Fast Boolean Matching Under Permutation Using Representative

Debatosh Debnath and Tsutomu Sasao
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820-8502, Japan

Abstract—This paper presents an efficient method to check the equivalence of two Boolean functions under permutation of the variables. The problem is also known as *Boolean matching*. As a basis of the Boolean matching, we use the notion *P-representative*. If two functions have the same P-representative then they match. We develop a breadth-first search technique to quickly compute the P-representative. On an ordinary workstation, on the average, our method requires several microseconds to test the Boolean matching for functions with up to eight variables. This approach is promising for Boolean matching of multiplexor-based field-programmable gate arrays (FPGAs) and for library matching with many large cells.

Index Terms—Boolean matching, technology mapping, variable permutation, P-equivalence.

I. Introduction

Boolean matching is a technique to detect the equivalence of two Boolean functions under permutation of the variables. One of the main application of Boolean matching is in *cell-library binding* [6]. An exhaustive method for Boolean matching is computationally infeasible even for functions with several variables. Thus, efficient Boolean matching algorithms for cell-library binding have been developed [1, 2, 7, 12, 13, 16, 17]. Boolean matching is also useful for logic verification where the correspondence of the inputs of the two circuits are unknown [10, 14, 18, 19]. Boolean matching is also effectively used in other areas of logic synthesis, such as in the design of AND-OR-EXOR three-level networks [4, 5].

In this paper, we present an efficient Boolean matching algorithm for cell-library binding. As a basis of the Boolean matching we use P-representative, which is unique among the functions of a P-equivalence class. The set of functions that are invariant under permutation of the variables form a *P-equivalence class* [8, 9, 15]. In a P-equivalence class the function that has the smallest binary representation is the *P-representative* of that class. Every P-equivalence class has a unique P-representative. Thus, if the P-representatives for the two functions are the same then one can be transformed into another by changing permutation of the variables.

To match against a library, our method computes the P-representatives for all the library cells and stores them in a hash table during a *setup phase*. Then computes the P-representatives for the functions to be matched and checks the hash table for the same P-representatives during *matching phase*. During setup phase for multiplexor-based field-programmable gate arrays (FPGAs), it generates a library with all the cells that an FPGA module can implement by bridging the inputs and setting the inputs to constants.

Important features of our method are as follows:

- First of all, P-representative is a powerful notion because it is *unique* for any P-equivalence classes. Burch and Long introduced a *semi-canonical form* for matching under permutation of the variables [1]. However, semi-canonical form is nonunique. As a basis of the Boolean matching many algorithms use *signatures*, which are mainly computed from some properties of the functions. Although signatures are extensively used in Boolean matching [10, 12, 14, 16, 17], they are unable to uniquely identify many P-equivalence classes. Thus, an exhaustive search is necessary to obtain a conclusive result [16, 17]. However, P-representative based method always gives conclusive result without any exhaustive search.
- The computational complexity of our method is independent of the number of cells in the library. It can efficiently handle libraries with extremely large number of cells. The number of cells is constrained only by the available memory resources. This feature is very important in table look-up based synthesis [8], where matching against a library with more than one million cells may necessary [4, 5]. Next, an increase in the number and in the size of the cells in a library improves the quality of the mapped circuits [11, 17]. However, Boolean matching for large libraries is computationally expensive. On the other hand, our method efficiently deal with large libraries.
- Our method is fast and flexible. Experimental result shows that it is about 40 times faster than the method in [17]. It can be used with *filters* [3, 10, 13] to further reduce the matching time.

- Cells with sufficiently large number of inputs can be handled by our method. The present implementation can treat cells with up to eight inputs. Its practical upper limit is nine-input cells.
- For up to seven-variable functions the method requires only about 500 kilobytes memory. For functions with up to eight variables the memory requirement is about 10 megabytes.
- P-representative is very simple and compact.
- Computation of any functional properties is unnecessary in our method. Thus, it makes our method independent of any cell architecture and simplifies the programming task. Many Boolean matching algorithms heavily depends on functional properties to reduce the computation time [10, 12, 16, 18].

