Minimization and Partitioning Method Reducing Input Sets

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

Abstract

The article describes a new Boolean minimization and single-level partitioning method based on the BOOM minimization system [6]. The minimization is performed with respect to the restrictions stated for the number of inputs and outputs of individual components and/or with the goal to reach load balancing for the inputs. The method 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

When designing logic circuits we often encounter constraints 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. The number of logical levels may be limited too, above all due to the propagation delay constraints. These requirements can be met by circuit decomposition. The common two-level minimization algorithms based on the Quine-McCluskey approach, like e.g. ESPRESSO [7, 9], do not support any decomposition features and the decomposition is done independently, as a sort of post-processing [10, 11]. The decomposition is then more difficult to do and the results are usually less satisfactory. Our algorithm combines these two phases to produce better results.

The problem to be solved here is a single-level decomposition of a combinational Boolean function. The resulting circuit has thus the two-level structure. 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.

The input of our algorithm is a Boolean PLA matrix [7] defining the on-sets and off-sets (optionally don't care sets) of the output functions. The algorithm assigns these outputs to individual functional blocks (see Figure 1) and minimizes each set as a sum-of-product (SOP) expression according the decomposition.

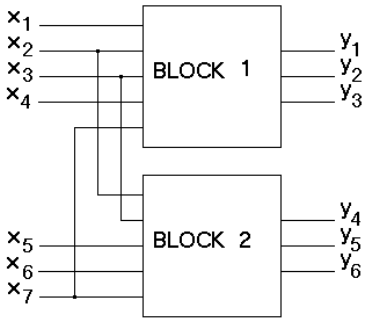

Figure 1. A single-level decomposition

In some cases not all input variables used in the input form are needed to produce the required output values. Similarly, some input variables can be substituted by others while preserving the required function. 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 stand-alone two-level blocks. The partitioning is based on finding the minimum of input variables needed to produce a group of output functions. First the outputs are assigned to the blocks and then the minimization is performed while trying to match the maximum allowed number of inputs into the blocks.

Principles of the BOOM Approach

The suggested partitioning algorithm is based on the BOOM minimization tool presented earlier [1-5]. It 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 addition to it, for multi-output functions the primes are reduced into group implicants and then the group CP is solved. More details concerning these procedures can be found in [2,4].

Combining Minimization with Decomposition

In order to perform the partitioning together with the minimization, the two major phases – CD-search and CP solution - have to be slightly modified. The CD-search consists in a search for the most suitable literals that should be added to some previously constructed term. For a two-level minimization of a Boolean function, the search for suitable literals that should be added to a term 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.

When literals with the maximum frequency of occurrence in the on-set are selected, 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 possible minimum of input variables by penalizing the selection of a variable that was not yet selected into the currently processed block. That implies that the input variables that were selected for a given block - i.e. wires entering the block - must always be recorded.

Covering Problem Solution

The second most important phase of BOOM - the CP solution, described e.g. in [8], must be modified in a similar way. Firstly, the CP is solved for each block individually. Further, when solving the CP, weights are assigned to implicants according to the number of input variables they would add to the given block if they were selected. The weights represent the penalization factor. The more input variables are newly added into the given block by a term, the less likely this term will be selected.

Experimental Results

Combining these two techniques 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, these two criteria of minimality are antipodal - 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" - i.e., coefficients penalizing the not-included variables. We have found that both the CD-search and the CP solution algorithm must be modified in this way to produce good results of the partitioning.

The influence of the algorithm modification is shown in Table 1. There we solved a problem of partitioning a function of 50 input variables with 20 outputs and 200 terms defined. This function was to be decomposed into four 5-output blocks. The table shows the minimization time, the complexity of the result (the number of literals in the SOP form, output cost and the number of terms) and the quality of the partitioning (sum of the number of inputs entering the four blocks) for various CD-search and CP solution partitioning forces. 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

COV/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
literals / output cost / terms
sum of inputs into blocks / inputs used

We can see that 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 CP solution algorithm does not produce sufficient results. Higher penalization values than those shown in the table proved to be inefficient. The solution time depends above all on the penalization of the CD-search, whereas the penalization of the CP solution algorithm does not affect the runtime.

A simple modification of this algorithm enables us to control the load distribution of the outputs of the preceding blocks, because inputs entering more than one block (branching inputs) are more loaded than those entering only one block. When input variables contained in other blocks are even more penalized, the occurrence of branching inputs is reduced and the output load of previous blocks is more balanced.

The decomposition algorithm was tested on several problems. Thus, e.g., a function with 100 inputs, 20 outputs and 100 minterms with defined output values was decomposed into 5 blocks, each having 4 outputs. The algorithm gave the following solutions. For a simple minimization all inputs were used and the average number of inputs into one block was 75. When the decomposition was performed, the number of inputs used was 58 and the average number of block inputs was 17. When the load balancing was used, the total number of inputs used was 80 and only 4 inputs were branching in contrast to 20 branching inputs in case of a decomposition without load balancing.

Conclusions

It was shown that combining the partitioning of a function with its minimization leads to better results than performing each of these design tasks independently. The modification consists in introducing partitioning 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.

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 the grant of the Czech Grant Agency GACR 102/99/1017.

References

[1] P. Fiser, and J. Hlavicka, 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] P. Fiser, and J. Hlavicka, Implicant Expansion Method used in the BOOM Minimizer. Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS'01), Gyor (Hungary), 18-20.4.2001, pp. 291-298

[3] J. Hlavicka, and P. Fiser, 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

[4] P. Fiser, and J. Hlavicka, On the Use of Mutations in Boolean Minimization. Proc. Euromicro Symposium on Digital Systems Design, Warsaw (Poland) 4-6.9.2001, pp. 300-307

[5] J. Hlavicka, and P. Fiser, BOOM - a Heuristic Boolean Minimizer. Proc. ICCAD-2001, San Jose, Cal. (USA), 4.-8.11.2001 (accepted for publication)

[6] http://cs.felk.cvut.cz/~fiserp/boom/

[7] R.K. Brayton et al., Logic minimization algorithms for VLSI synthesis. Boston, MA, Kluwer Academic Publishers, 1984

[8] O. Coudert, Two-level logic minimization: an overview. Integration, the VLSI journal, 17-2, pp. 97-140, Oct. 1994

[9] G.D.Hachtel and F. Somenzi, Logic synthesis and verification algorithms. Boston, MA, Kluwer Academic Publishers, 1996, 564 pp.

[10] L. Jozwiak and A. Chojnacki, 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

[11] C. Scholl: Multi-output functional decomposition with exploitation of don't cares. Proc. DATE 98, pp743-748