A Flexible Minimization and Partitioning Method

Petr Fiser, Jan Hlavicka
Czech Technical University, Karlovo nám. 13, 121 35 Prague 2
e-mail: fiserp@fel.cvut.cz, hlavicka@fel.cvut.cz

Abstract

The article describes a new Boolean minimization and single-level partitioning method based on the BOOM minimizer. The minimization is performed with respect to various restrictions stated for the use of input variables. This enables us to effectively decompose the circuit into several components for which the numbers of inputs and outputs are explicitly specified. The method can thus be used to produce circuits with minimized load of the external inputs, circuits with balanced input load, or easily testable circuits. It can handle extremely large functions (up to thousands of input variables) in a very short time and its use is advantageous above all for highly unspecified functions, where the number of don't cares is large.

1 Introduction

Logic design is always connected with many constraints, mostly following from the properties of the available physical elements. Whether we compose a circuit of simple logic gates, PGAs or LUTs in FPGAs, the number of inputs and outputs of one element is limited. Further limitations to be met are, e.g., the maximum load of the circuit inputs or maximum number of logic levels. All these requirements can be met by proper circuit decomposition. The common two-level minimization algorithms based on the Quine-McCluskey approach, like e.g. ESPRESSO [6], do not support any decomposition features, thus we cannot state any requirements and constraints for the properties of the resulting circuit. In this case the decomposition is done independently on the minimization, as a sort of post-processing [8, 9]. The decomposition is then more difficult to do and the results are usually less satisfactory. Our minimization algorithm takes into account pre-defined requirements for the resulting circuit, because it combines minimization and decomposition phases to produce better results.

In this article first the problem statement of the single-level decomposition, load balancing and DFT design is stated in Section 2. In Section 3 we describe the necessary modifications of the BOOM minimization algorithm and in Section 4 the modified covering problem (CP) solution algorithm is proposed. Section 5 contains the experimental results and Section 6 concludes the paper.

2 Problem Statement

2.1 Single-Level Decomposition

The problem to be solved here is a *single-level decomposition* of a combinational Boolean function. The resulting circuit has thus as many logic levels, as there are in each block. The idea is illustrated by Figure 1, where the logic function of 7 inputs x_1-x_7 and 6 outputs y_1-y_6 is decomposed into two 5-input and 3-output blocks while each block is a two-level (AND-OR) circuit.

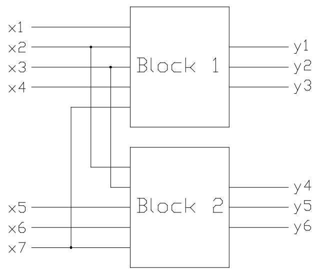


Figure 1. *A single-level decomposition*

In order to generate the required values of the output functions, mostly only a subset of all input variables has to be used for each of them, especially when the number of input variables is large. Moreover, even when maintaining the number of input variables used, there can exist several solutions with equal complexity. These observations give us some freedom when accomplishing the logic design, so that we can partially select which variables will be used in the final solution.

For example, in Figure 1 only five of seven input variables are needed to produce each bundle of three outputs. In this case the circuit can be decomposed into two stand-alone blocks. This type of partitioning is based on finding the minimal set of input variables needed to produce a group of output functions.

Let us have b logic blocks, e.g., PLA circuits, PALs, GALs or other logic components, implementing a set of two-level multiple-output Boolean functions. The maximum number of inputs into the i-th block will be denoted as I_i, the maximum number of outputs of the i-th block as O_i. Further, let us have a Boolean function F with n inputs and m outputs. The minimization of the function F combined with the partitioning into b blocks means finding b groups of sum-of-product (SOP) forms G_i, where the number of input variables used in each G_i is lower or equal to I_i and the number of SOP forms in each G_i is lower or equal to O_i for all $0 \le i < b$. At the same time, the complexities of all G_i have to be kept minimal to “fit” into the blocks.

