Fast and Effective P-Matching Method Based on Bipartite Graph

Anderson Santos da Silva, André Reis, Renato Ribas

Institute of Informatics, Federal University of Rio Grande do Sul
Av. Bento Gonçalves 9500, Porto Alegre, RS, Brazil

{assilva,rpribas,andreis}@inf.ufrgs.br

***Abstract* — In logic synthesis, the technology mapping process can be a very time consuming task when fitting cells into a target library. This work exploits the representation of Boolean functions through bipartite graph to propose a fast and quite effective way to calculate their P-matching equivalence. The proposed method applies reduction rules simultaneously over two graphs in order to verify the isomorphism property that is the condition for P-matching. A fast algorithm is developed based on such strategy, and experimental results have demonstrated that this method runs in linear time in most cases.**

I. INTRODUCTION

In integrated circuit (IC) design methodology, the standard cell flow is still playing a major role [1]. This flow is divided in several steps or tasks, in which the logic synthesis uses P-matching to find out equivalent cells in a target library to map part of the functionality of the circuit. This problem, named Boolean P-matching, determines when two functions are equivalent under the permutation of its variables [2]. As a result, this task has to be executed as fast as possible being that such speed is intrinsically associated to the representation form applied in process. In the way to speed up this process, several methods have been proposed in the literature to solve it [2][3]. Related to this issue, the data structure adopted is a big concern to guarantee short execution time and scalability. In some cases, structures based on elementary representation of truth-table are applied. This elementary representation brings all limitations of truth-table such as the practical number of inputs limit inherent to this structure. Therefore, a good model to represent a function in Boolean matching context is needed.

Boolean functions have several forms of representation. These forms are usually centralized to take the benefit of some function properties. For instance, truth-table representation is addressed for small number of inputs (variables), so being a good way to verify all input combinations. On the other hand, binary decision diagram (BDD) data structure is useful in cofactor exploitation point-of-view [4]. Graph representations can be exploited to describe the function in the structural domain. Moreover, in some cases, these graphs are used to minimize the expression form of Boolean function [5]. In other cases, they are used to map logic gates into standard cell design flow [6]. Indeed, these forms of representation have been developed for specific context in order to explore the facilities provided by each data structure.

This paper proposes a new way to generate a graph-based model for describing Boolean function, and it is exploited in a novel algorithm to solve the P-matching favouring the Boolean matching context. As the only essential information for P-matching of function from truth-table representation is maintained, the bipartite graph representation exploited herein is naturally more compact. The basic idea of our approach is to generate a bipartite graph with minterms and variables, described according to the function behavior. With this graph scheme, we represent the functions involved in the P-matching verification, in their graph form. Such verification is based on the application of rules for reducing them simultaneously. If each reduction step presents the same effect in each graph until the end (i.e., complete reduction to nothing), one can that they are isomorphic graphs and, consequently, P-equivalent Boolean functions. As such reduction occurs simultaneously in both graphs, a structural check can be performed over their topologies. This reduction algorithm is very promising because, in most of cases, the matching is done in linear time. In other cases, the computation time can be more than exponential, but it is a property of any NP-complete problem such as Boolean matching [5].

The structure of this paper is the following. Section II presents a brief technical background review about the main concepts for a better understanding of this approach. Section III describes the minterm-variable bipartite graph used herein. Section IV discusses the proposed algorithm, how to generate a minterm-variable graph for different Boolean functions, how to reduce such graph in comparison to others, and some examples. Section V provides some results in the generation of permutation classes, as well as the analysis of the algorithm complexity. Section VI outlines the conclusions.

II. TECHNICAL BACKGROUND

A. Graph Isomorphism

If two graphs, G_1 and G_2, has a function $f: G_1 \rightarrow G_2$ such that for every edge in G_1 with nodes u and v there is an edge in G_2 with nodes $f(u)$ and $f(v)$, then they are isomorphic graphs.

For instance, the bijection functions illustrated in Fig. 1 are P-equivalent according to the following:

$$f(A) = B;\ f(C) = C;\ f(B) = A;\ f(D) = D.$$

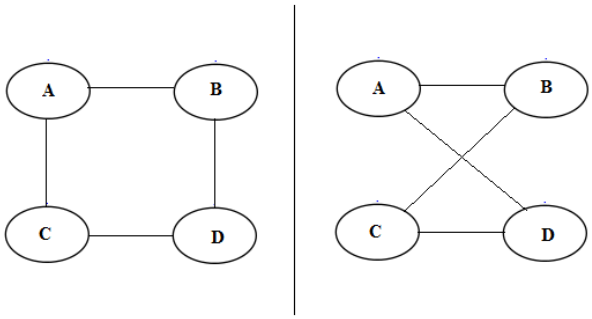

Figure 1 - Example of two isomorphic graphs.

