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*— This paper presents a new form of representing Boolean functions through a bipartite graph. This structure is favorable for the calculation of equivalence P-matching. In logic synthesis, the technology mapping process can be a very time consuming task when fitting cells into a target library. The proposed method turns a Boolean function in a graph representation and apply reduction rules in it graph to verify functional equivalence. A fast algorithm is provided using this structure and experimental results show that this method runs in linear time in most cases.**

***Keywords*— Boolean matching, graph isomorphism, P-matching, digital circuit, Boolean function.**

I. INTRODUCTION

The standard cell flow is still playing a major role among the current IC design methodology. This flow is divided in several steps, in which the synthesis and technology mapping step use P-matching to find equivalent cells in library to map part of the functionality of the circuit [1].

The problem of determining when two Boolean functions are equivalent under the permutation of it variables is named P-matching [2], and it is used in searching for equivalent components on a targeted library. As a result, this step has to be as fast as possible. Several methods are proposed to solve it, but they are limited on growing its data structures [3-4].

This paper proposes a graph-based algorithm to solve the P-matching. The idea is generate a graph minterm-variable starting from a Boolean function. With this graph scheme, we represent the functions involved in P-matching in it graph form and use rules to reduce it. With this reduction occurring in the same time in both graphs, a structural check can be performed in its topology. If the two graph topology is equivalent [5], the functions represented by graphs minterm-variable are P-equivalent.

This reduction is useful because in several cases, the matching is done very fast. In other cases, the computation time can be more than exponential, but it is a property of any NP-compete problem.

The structure of this paper is as follow: Section II presents a technical background with concepts to understand this approach. Section III presents the proposed method: how to generate a graph minterm-variable from Boolean function, how to reduce this graph in comparison to others, and some examples. Section IV presents the results in generation of permutation classes and complexity analysis. Section V presents the conclusion of this work.

II. TECHNICAL BACKGROUND

A. Graph Isomorphism

If two graphs, G_1, 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 $f(v)$.

For instance, the bijection function illustrated in Fig. 1 is:

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

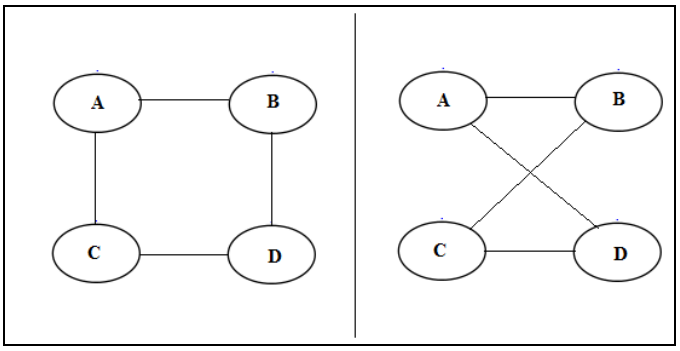

Figure 1 - Example of two isomorphic graphs.

B. P-Matching

The operation over two functions, f_1 and f_2, that determines if there is a permutation in variables in f_1 such that this function turns into f_2. [2].

If a P-matching exists between f_1 and f_2, the function f_1 is called P-equivalent with f_2.

C. Graph Minterm-Variable

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

Support of a Boolean function with n variables is a set $A = \{a1, a2, a3, \ldots an\}$ 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 follow:

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

$E = \{e(a_i, (m_1, m_2, \ldots, m_n)_k) / a_i$ belongs to A and $(m_1, m_2, \ldots, m_n)_k$ belong to B^n and $f((m_1, m_2, \ldots, m_n)_k) = 1$ and $m_i = 1\}$, where $i \leqslant n$ and $k \leqslant 2n$ and $e(i,j)$ is an edge between nodes i and j.

This graph is named graph minterm-variable [5].

III. Proposed Method

The proposed P-matching uses a graph minterm-variable structure to represent a Boolean function. The nodes of this graph can be divided in two subsets:

A. Graph Tentacle

Every node with degree equal to one and all reachable nodes that have degree equal to two is a node in a graph tentacle. Fig. 2 illustrate a graph with an tentacle in its nodes with white color. The black are belongs to other classification.

