Boolean Function Matching using Walsh Spectral Decision Diagrams

Jason Moore, Kenneth Fazel, Mitchell A. Thornton
Southern Methodist University
{jmoore,kfazel,mitch}@engr.smu.edu

D. Michael Miller
University of Victoria
mmiller@cs.uvic.ca

ABSTRACT

This paper investigates two approaches for Boolean matching using NPN *equivalence matching. Luks' hypergraph method is implemented and compared to the Walsh Spectral Decision Diagram (SDD) method that we propose here. Both methods determine a canonical representation for each* NPN *equivalence class. The target functions are then transformed into a canonical representation and compared to the representative canonical forms for the* NPN *classes. This paper presents the implementation and results of the spectral method in detail. It is shown that the spectral method compares favorably to Luks' method and is better in terms of computational requirements for large functions.*

1. INTRODUCTION

The standard-cell ASIC design process requires developing templates or cells that define basic logic elements and then using those cells to develop larger circuits. Technology mapping, or 'cell-library binding', is the process of selecting and connecting in proper sequence the appropriate cells from a "standard cell" library developed by an integrated circuit fabrication company. Technology mapping can be considered as comprising two tasks; Boolean matching and selection. *Boolean Matching* is determining whether or not two Boolean functions are equivalent with respect to some basic transformations. The result of Boolean matching is an equivalence class from the standard cell library that can implement the given function. Since each cell in the same class can implement equivalent functions, the "best" cell is selected from this equivalence class to maximize some objective function such as total area, delay or power consumption minimization. This latter process is called *selection*.

Equivalence classes can be defined as *P* (Permutation), *NP* (Negation-Permutation), or *NPN* (Negation-Permutation-Negation) equivalent [14]. Two Functions f and g, defined over the same variable set X, are *P* equivalent if there exists a permutation operator $\mathbf{P}$ such that $f(X) = g(\mathbf{P}X)$ where $\mathbf{P}$ represents an elementary permutation matrix. The naïve approach to this problem is to try all possible permutations. The complexity of this approach is $O(n!)$. The following example will help to explain *P* equivalence. Consider two functions $f = a_1a_2 + a_1a_3$ and $g = a_2a_1 + a_2a_3$. f and g are *P* equivalent if we switch the position of a_1 and a_2, that is $f(a_1, a_2, a_3) = g(a_2, a_1, a_3)$.

Two Boolean functions are said to be *NPN* equivalent if there exists a permutation operator $\mathbf{P}$, with two complementation operators $\mathbf{N}_o$ and $\mathbf{N}_i$, such that $f(x) = \mathbf{N}_o g(\mathbf{P}\mathbf{N}_i x)$. The complementation operators specify the possible negation of some of their arguments. *NPN* equivalence can be thought of as an extension to *P* equivalence. For each input permutation, the inputs must also be considered in negated form and can require $O(2^n)$ time. Thus, the complexity of *NPN* equivalence checking is $O(n!2^n)$. Through the application of DeMorgan's theorem, it is easy to see that the negation of the inputs of an OR-gate results in the functionality of a NAND gate. Therefore, two-input OR and NAND functions are *NPN* equivalent.

In this paper, we first discuss the implementation of a hypergraph isomorphism checking method proposed by Luks in [10]. Then we propose a method using a Walsh Spectral Decision Diagram (SDD) method, to overcome the weakness of Luks' method. Both of these methods find a canonical representation for each *NPN* equivalence class. The target functions are then transformed into a canonical representation and compared to the representative canonical forms for the *NPN* classes. We note that the method of Luks is interesting because it transforms the Boolean matching problem to one of hypergraph isomorphism and that the use of the Walsh spectrum for Boolean matching was first reported in [5]. This work leads to an interesting comparison in these two techniques and also yields an approach for Boolean matching that handles very large functions in a reasonable amount of computation time. The primary difference between the Walsh spectral method described here and that in [5] is that the use of efficient data structures known as SDDs are employed that reduce the average-case memory requirements to linear complexity in terms of dependent function variables as opposed to the exponential requirements needed in the method reported in [5].

2. PRELIMINARIES AND RELATED WORK

For effective technology mapping, equivalence classes must be selected very quickly and must contain all equivalent functions. In general, Boolean matching is intractable. Many algorithms have been proposed for Boolean matching including [1–6].