B. P-Matching

P-matching is the operation over two functions, f_1 and f_2, that determines if there is a permutation of variables in f_1 such that this function turns into f_2. [5]. If a P-matching exists between f_1 and f_2, the function f_1 is called P-equivalent with f_2.

For instance, the functions f_1=*0x119f* and f_2=*0x03d7* are P-equivalent. The expressions associated to both can be the following:

$$f_1 = ((!c * !d) + (!a * (!b + (c * d))))$$

$$f_2 = ((!b * !c) + (!a * (!d + (b * c))))$$

Notice that :

- a in f_1 turns on a in f_2;
- b in f_1 turns on d in f_2;
- c in f_1 turns on b in f_2;
- d in f_1 turns on c in f_2.

C. Graph Adjacency Matrix

Given a graph $G\ (V,E)$, where V is the set of nodes in graph, and E is the set of edges in G:

Adjacency matrix is the $n \times n$ matrix, where n is the cardinality of set V, denoted by $|V|$, where each element a_{ij} is 1 if an edge exists between i and j, and 0 otherwise. An adjacency matrix of two isomorphic graphs is illustrated in Table I.

Table I - Example of two isomorphic graphs.

	a	**b**	**c**	**d**
a	0	1	1	0
b	1	0	0	1
c	1	0	0	1
d	0	1	1	0

III. MINTERM-VARIABLE GRAPH

A Boolean function with n variables and one output is defined as $f: B^n \rightarrow B$, where $B = \{0,1\}$.

Support of a Boolean function with n variables is a set $A = \{a_1, a_2, a_3, \ldots a_n\}$ of its variables.

Minterms of Boolean function is the set:

$$B^n = \{ (m_1, m_2, \ldots, m_n)_1, (m_1, m_2, \ldots, m_n)_2, . (m_1, m_2, \ldots, m_n)_{2n}\}$$

where m_i belongs to B.

The graph $G\ (V,E)$, where V is the set of nodes in graph and E is the set of edges in graph associated with a Boolean function, is defined as follows:

$$V = \{A \text{ union } B^n\}$$

and

$$E = \{e(a_i, (m_1, m_2, \ldots, m_n)_k) / a_i \text{ belongs to } A \text{ and } (m_1, m_2, \ldots, m_n)_k \text{ belong to } B^n \text{ and } f((m_1, m_2, \ldots, m_n)_k) = 1 \text{ and } m_i = 1\}$$

where $i \leq n$ and $k \leq 2n$, and $e(i,j)$ is an edge between nodes i and j. This graph is named 'minterm-variable graph'.

The graph construction process can be illustrated with the following example. Suppose that we have the function *f=0x27* represented through the truth-table in Table II.

Table II – Truth-table for *f = 0x27*.

	X0	X1	X2	**f**
0:	0	0	0	**1**
1:	0	0	1	**1**
2:	0	1	0	**1**
3:	0	1	1	**0**
4:	1	0	0	**0**
5:	1	0	1	**1**
6:	1	1	0	**0**
7:	1	1	1	**0**

The lines that turn the function to '0' are not a concern, and can be discarded from the truth-table, so generating a reduced truth-table as shown in Table III.

Table III – Truth-table for *f = 0x27*
(the 1 induces the edge 2 to X1).

	X0	X1	X2	**f**
0:	0	0	0	**1**
1:	0	0	1	**1**
2:	0	1	0	**1**
3:	1	0	1	**1**

Such reduced truth-table seems like an adjacency matrix of graph. Every '1' value induces an edge between minterm and variable in the same line and column of 1 value. If we repeat this process for all '1' in truth-table, we obtain the graph show in Fig. 2.

This generated graph is useful to calculate P-matching because it structure is unique for all functions P-equivalent. However, a general graph can be drawn and labeled in several ways maintained its structure. Graphs in this case are named isomorphic because they have a bijection in their structure. In computer science, an efficient algorithm for graph

isomorphism is an open problem, and no proof of hardness of its problem (NP-completeness) can be encountered. Therefore, there is a room for investigation in this subject.