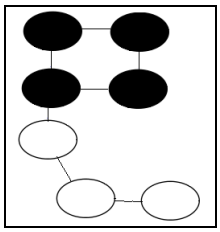

Figure 2 - Example of graph with a tentacle.

B. Graph Cycle

Every node that does not have in tentacle is on a cycle. These nodes are represented in nodes in black in Fig. 2.

C. Graph Code

Every node in graph has a stack of integer numbers representing the code that represents the information of it neighbour. Here, we have divided the set of nodes in two subsets of type of code:

- Nodes with degree lower than 2:
  When a node is removed simply, and sum the removed node code with the top of stack of adjcacents nodes.
- Nodes with degree greater than 2:
  When a node is removed, insert its code on the top of stack of it adjacent node, growing in one level the stack size.

D. Equivalence check

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

1) Generate the graph associated with 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 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 graph-code of node removed to adjacent node.
6) Check if removed nodes are equivalent in code and degree.
7) Repeat until the graph is empty or no equivalent deletion is encountered.

If both graphs are empty in final step, the functions are P-equivalent. If not, the functions are no P-equivalent.

Example 1: The function 0x8F and 0xD5, illustrated in Fig. 3, are P-equivalent. Initially, the code associated with each node is '$', representing empty code. In the next step is set the code 1 to all nodes with degree 1, as illustrated in Fig. 4. These nodes are represented with dotted edge.

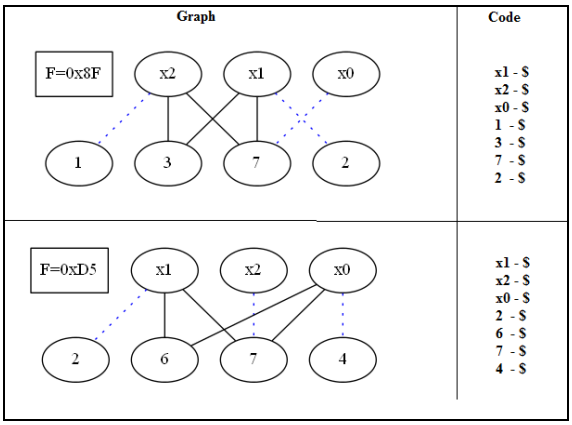

Figure 3 - Example of graph associated with function.

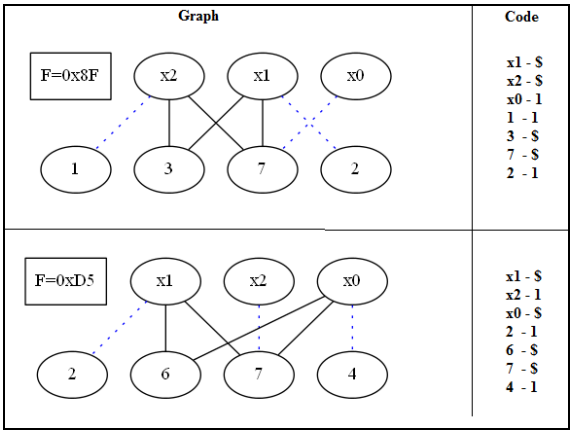

Figure 4 - Initial code completion.

To reduce these graphs, we remove all nodes that have the code equals to 1, and then its code is propagated to adjacent nodes, as seen in Fig. 5. This first reduction removes only

variables nodes in both graphs, and test if these deletions are in the same number.

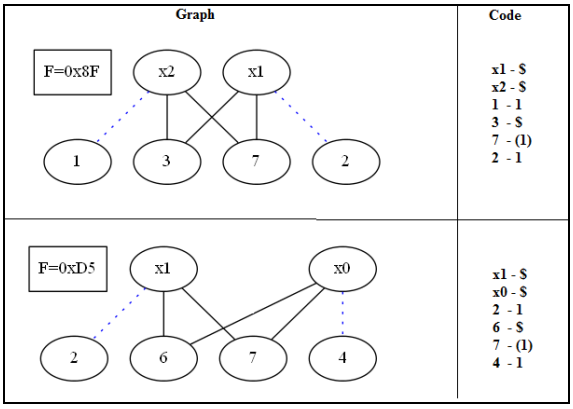

Figure 5 - First graph reduction.