The problem solution consists of two steps: first, all the m outputs of the function F have to be assigned to the individual functional blocks, then the minimization is performed, while the number of inputs entering the blocks is kept minimal. Thus, the inputs are assigned to the blocks during the minimization process. If all the resulting functions G_i meet the given constrains, the process is completed. If not, the minimization must be repeated with other parameters set and, if necessary, the outputs have to be reassigned.

2.2 Load Balancing

One of the next most frequent problems we encounter in logic design is the limited maximum fan-out of the logical elements. Especially nowadays, when the low-power design is often desired, the fan-outs of the gates are low, and thus the load minimization is strongly required. We have combined the logic minimization and partitioning process with the load minimization to meet the requirements for the maximum loads of the inputs.

We will consider the synthesis of a function F from the previous subsection, while the load minimization is now required. Let us assume the multilevel design where the inputs of the circuit, which is being synthesized, are fed from another logic circuit whose maximum fan-outs are limited. Next we assume that the used blocks contain input buffers, thus the inner complexity of the blocks does not affect the load of the inputs (let the load of all the inputs of the blocks be equal to one). The problem of the load minimization consists in the partitioning the function F into b blocks, while the number of branchings of each input is to be kept minimal. For example, if an input enters two blocks (i.e., one branching), the load of this input is equal to two. The load balancing means finding such a partitioning where the loads of all the inputs are balanced and kept minimal.

2.3 Design for Testability

It is well known that the testability of a circuit, measured by the complexity of a test pattern generator and/or by the length of the test set, depends to a great extent on the internal structure of the circuit used for the implementation of the given function. This means that if one and the same function is implemented by several different circuits, the testability of each of them may be different. Several approaches to the design for testability (DFT) have been developed so far [10]. One of them is the design of pseudo-exhaustively testable circuits [11]. These circuits allow the use of an exhaustive test set for every output cone, which is a set of all components feeding one external output (see Fig. 2). However, for practical reasons, the size of an exhaustively testable circuit is limited by the number of its inputs (the usually accepted maximum value is 30). Hence it is desirable to have a design method that guarantees, that the number of external inputs feeding any single external output will be lower than a given limit value. This can be achieved by limiting the number of input variables needed to produce an output during the minimization. Intuitively, reducing the number of inputs into the blocks by the partitioning also reduces the number of inputs feeding one single output, thus the size of the corresponding cone is equal or slightly lower than the number of inputs entering the block.

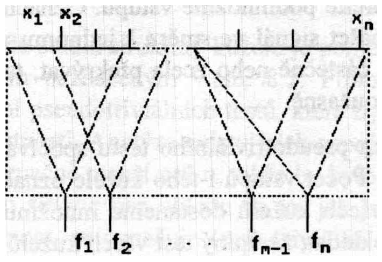


Fig. 2. *The circuit cones*

3 Principles of BOOM Minimizer

The suggested partitioning algorithm is based on the BOOM minimization tool presented earlier [1-5]. This tool proved to be very efficient for the minimization of large functions (functions with a large number of input variables) with many don't care states, thus where only few care terms are specified. For example, the minimization of a function of 2000 input variables with 1000 care terms took less than 5 minutes on an ordinary PC.

The minimization algorithm consists of two major phases: *generation of implicants* (prime implicants for single-output functions, group implicants for multi-output functions) and the subsequent solution of the *covering problem* (CP). The generation of implicants for single-output functions is performed in two steps: first the Coverage-Directed Search (CD-Search) generates a sufficient set of implicants needed for covering the source function and then the implicants are expanded into prime implicants in the Implicant Expansion (IE) phase. In addition to it, for multi-output functions the primes are reduced into group implicants and then the group CP is solved. The general structure of the minimization process can be illustrated by Fig. 3. The process is run repeatedly in order to reach better results. More details concerning these procedures can be found in [3, 4].

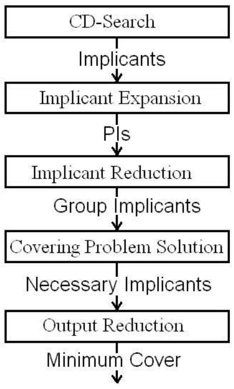


Fig. 3. *BOOM structure*

In order to perform the minimization together with the decomposition, the two major phases – CD-Search and CP solution - have to be slightly modified.