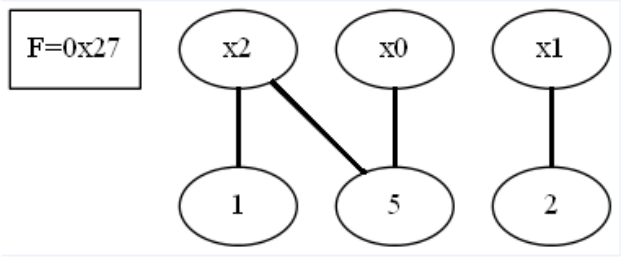

Figure 2 - Graph associated to the function in Table III.

IV. PROPOSED P-MATCHING METHOD

The proposed P-matching algorithm uses a minterm-variable graph structure to represent a Boolean function, and the fact that the nodes of this graph can be divided into two subsets: the 'tentacle' subset and the 'cycle' subset.

A. *Tentacle Graph Subset*

Every node with degree equal to one and all reachable nodes that have degree equal to two is a node in a 'tentacle' graph pattern. Fig. 3 illustrates a graph with a 'tentacle' composed by the nodes in white color. The black nodes belong to another graph subset, defined in the next section.

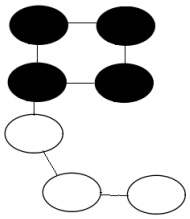

Figure 3 - Example of graph with a 'tentacle' and a 'cycle'.

B. *Cycle Graph Subset*

Every node that does not belong to a 'tentacle' is place on a 'cycle'. For instance, this kind of node is represented by the nodes in black in Fig. 3.

C. *Graph Code*

Every node in a minterm-variable graph has a set of integer numbers corresponding to the code that represents the information of its neighbors. Here, we have divided the set of nodes into two subsets of type of code:

- Nodes with degree lower than 2 – when a node is simply removed, it sums the removed node code with the top of stack of adjcacents nodes.
- Nodes with degree greater than 2 – when a node is removed, it inserts its code on the top of the stack of its adjacent node, growing in one level the stack size.

D. *Equivalence Check*

Given two Boolean functions in their minterm-variable graph representation, a reduction of these graphs can be performed as follows:

1) Generate the graph associated to these functions, $G_1$and G_2.
2) Detect the tentacles in G_1 and G_2, and check if both has the same number of tentacles.
3) If there is a tentacle:
   - Remove the **variables** nodes of tentacle that have degree one in both graphs; check if the number of removed nodes is the same.
   - Remove the **minterms** nodes of tentacle that have degree one in both graphs; check if the number of removed nodes is the same.
4) If there is not a tentacle:
   - Chose a node with unique code to remove in both graphs, and then remove it.
   - If there is not this unique node, choose one node in G_1 and test its deletion with all nodes tied with this in G_2.
5) In each deletion, propagates a code graph of node removed to the adjacent node.
6) Check if the removed nodes are equivalent in code and degree.
7) Repeat until the graph is empty or no equivalent deletion was encountered.

If both graphs are empty in final step, the functions are P-equivalent. Otherwise, they are not P-equivalent.

In the way to better explain how this algorithm works, two examples are presented:

Example 1: The function *f=0x68* and *f=0x68*, shown in Fig. 4, are P-equivalent because is the same function. If every node ties in its code, then we get all tied nodes in both graph, and put them on a set T1 associated with one graph and T2 associated with other graph. Then, extracting an element *t* of T1 and for all element *e* in T2 creates a pair (*t*,*e*), removing this pair in both graphs, and repeating the algorithm with the rest of the graph. If some pair returns true in its reduction, there is a matching in these graphs.

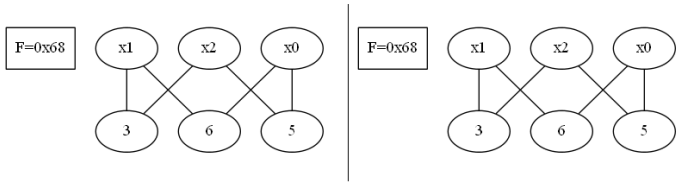

Figure 4 - Graph reduction: lower case.

In this example, we generate the set of tied nodes G1 = {x1,x2,x0,3,6,5} and G2 = {x1,x2,x0,3,6,5}. And we try to match a node in G1 to a node in G2. Maintaining the node 'x1', the possibilities are (x1,x1), (x1,x2), (x1,x0). Since variable node only match with variable node, x1 should have a variable that match with itself. A backtracking is performed in

these cases in order to find at least one pair that matches. Notice that to know if a pair matches, it is needed to run the algorithm until the end. Therefore, if one pair matches, other pairs do not need to be tested.