The remainder of the paper is organized as follows: Section II introduces terminology. Section III develops the technique to compute the P-representative, which is the basis of our Boolean matching algorithm. Section IV reports the experimental results. Section V presents conclusions.

II. Definitions and Terminologies

Definition 2.1 *The minterm expansion of an n-variable function is* $f(x_1,x_2,\ldots,x_n) = m_0 \cdot \bar{x}_1\bar{x}_2\cdots\bar{x}_n \vee m_1 \cdot \bar{x}_1\bar{x}_2\cdots x_n \vee \cdots \vee m_{2^n-1} \cdot x_1x_2\cdots x_n$, *where* $m_0,m_1,\ldots,m_{2^n-1} \in \{0,1\}$. *The* 2^n *bit binary number* $m_0m_1\cdots m_{2^n-1}$ *is the* ***binary representation*** *of* f. *To denote a binary number, a subscripted 2 is used after it.*

Example 2.1 Consider the three-variable function $f(x_1,x_2,x_3) = \bar{x}_1\bar{x}_2\bar{x}_3 \vee x_1$. The binary representation of f is 10001111_2.

Logic functions can be grouped into classes by using simple transformations.

Definition 2.2 *Two functions* f *and* g *are* ***P-equivalent*** *if* g *can be obtained from* f *by permutation of the variables [8, 9, 15].* $f \overset{P}{\sim} g$ *denotes that* f *and* g *are P-equivalent. P-equivalent functions form a* ***P-equivalence class*** *of functions.*

Example 2.2 Consider the three functions: $f_1(x_1,x_2,x_3) = \bar{x}_2\bar{x}_3 \vee x_1x_2x_3$, $f_2(x_1,x_2,x_3) = \bar{x}_1\bar{x}_3 \vee x_1x_2x_3$, and $f_3(x_1,x_2,x_3) = \bar{x}_1\bar{x}_2 \vee x_1x_2x_3$. Since $f_2(x_2,x_1,x_3) = \bar{x}_2\bar{x}_3 \vee x_1x_2x_3 = f_1(x_1,x_2,x_3)$, we have $f_1 \overset{P}{\sim} f_2$, and since $f_3(x_1,x_3,x_2) = \bar{x}_1\bar{x}_3 \vee x_1x_2x_3 = f_2(x_1,x_2,x_3)$, we have $f_2 \overset{P}{\sim} f_3$. Therefore, the functions f_1, f_2, and f_3 belong to the same P-equivalence class.

Definition 2.3 *The function that has the smallest binary representation among the functions of a P-equivalence class is the* ***P-representative*** *of that class.*

Example 2.3 All the functions of the P-equivalence class for $\bar{x}_2\bar{x}_3 \vee x_1x_2x_3$ are $f_1(x_1,x_2,x_3) = \bar{x}_2\bar{x}_3 \vee x_1x_2x_3$, $f_2(x_1,x_2,x_3) = \bar{x}_1\bar{x}_3 \vee x_1x_2x_3$, and $f_3(x_1,x_2,x_3) = \bar{x}_1\bar{x}_2 \vee x_1x_2x_3$. In binary representation: $\bar{x}_2\bar{x}_3 \vee x_1x_2x_3 = 10001001_2$, $\bar{x}_1\bar{x}_3 \vee x_1x_2x_3 = 10100001_2$, and $\bar{x}_1\bar{x}_2 \vee x_1x_2x_3 = 11000001_2$. Since $10001001_2 < 10100001_2 < 11000001_2$, the P-representative of this class is $\bar{x}_2\bar{x}_3 \vee x_1x_2x_3$.

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

(a) Truth-table for $f(x_1,x_2,x_3)$.

x_3	x_2	x_1	$f(x_3,x_2,x_1)$
0	0	0	m_0
0	0	1	m_4
0	1	0	m_2
0	1	1	m_6
1	0	0	m_1
1	0	1	m_5
1	1	0	m_3
1	1	1	m_7

(b) Truth-table for $f(x_3,x_2,x_1)$.

Fig. 1. Two different permutations of the variables for three-variable function.