3.1 The CD-Search

The CD-Search consists in a search for the most suitable literals that should be added to some previously constructed term. This search is directed towards finding an implicant that covers as many 1-terms as possible. To do this, we start implicant generation by selecting the most frequent input literal from the given on-set. The selected literal describes an *n-1* dimensional hypercube, which may be an implicant, if it does not intersect with any 0-term. If there are some 0-minterms covered, we add another literal and verify whether the new term already corresponds to an implicant by comparing it with 0-terms. After every literal removal we temporarily remove the on-terms that cannot be covered by any term containing the selected literal. These are the terms containing that literal with the opposite polarity. We continue adding literals until an implicant is generated, then we record it, remove the covered 1-terms, and start searching for other implicants. In this way we generate new implicants, until the whole on-set is covered. The output of this algorithm is a set of product terms covering all 1-terms and not intersecting any 0-term. This procedure is done independently for all output variables.

The basic CD-Search algorithm for a single-output function can be described by the following function in pseudo-code. The inputs are the on-set (F) and the off-set (R); the output is the sum of products (H) that covers the given on-set.

Algorithm 1. *CD-Search*

```
CD_Search(F, R) {
     H = ∅            // H is being created
     do
        F′ = F        // F′ is the reduced on-set
        t = true      // t is the term in progress
        do
           v = find_literal(F′)
           t = t AND v
           F′ = F′ – cubes_not_including(t)
        while (t ∩ R ≠ ∅)
        H = H ∪ t
        F = F - F′
     until (F == ∅)
     return H
}
```

3.2 CD-Search Modified for Partitioning

Originally, the literal selection function `find_literal` returns the literal with the maximum frequency of occurrence in the reduced on-set. Then the algorithm produces the best results in terms of minimality of the solution. However, this rule can be modified in order to produce a set of terms containing the minimum of input variables.

As the assignment of the output variables to the blocks was made before the CD-Search, we know in every CD-Search pass which block we are currently processing. When some input variable is selected during the CD-Search, it must be necessarily included in the processed block (if the term including this input variable is not removed during the CP solution phase). Thus we can record all input variables entering the blocks. The idea of adapting the CD-Search for partitioning consists in penalizing the selection of a variable that has not yet been selected into the currently processed block by some factor. Particularly, in the `find_literal` function the frequency of the literal that is already included in the processed block is multiplied by the CD-Search partitioning force PF_{CD}, and is thus preferred to other literals. The higher this force is, the less input variables will be entering the blocks, however, on the other hand, the complexity of the resulting SOP form will be higher.

3.3 CD-Search Modified for Load Balancing

Similarly, the `find_literal` function can be modified in order to minimize the load of the inputs. If input variable entering other blocks than the processed one was selected, the load of this input would be increased by one. To minimize the load, the selection of such a variable must be penalized. In praxis, the frequency of literal that is already included in *a* blocks and is not included in the currently processed block is divided by a CD-Search load balancing factor LBF_{CD} multiplied by *a*. Again, the higher this factor, the lower the load of the inputs, however it increases the complexity of the blocks.

The following pseudo-code describes the computation of the scoring function $S(v)$ of the literal v. The `find_literal` function then selects the literal with the maximum value of this function.

Algorithm 2. *Scoring function*

```
S(v) = count_frequency(F')          // F' is the reduced on-set
if (enters(v, current_block)) S(v) = S(v)*($PF_{CD}$ + 1);
else {
        a = 1;
        for (i = 0; i < blocks; i++)
        if (i != current_block && enters(v, i)) a++;
   S(v) = S(v) / (a * $LBF_{CD}$ + 1);
}
```

4 Covering Problem Solution

The second most important phase of any minimization method is the CP solution, described, e.g., in [7]. The CP algorithm used in BOOM must be modified too in order to support the partitioning. Because the CP solution is the final phase of the minimization process – and thus it produces the very terms included in the final solution, it could waste all the effort made in the CD-Search if it was not solved with respect to the partitioning. This fact is illustrated in Section 5.