Example 2: Considering the function *f=0x81* with *f=0x86*, illustrated in Fig. 5. Such functions do not match because their graphs are not isomorphic. These functions do not have cycle. It is the best case of matching because the 'tentacles' are removed very fast in linear time in number of nodes.

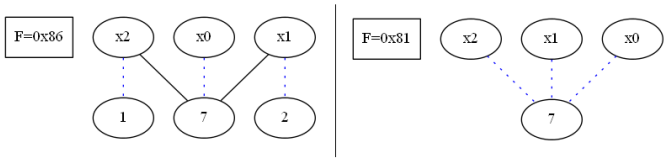

Figure 5 - Graph reduction: best case.

V. Experimental Results

The algorithm has been validated with the generation of all P-class functions with 2-, 3- and 4-input [5], in order to verify its correctness and efficiency.

In the way to reach more than 4-input functions, experiments with 5-input NPN class functions have been carried out, and every function was compared against to all other functions in the same class. Although 5-input NPN class set has 616,125 functions, this test spent around two days in computation time and no error has been detected. The results are show in Table IV. This table demonstrates that only few functions in the entire set were actually computation time consuming during the evaluation. Other random tests were performed considering up to 19-input functions. An investigation of which functions represent the bottlenecks for this approach was also made

Table IV – Function profile of linear and non-linear sets.

Set of functions	Number of variables			
	Up to #vars	**Total**	**Non Linear**	**% in total**
All 2 input	2	16	0	0
All 3 input	3	256	44	17.18
All 4 input	4	65536	9376	14.30
5 NPN	5	616125	42936	6.96

The results shown in Table IV are very promising. The larger number of functions presents linear computation time of the Boolean matching. Less than 20% of cases are actually time consuming using the proposed procedure.

In terms of algorithm complexity analysis, the following considerations can be done. Assuming a graph $G(V,E)$, it is known that V can be subdivided in two subsets: x in V that is on a 'tentacle', and y in V that is on a 'cycle'. We named the nodes in 'tentacle' as T, and C to nodes in cycles, since C is disjoint of T. The graph G can have several 'tentacles' and 'cycles'. Every 'tentacle' in T is reduced in time proportional to its number of nodes. In lower cases, it can be the entire V. Then, in these cases, we have $O(|V|)$ complexity, where $|V|$ represents the cardinality of set V.

In the case of 'cycles', the lower cases to number of nodes in C is $|C|=|V|$, and an exhaustive search is done. This exhaustive search uses at maximum $O(|V|)$ steps.

Therefore, a typical graph has a combination of these cases, but if it is a 'tentacle' or a 'cycle', the algorithm resolves the match in linear time $O(|V|)$. In the case of combination of both, and in presence of so many ties, the natural recursion of this method uses $O(|k1|*|k2|\ldots*|kn|)$, where $\sum ki = |V|$, $n \rightarrow 0$. It says that this approach runs in super-exponential time in lower cases, as every NP-complete problem like P-matching. However, as show in Table IV, these cases are very few from the universe of Boolean functions up to 5-inputs.

VI. Conclusions

This work proposes a new way to verify P-matching. This approach uses a graph representation for Boolean functions. It divides the universe of functions that have fast P-matching and the ones that are computation time consuming in this process. In the case of few tie in the searching of irredundant nodes in the graph representation, this runs in linear time. In the case of many ties, it follows the theoretical complexity of the problem, being super-exponential, but just for few cases in such universe.

References

[1] A. Mishchenko; S. Chatterjee; R. Brayton; W. Wang and T. Kam. "Technology Mapping with Boolean Matching, Supergates and Choices," ERL Technical Report, EECS Dept., UC Berkeley, Mar 2005.

[2] T. Sasao and J. T Butler, "Progress in Applications of Boolean Functions," Synthesis Lectures on Digital Circuits and Systems, vol. 4, no. 1, 2009, pp. 1-153.

[3] U. Hinsberger and R. Kolla, "Boolean matching for large libraries," In Proc. Design Automation Conference (DAC), Jun. 1998, pp. 206–211.

[4] Minato,Shin-ichi. "Binary decision diagramns and applications for VLSI CAD". Boston:Kluwer Academic, c1996. 141p.

[5] De Micheli,Giovanni. "Sysntehsis and Optimizations of digital circuits". McGraw-Hill series in electrical and computer engineering. Eletronics and VLSI design, 1994.

[6] D. Debnath and T. Sasao, "Efficient computation of canonical form for Boolean matching in large libraries," In Proc. Asia and South Pacific Design Automation Conference (ASP-DAC), 2004, pp. 591-596.