For an n-variable function, there are at most $n!$ P-equivalents. Among them, our objective is to find the P-equivalent that has the smallest binary representation as fast as possible.

III. Computing P-Representative

In this section, we show a method to compute P-representative, mainly, by using three-variable function. The extension to the functions with more variables is straightforward.

3.1. A Naive Method

The truth-table for a three-variable function $f(x_1,x_2,x_3)$ is shown in Fig. 1(a), where $m_0,m_1,\ldots,m_7 \in \{0,1\}$. We want to prepare the truth-table for $f(x_3,x_2,x_1)$ in Fig. 1(b). We do this by copying the minterms in Fig. 1(a) to Fig. 1(b), such that $f(a,b,c)$ in Fig. 1(a) and $f(c,b,a)$ in Fig. 1(b) become the same, where $a,b,c \in \{0,1\}$. The permutation of the variables for the functions in Fig. 1(a) and Fig. 1(b) are (x_1,x_2,x_3) and (x_3,x_2,x_1), respectively. Similarly, we can generate functions with other permutations of the variables, and take the function that has the smallest binary representation as the P-representative.

A close observation to the minterms in Fig. 1 reveals that many of the minterms of $f(x_1,x_2,x_3)$ moved to new positions in $f(x_3,x_2,x_1)$. For example, the fifth minterm, m_4, of $f(x_1,x_2,x_3)$ becomes the second minterm of $f(x_3,x_2,x_1)$. Note that each time we want to change the permutation of the variables of an n-variable function, we must compute the new positions for all the 2^n minterms. An n-variable function have at most $n!$ P-equivalents. Thus, to compute the P-representative for an n-variable function we must compute $n!2^n$ new positions for the minterms. As a result, the method requires excessive amount of computation time even for functions with several variables.

3.2. Using Precomputed New Minterm Positions

The variables of $f(x_1,x_2,x_3)$ can be permuted in six ways: (x_1,x_2,x_3), (x_1,x_3,x_2), (x_2,x_1,x_3), (x_2,x_3,x_1), (x_3,x_1,x_2),

$f(x_1,x_2,x_3)$	$f(x_1,x_3,x_2)$	$f(x_2,x_1,x_3)$	$f(x_2,x_3,x_1)$	$f(x_3,x_1,x_2)$	$f(x_3,x_2,x_1)$
m_0	m_0	m_0	m_0	m_0	m_0
m_1	m_2	m_1	m_4	m_2	m_4
m_2	m_1	m_4	m_1	m_4	m_2
m_3	m_3	m_5	m_5	m_6	m_6
m_4	m_4	m_2	m_2	m_1	m_1
m_5	m_6	m_3	m_6	m_3	m_5
m_6	m_5	m_6	m_3	m_5	m_3
m_7	m_7	m_7	m_7	m_7	m_7

Fig. 2. All possible P-equivalents of a three-variable function $f(x_1,x_2,x_3)$.

and (x_3,x_2,x_1). Fig. 2 shows a three-variable function $f(x_1,x_2,x_3)$ and its all possible P-equivalents. We can say that Fig. 2 shows the *new minterm positions* which can be used to generate P-equivalents. Thus, by using the precomputed new minterm positions of Fig. 2 we can easily generate all the P-equivalents of any given three-variable function. This method is much faster than the naive method of Section 3.1, because computation of the new positions for the minterms is unnecessary.

3.3. Using Breadth-First Search

For an n-variable function, the above method first generates $n!$ functions and then chooses the function that has the smallest binary representation as the P-representative. Since we are interested only in the function that has the smallest binary representation, we use breadth-first search technique for early detection of the variable permutation that cannot lead to the P-representative. We discard the variable permutation from the consideration as soon as we detect that it cannot lead to the P-representative. Note that breadth-first search technique is difficult to apply if Fig. 2 is unavailable.

3.4. A More Efficient Method

The above method uses breadth-first search on the new minterm positions in Fig. 2. The method is fast; however, we can further speed-up the computation.

