OUTPUT GROUPING-BASED DECOMPOSITION OF LOGIC FUNCTIONS

Petr Fišer, Hana Kubátová
Department of Computer Science and Engineering
Czech Technical University
Karlovo nám. 13, 121 35 Prague 2
e-mail: fiserp@fel.cvut.cz, kubatova@fel.cvut.cz

Abstract. *We present a method allowing us to determine the grouping of the outputs of the multi-output Boolean logic function for a single-level partitioning and minimization. Some kind of decomposition is often needed during the synthesis of logic circuits and the subsequent mapping onto technology. Sometimes a circuit has to be divided into several stand-alone parts, among its outputs, or possibly its inputs. It could be a case of a design targeted into PLAs, GALs, or any other monolithic components having a limited number of inputs and/or outputs. We propose a methodology to determine the way how the original circuit has to be partitioned into several parts of an arbitrary size, in order to reduce the complexity of the individual parts. The method is based on our FC-Min minimizer, even when no Boolean minimization has to be involved here. The efficiency of the method is demonstrated on the standard MCNC benchmarks.*

1 Introduction

Some kind of decomposition is always necessary to perform when designing complex VLSI circuits, with respect to the available components. Most of up to now proposed methods start with a two-level Boolean network (sum-of-products) and try to decompose it into a multi-level network. The Boolean function is being manipulated so as to extract subfunctions common to more of its parts. This is being done either algebraically, by finding the function's common divisors (kernels) [1], by using computationally demanding Boolean methods [2, 3], or by a functional decomposition [4, 5], lately based on BDDs [6, 7]. Nowadays, a functional bi-decomposition plays a big role, for it is generally usable for most of applications [8, 9, 10].

Most of the previously mentioned methods are primarily intended for single-output functions, even when they can be extended to multi-output functions. However, there is no method strictly determining the relations between the multi-output function's outputs. Our partitioning method is based on grouping *output* variables. There can be a relationship between several outputs of the function found. This relationship can be often derived from the function's group implicants. When, e.g., two outputs share most of the group implicants of the solution obtained after a two-level minimization, grouping these outputs together should be advantageous.

We present a method allowing us to find a proper grouping of output variables, based on our FC-Min Boolean minimizer [11]. The reason why this minimizer was selected is that it naturally produces group implicants of the minimized function and thus it is able to properly determine the outputs that share implicants. Moreover, it is extremely fast and thus its use as a pre-processor to the minimization almost does not increase the total design time. The method allows us to split the designed circuit into an arbitrary number of parts, each having an arbitrary number of outputs. Hence the method can be efficiently used for a decomposition of logic circuits to fit into any target device (PLA, GAL, FPGA). Moreover, the number of inputs of the individual parts of the decomposed circuit is reduced by the method too.

The paper is structured as follows: the principles of the single-level partitioning are given in Section 2, Section 3 briefly describes the Boolean minimizer FC-Min. The main ideas of our output-grouping method are stated in Section 4. The effectiveness of the method is documented by experimental results on the standard MCNC benchmarks in Section 5, Section 6 concludes the paper.

2 Single-Level Partitioning

There is a need to divide a circuit into several stand-alone blocks having a limited number of inputs and outputs (into e.g., PLAs, PALs, GALs). These blocks have to be synthesized separately then, since they cannot share internal signals. The blocks can share the input variables. Such a case of decomposition will be denoted as a *single-level partitioning,* since the number of levels of the circuit remains the same, see Fig. 1.

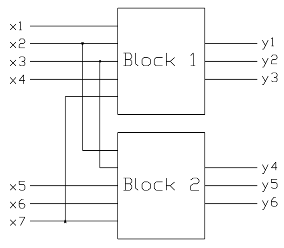

Figure 1: Single-level partitioning

Since the blocks cannot share the group terms (or subfunctions when they are designed as multi-level), the total complexity of the design is increased, in comparison to the all-in-one design. When our technique is used, this cost of the decomposition is reduced to minimum.