Firstly, the CP is solved for each block individually. If the partitioning is done into *b* blocks, then *b* group covering problems are solved independently. Intuitively, at the end of this phase *b* groups of SOP forms are produced and these forms are to be implemented by hardware blocks. To this point, any CP solution algorithm can be used to accomplish this requirement.

The next – and most important – modification requires the covering problem to be solved by a successive addition of terms into the solution. Thus some heuristic must be used, as the exact CP solution algorithms would be too difficult to modify. We use a CP solution algorithm based on computing the contributions (scoring functions) of terms as a criterion for their inclusion into the solution.

This means that a score is assigned to every implicant, which expresses its potential contribution to the minimal solution. In practice, for every on-set term to be covered its weight is computed as a reciprocal value of the number of implicants that cover this term. The weights of all on-set terms covered by one implicant are then summed up and assigned to this implicant as its contribution. The implicant with the highest contribution is then selected for the solution. For more detail description of this process see [4, 7].

To apply the partitioning, the contributions of the implicants have to be modified in a similar way as the CD-Search scoring function was. Input variables used in particular blocks are again recorded during the CP solution process. The additional weights of the implicants are computed to be proportional to the number of new input variables they would add to the currently processed block if they were selected into the solution. These weights represent the penalization factor PF_{COV}. Again, the more input variables are newly added into the given block by a term, the less likely this term will be selected.

Like in the CD-Search, the load minimization is realized by introducing a CP solution load balancing factor LBF_{COV} penalizing the selection of a term that would increase the load of the inputs.

The contribution modification method is the same as Algorithm 2, except of all the literals in the particular term are studied and their penalization factors are summed.

5 Experimental Results

5.1 Influence of the Partitioning Forces

Applying the partitioning in the CD-Search and CP solution algorithms allows us to generate a good solution in the terms of complexity of the blocks (the number of terms in blocks) and simultaneously to keep the number of input variables entering the blocks minimal when respecting the defined physical constraints. However, as was said before, these two criteria of minimality are controversial - the more we try to reduce the number of inputs into blocks, the more complex are the resulting SOP forms and vice versa. The type of the solution can be selected by adjusting the partitioning forces. We have found that both the CD-Search and the CP solution algorithm must be necessarily modified in this way to produce good results of the partitioning.

The influence of the algorithm modifications is shown in Table 1. We solved a problem of partitioning a Boolean function of 50 input variables with 20 outputs and 200 terms defined. This function was to be decomposed into four 5-output blocks. The first line in each entry of the table shows the time in seconds needed to complete one iteration of the modified BOOM algorithm, the second line indicates the complexity of the result, which is measured in terms of the number of literals in the SOP form, the output cost (the number of wires entering the OR-gates) and the total number of SOP terms. The third line shows the quality of the partitioning, which is the sum of the number of inputs entering the four blocks and the number of input variables that were needed to implement the function. This measurement was done for various CD-search and CP solution partitioning forces, while the load balancing forces were set to zero (no load balancing). The upper left corner represents the standard minimization procedure. Here no partitioning forces were applied and thus the partitioning was done just by solving four independent covering problems at the end of the minimization.

The CD-search partitioning forces are increasing in the horizontal direction, and the partitioning forces influencing the CP solution are changing in the vertical direction.

The program was run on a PC with Athlon 900 MHz processor and 256 MB RAM.