For all the functions in Fig. 2 the first minterms are the same. Thus, in breadth-first search any comparison is unnecessary for these minterms. Next we consider the second minterms for all the functions in Fig. 2. The usual way is to generate all the second minterms, and then retain only the variable permutations that have the smallest value for the second minterms and discard other variable permutations. However, if we partition all the variable permutations in Fig. 2 into three sets, $\{(x_1,x_2,x_3),(x_2,x_1,x_3)\}$, $\{(x_1,x_3,x_2),(x_3,x_1,x_2)\}$, $\{(x_2,x_3,x_1),(x_3,x_2,x_1)\}$, then instead of generating all the second minterms we need to generate only three second minterms, because for each of these sets the second minterms are the same. We retain only the variable permutations that have the smallest value for the second minterms and discard other variable permutations. Then the search continues with the third minterms in Fig. 2. Thus, we can reduce the computation time for the second minterms by a factor of 2 ($= 3!/3$). By using this technique for the n-variable functions, we can reduce the computation time for the second minterms by a factor of $n!/n$. For functions with more than three variables this technique is very effective to reduce the computation time, because we can recursively partition the variable permutations. In general, for n-variable functions, we can partition the variable permutations into n sets at first, then each of these sets can be again partitioned into $n-1$ sets; we can recursively partition each of these sets until the cardinality of the sets become one. As a result, we can drastically reduce the computation time for many other minterms.

We incorporate this idea to find P-representative as the traversal of a *breadth-first search tree*, which also uses the new minterm positions in Fig. 2. During *setup phase* of the Boolean matching we build this tree, which is the main data structure of our algorithm. Each node of the breadth-first search tree has multiple children, and the number of children of a node depends on the level of the node and on the number of variables. For n-variable functions, the tree has n levels. Let the root node of the tree is at level 1. Thus, the leaf nodes are at level n. The number of children of a node at level k is $n-k+1$, where $1 \le k < n$. Thus, the total number of nodes in the tree is $1 + n + n(n-1) + n(n-1)(n-2) + \cdots + n(n-1)(n-2)\cdots 4 \cdot 3 \cdot 2$.

3.5. Some Properties of New Minterm Positions

The new minterm positions in Fig. 2 shows that the first minterms are the same for all the functions. This is also true for the last minterms. Fig. 2 also shows that, for $n = 3$ if the i-th minterm is m_k, then the $(2^n - i + 1)$-th minterm be m_{2^n-k-1}, where $1 \le i \le 2^{n-1}$. These properties are useful to reduce the computation time and to save memory resources.

IV. Experimental Results

We implemented the proposed method of Boolean matching for functions with up to eight variables on an HP C160 workstation. It consists of about 3000 lines of C code and about five megabytes of dynamically linked data for the new minterm positions. The program requires about 10 megabytes memory, most of which is used for the matching of functions with eight variables. If the program is used for the functions with up to seven variables, it needs about 500 kilobytes memory. During setup phase the program constructs the breadth-first search trees; it takes about 50 milliseconds.

To demonstrate the effectiveness of our matching technique, we conducted an experiment by using 2,000,000 pseudo-random functions with three to eight variables

TABLE I
AVERAGE TIME FOR BOOLEAN MATCHING

Number of variables	Time (microseconds)
3	2.28
4	4.96
5	7.09
6	9.30
7	13.56
8	21.07

and tried to match them against a library with 20,000 randomly generated cells. The pseudo-random functions for each variable are generated such that about five percent of them can find a match in the library. We computed the P-representatives for all the library cells and stored them in a hash table during the setup phase. Then the P-representatives for all the pseudo-random functions are computed and compared with the P-representatives for the library cells. Table I summarizes the average Boolean matching time in microseconds. Note that, it is the time to match a function against the entire library cells.

Schlichtmann, Brglez, and Schneider [17] reported that, for three-, four-, five-, six-, seven-, and eight-variable functions the Boolean matching time is approximately 2.0, 4.0, 6.0, 8.0, 12.0, and 18.0 milliseconds, respectively, on a DECStation 5000/200. An HP C160 workstation is about 20 times faster than a DECStation 5000/200. Thus, taking the speed differences of the two machines into account, our method is about 40 times faster than the method in [17].