There are two decisions to be made: the choice of outputs to be grouped together and what a limited number of inputs can feed the blocks. An intensive work has been made to modify our Boolean minimizer BOOM [12, 13] to support the single-level partitioning, in order to reduce the numbers of input variables entering the blocks [14, 15]. The method how to determine the output grouping is presented in this paper. It is based on the FC-Min minimizer. For understanding the further text and the principles of the output-based partitioning, the main FC-Min principles will be described in the following Section.

3 FC-Min

The FC-Min minimizer has been developed to efficiently handle functions with a large number of output variables, [16]. The minimization is being conducted in a reverse way than the standard minimizers do. First, the cover of the on-set is found, independently on the source implicants. After that the minimized implicants are produced by joining the source implicants. This process is directed towards satisfying the cover. After that the implicants are expanded to reduce the number of literals. This approach makes FC-Min a very fast two-level group minimizer, since only implicants that will be a part of the final solution are produced.

The whole minimization process consists of three phases: the *Find Coverage* phase, *Find Implicants Phase* and *Expand Implicants* phase. The first two phases, together with the concept of the iterative minimization will be described in this section.

3.1 The Find Coverage Phase

The Find Coverage is the essential phase of the FC-Min algorithm. The whole cover of the on-set of the multi-output function is found, using the output part of the source function only. The algorithm tries to find a cover of the on-set by finding a rectangle cover [17] of all the “1” values in the output matrix, and then generates implicants having the properties given by this cover.

An example of such a cover is shown in Fig. 2. There is a 5-input and 5-output function defined by 10 terms shown, in a form of a truth table. The rest out of the total 32 terms is assigned as don’t cares. The result of the Find Coverage algorithm is a cover consisting of six *coverage elements*, $t_1 - t_6$. A coverage element is a Cartesian product of two sets, the *coverage set $C(t_i)$* and the *coverage mask $M(t_i)$*. The coverage set describes the rows that are covered by t_i, the coverage mask gives the output variables covered by t_i. Our example coverage elements are shown in Table 1.

Each coverage element describes properties of an implicant. For example, the *group* term (implicant) t_1 covers “1”s of the fourth and fifth output variable (y_3 and y_4) in the vectors 4, 6 and 8. Let us note that the structure of the terms is not known yet; only the set of covered “1”s is known. Now it is apparent, that if we succeed in finding the implicants having the properties of $t_1 - t_6$ (i.e., the terms cover the appropriate “1”s), the solution will consist of six implicants. To solve the coverage finding problem we use a greedy heuristic, since it is NP-hard [16].

```
11010 10000
10000 11100
01001 01100
01111 01010
00110 00111
01110 00000
10110 00011
00001 01101
10101 10111
11100 10100
```

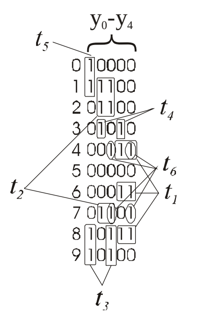

Figure 2: Cover of the output matrix

Table 1: Coverage elements Fig. 2

Implicant	$C(t_i)$	$M(t_i)$
t_1	{4, 6, 8}	$\{y_3, y_4\} \equiv 00011$
t_2	{1, 2, 7}	$\{y_1, y_2\} \equiv 01100$
t_3	{8, 9}	$\{y_0, y_2\} \equiv 10100$
t_4	{3}	$\{y_1, y_3\} \equiv 01010$
t_5	{0, 1}	$\{y_0, y_1\} \equiv 10000$
t_6	{4, 7}	$\{y_2, y_4\} \equiv 00101$

3.2 Implicant Generation Phase

The implicants forming the cover are generated in this phase. Considering the conditions described above, particularly the definition of the rows each cover element should cover ($C(t_i)$), a simple rule the implicants have to satisfy can be derived: the *minimum implicant* satisfying the particular cover can be constructed as a *minimum supercube* of all the input vectors corresponding to the rows of the cover of t_i. Moreover, this supercube must not intersect any term that is not included in the particular cover $C(t_i)$, since it would cover some zeros then. In our example, a minimum implicant t_1 would be `(-01--)`, because

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

The Implicant generation phase produces the minimal implicants, thus the implicants satisfying the above-described condition and having the maximum of literals. They can be further expanded to reduce the number of literals. This is being done by simply removing the literals from the implicants, when it is possible.

