An example of an iterative FC-Min run is shown in Fig. 6. The sample problem solved was a randomly generated function of 20 input and 20 output variables, with 200 terms defined. The depth factor was set to 2:1. The thin line indicates the growth of the number of implicants. We can see that in 10000 iterations the total number of different implicants increased from 200 to 4000. The thick line shows the quality of the result after each iteration. It is measured in gate equivalents (GEs), which is a good approximation of a complexity of the physical implementation of the circuit [15]. The number of GEs was reduced by 8% in 4000 iterations, then it remains unchanged, even if the number of implicants is still increasing.

Figure 7 illustrates the influence of the depth factor DF on the implicant growth rate. For an extremely low DF the number of implicants remains almost unchanged. On the other hand, many different implicants are being generated when increasing DF. This allows us to reach a better result in a shorter time.

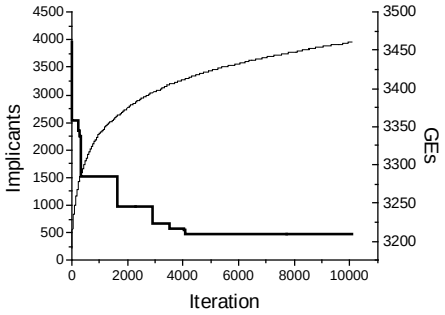

Figure 6: Iterative minimization example

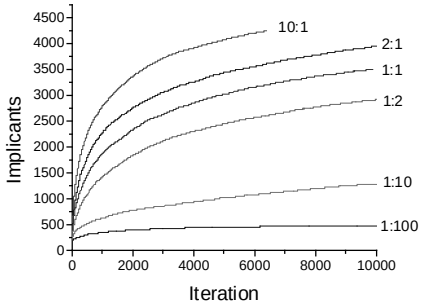

Figure 7: Influence of DF

5 Experimental Results

A lot of extensive testing was done to evaluate the performance of the method. We have compared the FC-Min results with ESPRESSO [3, 11]. The algorithm was programmed in C++ and the experiments were run on a computer with Athlon 900 MHz processor.

The first set of experiments were the MCNC benchmarks [11]. Only one iteration of FC-Min was necessary to obtain satisfactory results here. For 72% of the benchmarks we obtained the solution in a shorter time than ESPRESSO did and for 86% of the benchmarks FC-Min gave the same or even better result than ESPRESSO. For more detailed comments see [7].

5.1 Randomly Generated Problems

The second set of problems on which we have tested FC-Min were randomly generated functions, functions with no special properties (no aggregated ones in the output matrix, etc.). With such problems we can easily observe the properties and scalability of the algorithm. One of the reasons why FC-Min was developed was a need to synthesize the combinational logic for BIST, namely the output decoder transforming the LFSR patterns into test patterns pre-generated by an ATPG tool [9]. Both the LFSR and ATPG patterns mostly have a random nature, and thus the randomly generated benchmarks simulate these practical problems very well.

Problems with a varying number of input variables and terms were generated, the number of outputs was fixed to 15. These artificial benchmarks were solved both by FC-Min (in one iteration) and ESPRESSO to compare the performance. Table 2 shows the results of the minimization. The number of inputs increases in the horizontal direction (n), the number of care terms in the vertical direction (p). Each of the cells contains average values of ten problems of the same size that were solved, to ensure steady statistical values. The first row of each cell in the table contains results obtained by FC-Min, the second row shows ESPRESSO results. Each row indicates the runtime in seconds, the number of literals in the resulting SOP form, the output cost (the number of inputs into all output OR gates) and the number of group terms. The depth factor was set to 10:1.

We can see that in all the cases the FC-Min completed the minimization in a significantly shorter time than ESPRESSO, which was paid by a slightly worse quality of the results.