The next reduction, illustrated in Fig. 6, removes the minterms nodes and check if the deletions are in the same number.

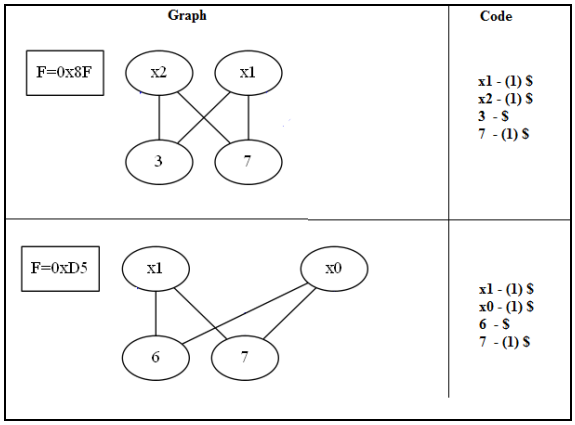

Figure 6 - Second graph reduction.

Here, we generate a cycle and, therefore, we need cut this finding an irredundant node and then remove it from both graphs. (Example 2, described after, represents a case where no irredundant node exists).

In the case of the graph in Fig. 6, the nodes 3 and 6 are unique in its code in graph 0x8F and 0xD5, respectively. Thus, these nodes are removed and its code propagated.

We repeat the algorithm in this point, always testing if there are tentacles in graph, as illustrated in Fig. 7. In this case, the entire graph is a single tentacle. The repetition removes the nodes of variable, and check the number of deletions.

In Fig. 8, the graph was reduced and last node has the same code associated. There are match in both functions because the deletion of last node leaves an empty graph.

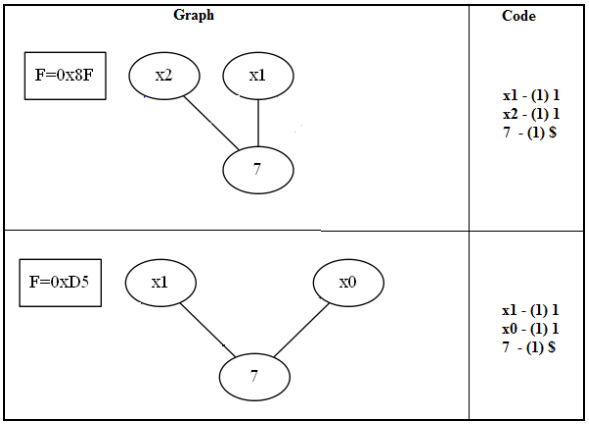

Figure 7 - Reduction of cycle.

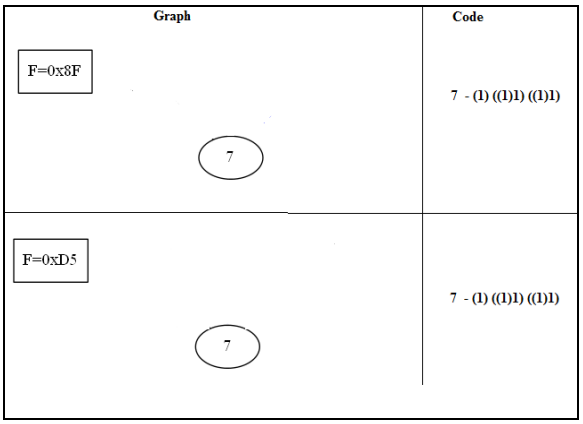

Figure 8 - Reduction of tentacles.

Example 2: The function 0x68 and 0x68, shown in Fig. 9, are P-equivalent because is the same function.