3.3 Iterative FC-Min

The FC-Min algorithm is not deterministic in most cases – the progress of the Find Coverage phase is controlled by a random number generator. Thus, a 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 a standard covering problem using all implicants. Even a properly selected combination of the implicants obtained from different iterations might produce a better solution.

4 Grouping of the Outputs

When the FC-Min Boolean minimizer emerged [11], the way to determine the output grouping has been set. The method is based on the idea of putting together output variables that have many common group implicants. Such output variables will more likely share some terms, thus grouping them together would be advantageous for a two-level minimization but we have found experimentally that the same effect can be observed for a multi-level synthesis as well.

The main output grouping idea is simple: first, we perform a two-level minimization of the unmodified multi-output function. Then, from the group implicants obtained, we identify the output variables to be grouped and finally we perform the rest of the synthesis, for the partitioned circuit. Any multi-output two-level minimizer, like BOOM [13] or ESPRESSO [17], can be used, however it is extremely advantageous to exploit FC-Min here: implicants that are shared among many outputs are immediately discovered, unlike when any other minimizer is used.

In the simplest case, only the Find Coverage phase needs to be used here (see Subsection 3.1) to obtain a satisfactory information to derive the output grouping. However, some covers computed in this phase may be not valid (no implicants can be produced for such a cover), and thus they can be misleading. Hence we execute the Find Implicants phase is being performed as well.

4.1 Grouping Matrix

The grouping of the outputs is derived from the valid coverage of the on-set. Since often there are big numbers of group implicants (coverage elements) and output variables, it is not easy to combine the influences of the implicants. We have found that an efficient way to estimate the grouping of the outputs is by constructing a *grouping matrix G*. It is a symmetric matrix of dimensions [*m*, *m*], where *m* is the number of output variables. The value G[*i, j*] defines the strength binding the two output variables *i* and *j* together.

The G-matrix is computed from the *temporary matrix G'*. The G' matrix is being constructed during the coverage generation process. Firstly, the matrix is filled with zeros. After each valid coverage element is produced, the values in all the positions in G' corresponding to all the couples of variables in $M(t_i)$ are increased by one. In our example (Fig. 2), after t_1 is found, the cells G'[3, 4] and G'[4, 3] are set to one. This describes an increased likelihood that the outputs y_3 and y_4 will be grouped together. The whole G' matrix computation process for our example is shown in Fig. 3.

$$\begin{matrix}00000\\00000\\00000\\00000\\00000\end{matrix}\xrightarrow{t_1}\begin{matrix}00000\\00000\\00000\\00001\\00010\end{matrix}\xrightarrow{t_2}\begin{matrix}00000\\00100\\01000\\00001\\00010\end{matrix}\xrightarrow{t_3}\begin{matrix}00100\\00100\\11000\\00001\\00010\end{matrix}\xrightarrow{t_4}\begin{matrix}00100\\00110\\11000\\01001\\00010\end{matrix}\xrightarrow{t_6}\begin{matrix}00100\\00110\\11001\\01001\\00110\end{matrix}$$

Figure 3: G'-matrix construction

It is a very simple example, however, in practice the G'-matrix mostly contains values bigger than 1. Bigger values indicate that the respective two variables have more than one common implicants in the solution.