V. CONCLUDING REMARKS

In this paper, we used the notion P-representative, which is unique for any P-equivalence classes, and presented a breadth-first search algorithm for its quick computation. We used P-representatives to efficiently check the equivalence of two Boolean functions under permutation of the variables. The concept P-representative is extended to the N-, NP-, and NPN-equivalence classes [8, 9, 15]; and a breadth-first search approach is also devised to compute their respective representatives. The preliminary results are promising. Our Boolean matching technique can handle very large libraries. In addition, our method is fast and flexible. It can be used with filters [3, 10, 13, 16] to further reduce the computation time. Our future work is to develop a cell-library binding system by using the proposed matching technique.

ACKNOWLEDGEMENT

This work was supported in part by a Postdoctoral Fellowship of the Japan Society for the Promotion of Science and in part by a Grant-in-Aid for the Scientific Research of the Ministry of Education, Science, Culture, and Sports of Japan.

REFERENCES

[1] J. R. Burch and D. E. Long, "Efficient Boolean function matching," *Proc. International Conference on Computer-Aided Design*, pp. 408–411, Nov. 1992.

[2] K.-C. Chen, "Boolean matching based on Boolean unification," *Proc. European Design Automation Conference*, pp. 346–351, Sept. 1993.

[3] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," *Proc. Design Automation Conference*, pp. 54–60, June 1993.

[4] D. Debnath, *On the Minimization of AND-EXOR and AND-OR-EXOR Networks*, Doctoral Dissertation, Kyushu Institute of Technology, Dept. of Computer Science and Electronics, March 1998.

[5] D. Debnath and T. Sasao, "A heuristic algorithm to design AND-OR-EXOR three-level networks," *Proc. Asia and South Pacific Design Automation Conference*, pp. 69–74, Feb. 1998.

[6] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[7] S. Ercolani and G. De Micheli, "Technology mapping for electrically programmable gate arrays," *Proc. Design Automation Conference*, pp. 234–239, June 1991.

[8] M. A. Harrison, *Introduction to Switching and Automata Theory*, McGraw-Hill, 1965.

[9] S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic*, Academic Press Inc., 1985.

[10] Y.-T. Lai, S. Sastry, and M. Pedram, "Boolean matching using binary decision diagrams with applications to logic synthesis and verification," *Proc. International Conference on Computer Design*, pp. 452–458, Oct. 1992.

[11] C. Liem and M. Lefebvre, "Performance directed technology mapping using constructive matching," *Proc. International Workshop on Logic Synthesis*, May 1991.

[12] F. Mailhot and G. De Micheli, "Algorithms for technology mapping based on binary decision diagrams and on Boolean operations," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-12, No. 5, pp. 599–620, May 1993.

[13] Y. Matsunaga, "A new algorithm for Boolean matching utilizing structural information," *IEICE Trans. Information and Systems*, Vol. E78-D, No. 3, pp. 219–223, March 1995.

[14] J. Mohnke and S. Malik, "Permutation and phase independent Boolean comparison," *Proc. European Conference on Design Automation*, pp. 86–92, Feb. 1993.

[15] S. Muroga, *Logic Design and Switching Theory*, John Wiley & Sons, 1979.

[16] U. Schlichtmann, F. Brglez, and M. Hermann, "Characterization of Boolean functions for rapid matching in EPGA technology mapping," *Proc. Design Automation Conference*, pp. 374–379, June 1992.

[17] U. Schlichtmann, F. Brglez, and P. Schneider, "Efficient Boolean matching based on unique variable ordering," *Proc. International Workshop on Logic Synthesis*, pp. 3b:1–3b:13, May 1993.

[18] C. Tsai and M. Marek-Sadowska, "Boolean functions classification via fixed polarity Reed-Muller forms," *IEEE Trans. Comput.*, Vol. C-46, No. 2, pp. 173–186, Feb. 1997.

[19] K.-H. Wang, T. Hwang, and C. Chen, "Exploiting communication complexity for Boolean matching," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-15, No. 10, pp. 1249–1256, Oct. 1996.