In [1], Ciric and Sechen propose a solution that constructs a table for each Boolean function and then permute a table to the lowest cost according to their cost function. Their approach does not consider input or output negation. The method only classifies the functions in *P* (permutation) classes. In [2], Correia and Reis present a method based on the size of a Reduced Ordered Binary Decision Diagram (BDD). The BDD is negated and permuted until the BDD reaches a small size. Then the function is classified by the size of the BDD. Hinsberger and Kolla [3] consider each possible permutation and negation as a block in a tree. They are able to reduce the search space because they are only interested in finding the maximum leaf in the type of tree structure that they define. The maximum leaf is used as the unique representation of the Boolean function. They also suggest storing the cells in the cell library according to their equivalence class. The approach in [3] is also referred to as a cut-based Boolean mapping technique.

Recently, authors in [6] propose a simplified version of the cut-based Boolean mapping technique. Instead of storing one canonical

representation of a library gate, all functions obtained by permuting the inputs to library gates are precomputed and stored. In this way, only *N* equivalence mapping is necessary instead of *NPN* equivalence mapping. Since *N* equivalence is much easier to compute than *NPN* equivalence, Boolean matching is faster than the original cut-based method. However, as the authors also indicate in their paper, this method is not scalable. Indeed, this method would not work if the number of inputs for functions is larger than 10 variables.

In [4], a spectral method is proposed for the Boolean matching problem. Chang and Falkowski arrange the Reed-Muller expansion of a Boolean Function in ascending order of their magnitudes in terms of weights of their literal vectors.

In [5], Miller shows how Boolean matching can be accomplished using the Walsh spectrum. The Boolean functions are represented by their Walsh spectra which is arranged such that the largest values (in magnitude) are as early as possible and negative values are as late as possible in the spectral vector. Although this method requires exponentially large memory space, this observation is key to the results presented here. The use of decision diagram based methods for the computation of the Walsh spectrum allows this approach to become practical.

The algorithm proposed in this paper is similar to the algorithm in [5]; however, in this work we use decision diagrams for spectral computations as is described in [8,13] and we exploit the special conditions of symmetry and parity functions that allow for dramatic improvement in terms of circuit size and cases that were "hard" in the previous work of [5]. The spectral computation algorithms used in this work are based on graph algorithms that were first presented in [13].

The *Walsh spectrum* is a Fourier representation of a Boolean function expanded about the orthogonal Walsh basis functions. The Walsh spectrum can be defined using linear transformations. As is shown in [7], the Walsh matrix w_n can be calculated by applying the Kronecker product $n-1$ times starting with the matrix w_1. This can be expressed as $w_n = \bigotimes_{i=1}^{n-1} w_i$, where

$$w_1 \equiv \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad w_n \equiv \begin{bmatrix} w_{n-1} & w_{n-1} \\ w_{n-1} & -w_{n-1} \end{bmatrix}$$

The Boolean function $f(X)$ can be represented as a column vector F where the corresponding 0 and 1 range values of f are replaced with 1 and -1 respectively.

The Walsh spectrum can be efficiently computed using a "fast" transform which is extensively documented in [9]. We briefly review the work in [9] for the completeness of the paper. The Walsh transform can be computed using less time and less space through the use of BDDs [7] and the "butterfly diagram technique" commonly attributed to Cooley and Tukey. The resulting spectrum is also in a DD form that is referred to as a *Spectral Decision Diagram* (SDD). The basic idea of computing an SDD is to change the terminal nodes "0" and "1" in the BDD to "1" and "-1" respectively. The graph is traversed until a node whose children are either a terminal node or a Walsh node. Then, the predecessor node is transformed by making the appropriate changes to the children through the "butterfly transform" as is shown in Figure 1 (details are provided in [8,13].):

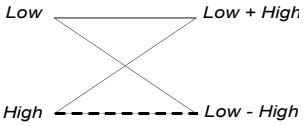

Fig. 1: Butterfly flowgraph of the fast transform [8]

The node is then marked as a Walsh node. The procedure stops when all the nodes are Walsh nodes. Figure 2 illustrates an example of this algorithm for a two input AND function.