The precision of the grouping can be improved by an Iterative FC-Min (see 3.3). To eliminate the effect of unequal numbers of terms in different solutions, the G'-matrix is transformed into a G-matrix in the following way: first, we find a minimal non-zero (g_{min}) and maximal (g_{max}) value in G'. Then, each cell in G is computed as:

$$G[i, j] = (G'[i, j] - g_{min}) / (g_{max} - g_{min}) \qquad (1)$$

Thus, all the values are transformed into the interval <0, 1>. (In our example, where $g_{max} = g_{min} = 1$, the transformation has no meaning.) This process will be called a *G-matrix normalization.* In the repeated FC-Min run, the G-matrices are being summed together.

We will continue constructing the G-matrix using our example. Let us assume that the FC-Min phase has been run one more time yielding a different solution as it is shown in Table 2. The solution consists of 7 terms.

Table 2: Different solution of the example

Implicant	$C(t_i)$	$M(t_i)$	PLA term & output
t_1	{4}	$\{y_2, y_3, y_4\} \equiv 00111$	`00110 00111`
t_2	{7}	$\{y_1, y_2, y_4\} \equiv 01101$	`00001 01101`
t_3	{3}	$\{y_1, y_3\} \equiv 01010$	`11010 10000`
t_4	{1, 8}	$\{y_0, y_2\} \equiv 10100$	`01111 01010`
t_5	{1, 2, 7}	$\{y_1, y_2\} \equiv 01100$	`1--0- 10100`
t_6	{4, 6, 8}	$\{y_3, y_4\} \equiv 00011$	`--00- 01100`
t_7	{0}	$\{y_0\} \equiv 10000$	`-01-- 00011`

Then the second G'-matrix will be constructed like we show in Fig. 4. E.g., for the t_1 term grouping together the variables y_2, y_3 and y_4 the cells G'[2, 3], G'[2, 4], G'[3, 4], G'[3, 2], G'[4, 2] and G'[4, 3] will be set to 1, since they represent all the combinations of the output variables.

00000 00000 00000 00000 00000	t_1 ---->	00000 00000 00011 00101 00110	t_2 ---->	00000 00101 01012 00101 01210	t_3 ---->	00000 00111 01012 01101 01210	t_4 ---->	00100 00111 11012 01101 01210	t_5 ---->	00100 00211 12012 01101 01210	t_6 ---->	00100 00211 12012 01102 01220

Fig. 4: G'-matrix construction (2)

The G'-matrix obtained contains values higher than 1, thus it has to be normalized using the formula (1), see Fig. 5. After that the two G'-matrices (see Fig. 3 and Fig. 5) are summed together, to obtain the final G-matrix, in Fig. 6.

```
0    0    0.5   0    0
0    0    1     0.5  0.5
0.5  1    0     0.5  1
0    0.5  0.5   0    1
0    0.5  1     1    0
```

Fig. 5: Normalized G'-matrix

```
0    0    1.5   0    0
0    0    2     1.5  0.5
1.5  2    0     0.5  2
0    1.5  0.5   0    2
0    0.5  2     2    0
```

Fig. 6: Normalized G'-matrix

4.2 Deriving the Output Grouping

After the G-matrix computing, the distribution of the function's outputs among the blocks has to be found. Let us assume that all the blocks have a limited strictly given number of outputs. The algorithm will proceed as follows: first, we find the *maximum* value in the G-matrix, let it be G[i, j]. When there are more possibilities for a choice, one is selected at random. Both the respective output variables (i, j) are assigned to the first block. After that we look for the next highest value in the i-th and j-th G-matrix rows. The new output variable corresponding to the selected column is added to the block under construction. This process is repeated until all the outputs of the block under processing are assigned. Then we repeat the process from the beginning, for the remaining blocks.

Let us assume that our example function is to be divided into two blocks, each having 3 outputs. Since the final G-matrix contains six maximums "2", one cell is selected at random, e.g. G[1, 2]. The variables y_1 and y_2 will be included into the first block. Then we search for the maximum value in the rows 1 and 2. The only possibility is G[2, 4], thus y_4 is included into the first block. The remaining two outputs are assigned to the second block. Thus, the decomposition will be [{y_1, y_2, y_4}, {y_0, y_3}].