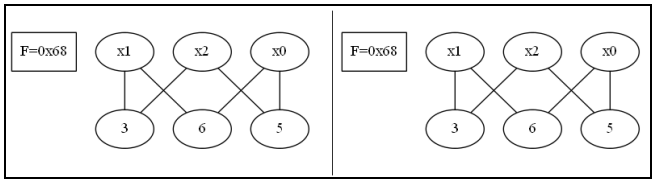

Figure 9 - Graph reduction lower case.

If every node ties in its code, then we get all tied nodes in both graph, and put them on a set T_1 associated with one graph and T_2 associated with other graph.

Then, extract an element t of T_1 and for all element e in T_2 creates a pair (t,e), removing this pair in its graph, and repeat the algorithm with the rest of graph. If some pair returns true in its reduction, there are a matching in graph.

In this example, we generate the set of tied nodes G_1 = {x1,x2,x0,3,6,5} and G_2 = {x1,x2,x0,3,6,5}. And we try to match some node in G_1 to some node in G_2. Maintaining the node x_1, the possibilities are (x_1,x_1), (x_1,x_2), (x_1,x_0).

Since variable node only match with variable node, x_1 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, others pairs do not need to be tested.

Example 3: Considering the function 0x81 with 0x86, illustrated in Fig. 10. Such functions do not match because their graphs are not isomorphic.

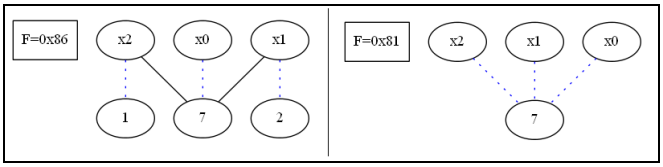

Figure 10 - Graph reduction best case.

These functions do not have cycle. It is the best case of match because the tentacles are removed very fast in linear time in number of nodes. Just to notice, the example 1 has a unique node when a tie occurs and then runs in linear time too.

IV. Experimental Results

The algorithm has been validated with the generation of P-class with 1, 2, 3 and 4 input functions. This step validates the correctness of algorithm.

In the way to reach more than 4-inputs functions, a test with 5-inputs NPN class was made and every function was tested with all other functions in the same class. Although 5-inputs NPN class has 616,125 functions, this test spent around two days in computation time and no error was encountered. The results are show in Table I. The #n represents all functions with 'n' variables. Table I demonstrates that only few functions in the entire set are computation time consuming during the evaluation

TABLE. 1 - GRAPH LOWER CASE OCCURRENCES.

	#1	#2	#3	#4	5 NPN
Total	4	16	256	65,536	616,125
Functions with tie	0	0	44	9,376	42,936

Random tests were performed considering up to 19-input functions. A study of which function represents a bottleneck for this approach was also made.

In order to explain a formal proof of this work the time complexity of this approach is the following. Assuming a graph $G(V,E)$, we know 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 this set as T, representing nodes in tentacle and C to nodes in cycles, since that C is disjoint of T. The graph G can have several tentacles and cycles. Every tentacle in T is reduced in time proportional to it number of nodes, in lower case it can be the entire V. Then, in this case, we have $O(|V|)$ complexity when $|V|$ represents the cardinality of set V.

In the case of cycles, the lower case 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, we resolve 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 $O(|k_1|*|k_2|...*|k_n|)$, where $\sum k_i = |V|$, $n \rightarrow 0$. It says that this approach runs in super-exponential time in lower case, as every NP-complete problem like P-matching. However, as show in Table I, these cases are very few from the universe of Boolean functions.

V. Conclusions

This work proposes a new way to verify P-matching. This approach uses a graph representation for Boolean functions. This approach divides the universe of functions that have fast P-matching and the ones that do not have. In the case of few tie in search of irredundant nodes in 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.

Acknowledgment

Research funded by the Brazilian funding agencies CNPq and FAPERGS, under grant 11/2053-9 (Pronem), and by the European Community's Seventh Framework Programme under grant 248538-Synaptic.

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] 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.

[5] Silva, Anderson Santos da; Ribas, Renato; and Reis, Andre; "A graph-based approach for Boolean matching", In XXVII SIM- South Symposium on Microeletronics, 2012.