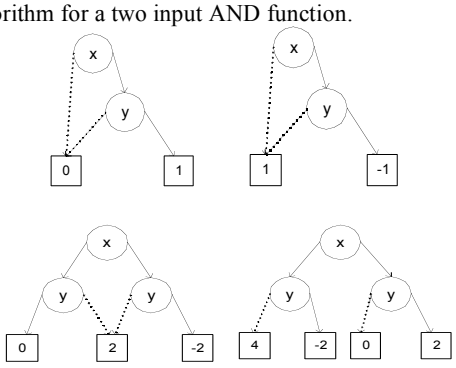

Figure 2: Calculating SDD using BDD

One advantage in using BDDs to compute SDDs is that the complexity is no longer directly dependent on the number of inputs, *n*, but instead dependant on the size of the BDD (the number of vertices *N*). *N* in the best case is linear in terms of *n*. And SDDs can share isomorphic subgraphs thus providing even more savings in space. The runtime to build a SDD from a BDD is $O(N^2)$ since each node in the BDD is visited once and the runtime for adding two BDDs is $O(N^2)$. The size complexity for the SDD is $O(N)$ as well.

3. MATCHING TECHNIQUES

3.1 LUKS' METHOD

In [10], Luks presents a method for Boolean matching using hypergraph isomorphism checking. Let f be an *n*-input logic function with variable set $\{\sigma_1, \sigma_2, ..., \sigma_n\}$. An associated hypergraph $H_f = \{\sum, E_f\}$ is defined with vertices $\sum = \{\sigma_{1,0}, \sigma_{1,1}, \sigma_{2,0}, \sigma_{2,1}, ..., \sigma_{n,0}, \sigma_{n,1}\}$ and an edge set $E_f = \left\{ \{\sigma_{i,a_i}\}_{1 \le i \le n} \middle| (a_1, ..., a_n) \in \{0,1\}^n, f(a_1, ..., a_n) = 1 \right\}$. The vertices of H_f consist of the original and complemented forms of σ_i ($\sigma_{i,0}, \sigma_{i,1}$). The hyperedges are the minterms of f. To check for *NPN* equivalence between two functions *f* and *g*, we check for hypergraph isomorphism between H_f and H_g and between H_f and $H_{g'}$ where, H_f, H_g, and $H_{g'}$ are hypergraphs for the functions f, g, and g' respectively (g' is negation of g). If either check shows isomorphism then *f* and *g* are *NPN* equivalent. One method for determining hypergraph isomorphism is to derive canonical forms of the induced bipartite graphs of H_f, H_g and $H_{g'}$ and then to compare appropriate bipartite graphs for equivalence matching.

The previous example of the NAND ($f = NAND(a,b)$) and the OR ($f = OR(a,b)$) are processed using Luks' hypergraph algorithm as follows. The list of minterms for each function is

$f = \{a'b', ab', a'b\}$ and $g = \{ab', a'b, ab\}$. The bipartite graphs for *f* and *g* are then constructed as shown in Figure 3. The two bipartite graphs are isomorphic which indicates that the 2-input NAND is *NP* (and hence *NPN*) equivalent to the 2-input OR function. The tools for isomorphic checking of two bipartite graphs can be found in [15].

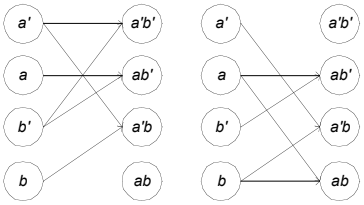

Figure 3: Bipartite graphs for *f* and *g*

3.2 WALSH SPECTRUM

Each of the coefficients of the Walsh spectrum for the function $f(X)$ measures the correlation between $f(X)$ and the exclusive-OR of some subset of X. The coefficients of the Walsh spectrum are denoted as W_α, where α identifies the variables involved in the corresponding exclusive-OR function. We also classify the coefficients based on the cardinality of the set α. For example, W_1 is the first order Walsh coefficient; W_{12} is the second order coefficient dependent on the 1st and 2nd inputs; and so on. All Walsh coefficients for a 4-input function can be represented as a vector $W = (W_0, W_1, W_2, W_3, W_4, W_{12}, W_{13}, W_{14}, W_{23}, W_{24}, W_{34}, W_{124}, W_{134}, W_{234}, W_{1234})$ If the absolute values of the 0^{th} and n^{th} order coefficients of two functions to be compared are not equal, then the two functions are not an *NPN* equivalence match [5]. Otherwise, it is possible that the two functions are a match. The following lemmas describe properties important to identifying *NPN* equivalence of two Boolean functions.

Lemma 1: If the zero[th] ordered Walsh coefficient, W_0, is negated, then all other coefficients W_α must also be negated.

Lemma 2: If a first order coefficient W_i is negated, all higher order coefficients W_α where $i \in \alpha$ must also be negated.

Lemma 3: If first order coefficients, W_i and W_j, are interchanged then all coefficients $W_{i,\alpha}$ and $W_{j,\alpha}$ must also be interchanged.