The computational time needed to make the output assignment is negligible, comparing to the whole minimization process, even for a large number of output variables.

5 Experimental Results

All the experiments were performed on the standard "hard" MCNC benchmarks. For each of the benchmark circuits we have made three experiments:

- First, the respective benchmark circuit has been minimized by BOOM [13] and then decomposed into two-input gates, using SIS 1.2 [18]. The `script.rugged` has been used, together with the `tech_decomp -a 2` command to decompose the circuit into a network of 2-input gates. This experiment has been done to estimate the circuit size when the partitioning is used.

- In the second group of experiments we have divided the circuit into several blocks (*b*), while all the output variables were assigned to the individual blocks *purely at random*. Then the circuit has been minimized by BOOM-II, using the method described in [15]. After that we have performed a decomposition into a multi-level network, as described in the previous paragraph.
- Finally we have exploited our output grouping method, based on FC-Min. We have made a similar experiment to the previously described one, but the output variables were assigned to the blocks using the method proposed here. The FC-Min has been run for 10 iterations here.

These three experiments will show the differences between the all-in-one implementation of the benchmark circuit, the circuit divided into several blocks with randomly assigned outputs and our new method. The number of blocks was selected so to be the number of the inputs of the blocks approximately 10. However, any circuit may be divided into an arbitrary number of blocks. We have found experimentally, that the results do not vary significantly. We have observed a reduction of the number of the inputs entering the blocks when our method is used. Each input is entering each block in a worst case. When a proper decomposition is used, the number of inputs entering the blocks is reduced.

The benchmark results are shown in Table 3. After the benchmark name the numbers of the primary inputs (*i*) and outputs (*o*) of the circuit are presented. The next column gives the number of 2-inupt gates after the two-level minimization and decomposition by SIS. Next, there is the number of blocks, into which the circuit is being decomposed. The numbers of outputs of all the blocks are equal. The “Random output grouping” columns describe the minimization and decomposition results, for the experiment where the outputs are assigned to the blocks randomly. The “*inputs*” column describes the average number of inputs entering the blocks. The “FC-Min based output grouping” labeled columns describe the results obtained by our FC-Min based output grouping method. The last column, “*impr.*” shows the improvement against the previous (random) method.

We can observe that the improvement reaches almost up to 40% of total gates. Even the number of inputs is reduced when our method is used. Due to this fact, the decomposed circuit can be implemented into devices having fewer inputs than the original circuit has. Let us note that the same reduction of the area of the circuit decomposed by our method can be observed even when no minimization is performed. However, the number of inputs entering the blocks is not reduced.

Table 3: The experimental results

			No decomp.		Random output grouping		FC-Min based output grouping		
bench	*i*	*o*	*gates*	*blocks*	*gates*	*inputs*	*gates*	*inputs*	*impr.*
duke2	22	29	454	3	676	21.3	517	18.3	24%
jbp	36	57	422	6	755	25	600	20.2	21%
mainpla	27	54	3865	6	5742	25.2	4506	24.7	22%
mish	94	43	136	5	156	23.4	136	20.8	13%
misj	35	14	61	3	78	15.7	75	15	4%
soar	83	94	610	10	1424	35.6	970	26.7	32%
spla	16	46	427	5	1033	15.6	811	15.6	22 %
ti	47	72	763	7	1496	34	1173	23.7	22%
x2dn	82	56	193	6	331	23.5	206	16.5	38%

6 Conclusions

We have proposed a method of decomposition of a two-level circuit into several parts, by its outputs. The decomposition is based on a grouping of the outputs of the circuit, so that the output variables which share many group terms in the two-level representation of the minimized function are grouped together. The output grouping retains a two-level nature of the circuit, hence we call it a single-level partitioning.