Table 1. *Results of the partitioning test run*

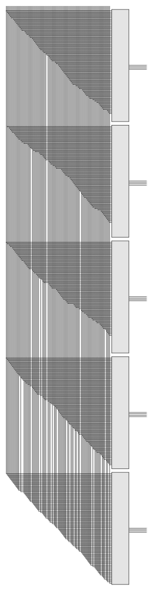
PF_{COV} / PF_{CD}	0.0	0.5	1.0	1.5
0.0	6.04 2714/506/455 198/50	6.81 3016/592/501 174/50	10.18 3405/804/560 176/50	10.14 3444/826/563 170/48
0.5	6.27 2731/515/456 197/50	6.95 2989/575/501 157/50	10.26 3546/746/590 112/47	9.95 3539/740/587 113/46
1.0	6.30 2730/514/457 197/50	6.95 3016/566/505 154/50	10.25 3630/750/604 103/47	10.09 3638/736/606 98/44
1.5	6.36 2820/522/472 196/50	7.00 3051/573/511 154/50	10.31 3616/747/603 103/47	10.09 3654/735/609 98/44
2.0	6.36 2876/548/478 196/50	6.98 3061/572/513 154/50	10.26 3669/740/612 103/47	10.26 3677/740/611 98/44
2.5	6.43 2993/547/498 196/50	7.04 3114/573/520 154/50	10.59 3686/739/615 103/47	10.20 3715/735/618 98/44
3.0	6.42 3128/573/518 196/50	6.99 3118/572/521 154/50	10.30 3747/743/626 103/47	10.11 3761/739/626 98/44
3.5	6.43 3139/571/520 197/50	6.99 3134/578/523 154/50	10.35 3777/741/631 103/47	10.16 3784/736/630 98/44

Entry format: solution time in seconds
# of literals / output cost / # of terms
sum of inputs into blocks / inputs used

We can see that when using the simple CD-Search and simple CP solution (upper left corner), we obtain a solution in which almost every input wire enters all blocks. By increasing the partitioning forces we reduce the number of inputs into blocks down to 98 (lower right corner) where every input enters in the average two blocks. Thus we can conclude that both partitioning forces are important, whereas modifying only the CD-Search or only the CP solution algorithm does not produce satisfactory results. Higher partitioning forces than those shown in the table proved to be in this case inefficient. The solution time depends above all on the penalization of the CD-Search, whereas the penalization of the CP solution algorithm affects the runtime only slightly.

5.2 Illustrative Example

In this subsection we will illustrate the partitioning and load minimization on a particular circuit and show the results on schematics. We have a Boolean function with 100 input variables, 20 output variables and 100 care terms described by the truth table. The function is to be minimized and decomposed into five 4-output blocks. Figure 4 shows the resulting circuit when the minimization without any partitioning and load balancing is applied (all partitioning forces equal to zero). Although the complexity of the logic of the blocks is relatively low, the required number of inputs to the blocks is large (maximum is 80 inputs per block) and the maximum load is five (this means that at least one input enters all the blocks). When the partitioning is applied (Figure 5), the maximum necessary number of inputs to the blocks is reduced to 17, while also the maximum load is lower. In this case both the PF_{CD} and PF_{COV} were set to 2. Figure 6 shows the circuit when the load minimization is applied ($PF_{CD} = PF_{COV} = 1$, $LBF_{CD} = LBF_{COV} = 2$). Here the maximum load of the inputs is 2 (but only for 4 inputs). Table 2 summarizes all the obtained results in detail. Let us note, that the maximum size of a cone when no partitioning is applied is equal to 33, which makes the circuit pseudo-exhaustively nontestable. If the partitioning is applied, the maximum cone size is reduced to 15. In this case, the pseudo-exhaustive test can be applied to the circuit.



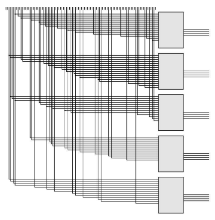


Figure 5. *Partitioning applied*

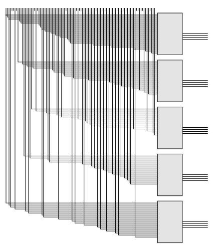


Figure 6. *Load balancing applied*

Figure 4. *No partitioning forces*

Table 2. *Summary of the decomposition results*

	no partitioning	partitioning only	load balancing
sum of inputs into blocks	371	69	84
used inputs	100	50	80
branching inputs	93	16	4
branchings	271	19	4
maximum load	5	3	2
average load	3.71	1.38	1.05
maximum cone size	33	15	18
average cone size	28.50	12.60	14.55
block 0	75x4, 128/28/28	14x4, 273/64/58	20x4, 247/56/53
block 1	70x4, 133/30/30	17x4, 297/72/64	17x4, 230/55/50
block 2	68x4, 123/28/28	13x4, 252/65/55	14x4, 233/56/51
block 3	78x4, 135/30/30	12x4, 257/67/55	15x4, 243/59/54
block 4	80x4, 139/30/30	13x4, 242/58/53	18x4, 252/54/53