If the initial check for the 0^{th} and n^{th} coefficients indicate that the two functions cannot be an *NPN* match, then the CPU runtime is $O(N^2 + n)$ where n is the number of inputs and N is the size of the BDD. However, if the initial check does not eliminate the possibility that the two functions can be an *NPN* match, we can transform the function into a canonical form via the "**canonicalize**" function where we compare the two SDDs. If their canonical forms are equivalent, two functions are a match.

In the **canonicalize** routine we rearrange the Walsh spectral values such that the coefficients are decreasingly ordered (in magnitude) as was done in [5]. If a coefficient is negative, we negate the coefficient and make it positive and move it to the front based on the Lemmas. Although this definition of a canonical representation is arbitrary, it provides a technique to compare two functions. Regarding negation, all first-order coefficients can be made to be positive. For the second or higher order coefficients, any negative coefficient can be negated, if and only if, the needed lower level coefficients are 0. For example, W_{13} can be negated if W_1 or W_3 is 0. And also if W_1 is equal to 0 and W_3 is not equal to 0, then W_{12} must also equal 0. The worst-case time for this algorithm is $O(n^3 2^n)$. The worst case is when most of the first order coefficients are 0 but not all of them.

We also propose two techniques to speed up the procedure. The first one utilizes the property of symmetry. The algorithm can achieve a quadratic runtime improvement compared to past techniques when the function is completely symmetric and one of the following conditions holds: either none of the first order coefficients are 0, or, when they are a parity function. The parity function is detected in our algorithm by the property that all the Chow parameters ($W_1, W_2, ..., W_n$) are 0 and the n^{th} order coefficient must equal 2^n [11]. Figure 4 shows the SDD for a 4 input parity function.

Another speed up is incorporated by checking the 1^{st} order coefficients. If all the 1^{st} order coefficients of a function are equal and none of them are 0, the procedure can return since the other function needs to have the same 1^{st} order coefficients for equivalence matching in this situation.

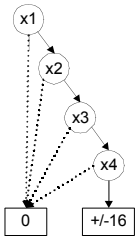

Figure 4: SDD for a 4 input parity function

Figures 6 through 8 demonstrate this algorithm using the example of 2-input NAND and OR functions. Since the CUDD package [12] is used to implement this algorithm, a unique table is employed to store the BDDs and comparing the two BDDs in Figure 6 takes constant time.

4. RESULTS

Luks' method and the Walsh spectrum method are implemented and run on a benchmark set from MCNC. From the bit vectors in the benchmark set, other bit vectors are generated. Set one contains five bit vectors of varying Hamming distance (h_d): 20%, 40%, 60%, 80%, and 100%, from the original bit vector. The bit vectors in set 1 were calculated by changing $\lfloor 2^n * h_d \rfloor$ values. Set 2 contains five bit vectors that are *NPN* equivalent to the original bit vector. All of the results were run on a Linux machine with a 2 GHz Pentium4 processor and 2 Gigabytes of RAM.

Table 1 shows that the Walsh Spectral method for Boolean matching becomes faster than Luks' method when the number of inputs is larger than 10. For benchmarks where the number of inputs is larger than 16, Luks' method cannot finish the matching problem due to running out of memory. Walsh spectral methods can be applied to the matching problem within seconds when using modern techniques based upon Walsh spectra and decision diagrams as discussed in this paper. One of the reasons that the Walsh method outperforms Luks' method for larger functions is that the Walsh

algorithm is dependant on the size of the SDD instead of the number of inputs. Even in a potentially worst-case scenario, the run time can be small if the SDD is within reasonable size.

5. CONCLUSIONS

This paper covers the topic of Boolean Matching in technology mapping. Two algorithms for *NPN* equivalence Boolean Matching were investigated in detail. Although in the worst-case these algorithms are exponential, for many benchmark circuits, the algorithms are polynomial. The first algorithm was Luks' hypergraph method in which Boolean functions are represented as bipartite graphs and then are subsequently transformed as DAGs where the isomorphism check on the bipartite graph is performed using traditional heuristics. This method implements the *NPN* equivalence checking problem as a hypergraph isomorphism problem. The second algorithm is the Walsh spectral matching method using SDDs, in which the Boolean function is represented in terms of the Walsh spectrum and then compared to a SDD for *NPN* equivalent checking.