Table 2: Randomly generated problems

p/n	25	50	75	100
50	0.03/463/267/61 2.28/199/330/47	0.04/718/210/59 9.40/200/308/48	0.04/995/182/57 18.05/190/300/47	0.05/1185/175/54 25.18/179/293/45
100	0.15/945/539/113 8.67/508/603/94	0.13/1454/431/109 50.66/468/562/89	0.15/1832/389/104 106.10/453/550/88	0.17/2104/363/98 209.03/431/525/86
150	0.37/1442/850/171 23.62/830/896/139	0.30/2178/657/159 157.34/765/801/130	0.32/2588/586/148 358.15/733/774/126	0.35/3234/536/145 700.35/723/766/126
200	0.70/1889/1176/222 38.27/1179/1150/183	0.57/2902/922/209 312.75/1078/1061/170	0.57/3599/775/196 735.90/1035/1002/165	0.60/4250/713/188 1642.57/1009/958/161

Entry format: time [s] / #of literals / output cost / #of implicants

6 Conclusions

We propose a new two-level Boolean minimization algorithm called “FC-Min”. The implicants are being constructed using a pre-computed cover of the on-set, thus only implicants that will be a part of the final solution are produced. With respect to the other minimization tools the algorithm is extremely fast, while the quality of the result is comparable. FC-Min is extremely efficient for functions with a large number of input and output variables, where the other algorithms are too slow. The algorithm was tested on a set of MCNC benchmarks, where it has beaten ESPRESSO in most cases. For randomly generated problems FC-Min it is significantly faster than other algorithms.

Acknowledgement

This research was in part supported by grant #102/01/0566 “Built-in Self-Test Equipment Optimization Methods in Integrated Circuits” of the Czech Grant Agency (GACR) and MSM 212300014 , 1999 – 2003.

References

[1] E.J. McCluskey, “Minimization of Boolean functions”, The Bell System Technical Journal, 35, No. 5, Nov. 1956, pp. 1417-1444

[2] S.J. Hong, R.G. Cain and D.L. Ostapko, “MINI: A heuristic approach for logic minimization”, IBM Journal of Res. & Dev., Sept. 1974, pp.443-458

[3] R.K. Brayton et al., “Logic minimization algorithms for VLSI synthesis”, Boston, MA, Kluwer Academic Publishers, 1984, 192 pp.

[4] P. McGeer et al., “ESPRESSO-SIGNATURE: A new exact minimizer for logic functions”, Proc. DAC’93

[5] O. Coudert, “Doing two-level logic minimization 100 times faster”, Proc. of the sixth annual ACM-SIAM symposium on Discrete algorithms, 1995, pp.112-121

[6] J. Hlavička and P. Fišer, “BOOM - a Heuristic Boolean Minimizer”, Proc. ICCAD-2001, San Jose, Cal. (USA), 4.-8.11.2001, 439-442

[7] P. Fišer, J. Hlavička and H. Kubátová, “FC-Min: A Fast Multi-Output Boolean Minimizer”, Proc. Euromicro Symposium on Digital Systems Design (DSD'03), Antalya (TR), 3.-5.9.2003

[8] M. Chatterjee and D.J. Pradhan, “A novel pattern generator for near-perfect fault coverage”, Proc. of VLSI Test Symposium 1995, pp. 417-425

[9] P. Fišer and J. Hlavička, „Column-Matching Based BIST Design Method“, Proc. 7th IEEE European Test Workshop (ETW'02), Corfu (Greece), 26.-29.5.2002, pp. 15-16

[10] S. Hassoun and T. Sasao, „Logic Synthesis and Verification", Boston, MA, Kluwer Academic Publishers, 2002, 454 pp.

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

[12] S. Yang, „Logic Synthesis and Optimization Benchmarks User Guide“, Technical Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, NC, January 1991

[13] P. Fišer, J. Hlavička and H. Kubátová, "Coverage-Directed Assignment Approach to BIST", Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS'03), Poznan (Poland), 14.-16.4.2003, pp. 87-92

[14] P. Fišer, J. Hlavička and H. Kubátová, "CD-A Based BIST Method", Proc. 6th International Workshop on Electronics, Control, Measurement and Signals (ECMS'03), Liberec (CR), 2.-4.6.2003, pp. 279-283

[15] J. Hartmann and G. Kemnitz, "How to Do Weighted Random Testing for BIST", Proc. of International Conference on Computer-Aided Design (ICCAD), pp. 568-571, 1993