Block info entry format: # of inputs of the block x # of outputs / # of literals / output cost / # of terms

6 Conclusions

We have shown that the top-down minimization method used in the BOOM minimizer suits very well for modifications like partitioning, load balancing and DFT. The minimization process can be guided by the predefined requirements, and thus influence the result of the minimization to suit our needs. The method is advantageous if we want to perform the minimization together with single-level decomposition. Further, the algorithm gives the possibility to minimize the load of the outputs of the preceding circuit by the load balancing. Another feature of the algorithm, namely the possibility of DFT, allows us to make the circuit pseudo-exhaustively testable.

The modification of the original BOOM algorithm consists in introducing partitioning and load balancing forces into two phases of the minimization algorithm denoted as CD-Search and CP solution. The effect of these forces on the quality of solution of a typical example was investigated and documented by a table of results. The partitioning and load balancing were applied to one sample function and the results are illustrated by figures of the resulting circuits.

Further research will be directed towards multi-level decomposition and the applications in the area of low-power devices.

Acknowledgment

This research was in part supported by grant 102/01/1531 of the Czech Grant Agency (GACR) "Formal Approaches in Digital Circuit Diagnostics - Testable Design Verification"

References

[1] Fiser, P. – Hlavicka, J.: Efficient Minimization Method for Incompletely Defined Boolean Functions, Proc. 4th Int. Workshop on Boolean Problems, Freiberg (Germany), Sept. 21-22, 2000, pp. 91-98
[2] Hlavička, J. - Fišer, P.: A Heuristic method of two-level logic synthesis, Proc. The 5th World Multiconference on Systemics, Cybernetics and Informatics SCI'2001, Orlando, Florida (USA) 22-25.7.2001, vol. XII, pp. 283-288
[3] Hlavička, J. - Fišer, P.: BOOM - a Heuristic Boolean Minimizer. Proc. International Conference on Computer-Aided Design ICCAD 2001, San Jose, California (USA), 4.-8.11.2001, pp. 439-442
[4] Fišer, P. - Hlavicka, J.: BOOM - a Boolean Minimizer. Research Report DC-2001-05, Prague, CTU Publishing House, June 2001, 37 pp.
[5] Hlavička, J. - Fišer, P.: Minimization and Partitioning Method Reducing Input Sets. Proc. 1st International Workshop on Electronic Design, Test & Applications (DELTA 2002), New Zealand, 29.-31.1., 2002 (poster)
[6] Brayton, R.K., et al.: Logic minimization algorithms for VLSI synthesis. Boston, MA, Kluwer Academic Publishers, 1984
[7] Coudert, O.: Two-level logic minimization: an overview. Integration, the VLSI journal, 17-2, pp. 97-140, Oct. 1994
[8] Jozwiak, L. - Chojnacki, A.: Effective and Efficient FPGA Synthesis through Functional Decomposition Based on Informational Relationship Measures, Proc. Euromicro Symposium on Digital Systems Design, Warsaw (Poland) 4-6.9.2001, pp. 30-37
[9] Scholl, C.: Multi-output functional decomposition with exploitation of don't cares. Proc. DATE 98, pp. 743-748
[10] Abramovici, M. - Breuer, M. - Friedman, A.: Digital systems testing and testable design. New York, Computer Science Press 1990
[11] McCluskey, E.J.: Verification testing - a pseudoexhaustive test technique. IEEE Trans. on Comp., Vol. C-33, No.6, June 1984, pp. 541-546
[12] http://service.felk.cvut.cz/vlsi/prj/BOOM/