The method is based on our two-level minimizer FC-Min, even when no minimization has to be involved here. A significant reduction of logic can be observed, when compared to the random output distribution. The same reduction of logic can be observed when the output grouping method is used without a subsequent two-level minimization. However, when the minimization is used, the number of inputs entering the blocks is reduced too. The method is efficiently applicable even when no two-level minimization is used. The results do not vary, even when a multi-level decomposition is used to synthesize the final logic.

Acknowledgement

This research was supported by a grant GA 102/04/2137 and MSM6840770014.

References

[1] Brayton, R., K., McMullen, C.T.: The Decomposition and Factorization of Boolean Expressions, In Proc. of the IEEE International Symposium on Circuits and Systems, pp. 49-54, 1982

[2] Muroga, S., Kambayashi, Y., Lai, J., C., Culliney, J., N.: The Transduction Method – Design of Logic Networks Based on Permissible Functions, IEEE Trans. on Computers, C-38(10), pp. 1404-1424, 1989

[3] Stanion, T., Sechen, C.: Boolean Division and Factorization using Binary Decision Diagrams, IEEE Trans on CAD, CAD-13(9), pp. 1179-1184, 1994

[4] Ashenhurst, R., L.: The Decomposition of Switching Functions, In Proc. of International Symposium on the Theory of Switching, pp. 74-116, 1957

[5] Roth, J., P., Karp, R., M.: Minimization over Boolean Graphs, IBM Journal of Research and Development, Vol. 6, No. 2, pp. 227-238, 1962

[6] Bryant, R., E.: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. on Computers, C-35(8), pp. 677-691, 1986

[7] Lai, Y., T., Pedram, M., Vrudhula, S.: BDD Based Decomposition of Logic for Functions with Applications to FPGA Synthesis, In Proc. Design Automation Conference, pp. 642-647, 1993

[8] Sasao, T., Butler, J., T.: On Bi-Decompositions of Logic Functions, ACM/IEEE International Workshop on Logic Synthesis, Tahoe City, California, 1997

[9] Mischenko, A., Steinbach, B., Perkowski, M.: An Algorithm for Bi-decomposition of Logic Functions, In Proc. of Design Automation Conference, pp. 103-108, 2001

[10] Jozwiak, L., Bieganski, S.: Information Trans-coders in Information-driven Circuit Synthesis, Proc. 30th Euromicro Symposium on Digital Systems Design (DSD'04), Rennes (FR), 31.8. - 3.9.04, pp. 288-297

[11] Fišer, P., Hlavička, J., Kubátová, H.: FC-Min: A Fast Multi-Output Boolean Minimizer, Proc. 29th Euromicro Symposium on Digital Systems Design (DSD'03), Antalya (TR), 1.-6.9.2003, pp. 451-454

[12] 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

[13] Hlavička, J., Fišer, P.: BOOM - A Heuristic Boolean Minimizer, Computers and Informatics, Vol. 22, 2003, No. 1, pp. 19-51

[14] Hlavička, J., Fišer, P.: A Flexible Minimization and Partitioning Method. Proc. 5th Int. Workshop on Boolean Problems, Freiberg (Germany) 19.-20.9.2002, pp. 83-90

[15] Fišer, P., Kubátová, H.: Single-Level Partitioning Support in BOOM-II, Proc. 2nd Descrete-Event System Design 2004 (DESDes'04), Dychów, Poland, 15.-17.9.04, pp. 149-154

[16] Fišer, P., Kubátová, H.: Boolean Minimizer FC-Min: Coverage Finding Process, Proc. 30th Euromicro Symposium on Digital Systems Design (DSD'04), Rennes (FR), 31.8. - 3.9.04, pp. 152-159

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

[18] Sentovich, E., M., et al.: SIS: A System for Sequential Circuit Synthesis, Electronics Research Laboratory Memorandum No. UCB/ERL M92/41, University of California, Berkeley, CA 94720, 1992