Our results indicate that Luks' method is faster than the Walsh spectral method for small functions. However, Luks' method runs out of memory as the number of input variables increase. This leads us to conclude that the spectral method is superior for large functions and has applicability for the matching of large cells and for function identification problems that arise in logic synthesis and formal verification.

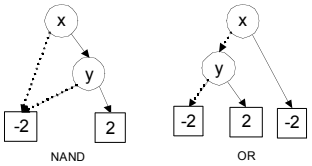

Figure 5: SDDs for 2 input NAND and OR

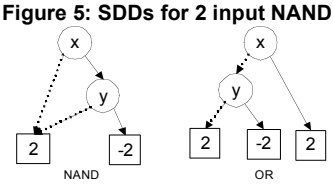

Figure 6: Negating outputs of NAND and OR

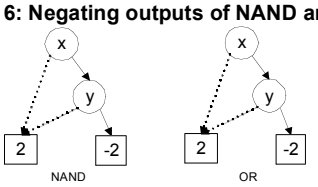

Figure 7: Negate Y input on the OR gate

6. REFERENCE

[1] J. Cric, C. Sechen, "*Efficient Canonical Form for Boolean Matching of Complex Functions in Large Libraries*" in IEEE Trans. on CAD, Vol. 22, May 2003, 535-544.

[2] V.P. Correia, A. Reis, "*Classifying n-Input Boolean Functions*", in Proc. IWS 2001, 2001.

[3] U. Hinsberger, R. Kolla, "*Boolean matching for large libraries*", in Proc. Design Automation Conference 98, Jun. 1998, pp.206-211.

[4] C. Chang, B. Falkowski, "*NPN classification using weight and literal vectors of Reed-Muller expansion*", Electronic Letters Vol. 35 No. 10, May 1999, pp. 798-799

[5] D. M. Miller, "*A spectral method for Boolean function matching*", in Proc. of *Design Automation and Test in Europe*, Mar. 1996, pp 602.

[6] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "*Reducing Structural Bias in Technology Mapping*," Proc. of IWLS, 2005

[7] M. Thornton, R. Drechsler, and D.M. Miller, *Spectral Techniques in VSLI CAD,* Boston: Kluwer Academic Publishers, July, 2001.

[8] M. Thornton, R. Drechsler, "*Spectral decision diagrams using graph transformations*", in Proc. of Design Automation and Test in Europe, Mar. 2001, pp. 713-717.

[9] M.A. Thornton, D.M. Miller, and R. Drechsler, "Transformations Amongst the Haar, Walsh, and Reed-Muller Spectral Domain." in *International Workshop on Applications of the Reed-Muller Expansion in Circuit Design (RMW),* August 10-11, 2001, 215-225

[10] E. M. Luks, "*Hypergraph Isomorphism and Structural Equivalence of Boolean Equations*", STOC 1999 Atlanta GA, 1999, pp. 652 - 658.

[11] M. Thornton and V.S.S Nair, "Parity Function Detection and Realization Using a Small Set of Spectral Coefficients", *IEEE/ACM International Workshop on Logic Synthesis (IWLS),* May 1995, pp. 8-39 - 8-47

[12] F. Somenzi, CUDD http://vlsi.colorado.edu/~fabio

[13] D. M. Miller. Graph algorithms for the manipulation of boolean functions and their spectra. In Congressus Numerantium, pp. 177–199, Winnipeg, Canada, 1987.

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

[15] http://cs.anu.edu.au/~bdm/nauty/

Table 1: Experimental Results

Benchmarks	Inputs	set 1		set 2	
		Walsh	Luks'	Walsh	Luks'
majority	5	1.67E-03	4.36E-05	5.00E-03	1.13E-04
xor5	5	<1E-6	7.49E-05	<1E-6	1.51E-04
con1	6	<1E-6	6.32E-05	1.67E-03	2.01E-04
5xp1	7	<1E-6	8.31E-05	3.33E-03	3.33E-03
9sym	9	1.67E-03	2.20E-02	1.47E-01	1.40E-01
Z9sym	9	3.33E-03	2.19E-02	1.50E-01	1.26E-01
dk17	10	2.00E-02	3.85E-04	9.17E-02	1.06E-03
sym10	10	3.33E-03	1.24E-01	6.67E-03	5.90E-01
xor10	10	1.67E-03	1.96E-01	3.33E-03	1.77E-01
t481	16	5.50E-01	mem out	1.95E+00	mem out
xor20	20	3.54E+00	mem out	2.35E+00	mem out
or50	50	mem out	mem out	1.50E-01	mem out