Boolean Function Spectrum Computation Using a Structural Representation

M. A. Thornton, V. S. S. Nair
Southern Methodist University
Dept. of Computer Science and Engineering
Dallas, Texas 75275
(214)768-2861
FAX:(214)768-3085
mitch@seas.smu.edu, nair@seas.smu.edu

Abstract

A method for computing the spectral coefficients of a Boolean function represented in a structural form is introduced. This result is important since spectral coefficients may be used in many areas of digital systems engineering including synthesis, verification, and fault detection. In the past, efficient methods for computing the spectrum of a Boolean function required the function to be represented as a binary decision diagram or a set of cubes that provide a complete cover. The methodology described here allows the computations to be performed directly using the topology of a logic diagram representation. By using the structural form of representation, the spectral coefficients of certain classes of Boolean functions are computed more efficiently than previous methods allowed.

1 Introduction

There are many problems in the field of digital systems engineering that may be solved elegantly using spectral methods. However there have not been efficient methods for computing the spectral coefficients of a Boolean function until recently. This paper introduces a methodology for computing a spectral coefficient directly from a structural representation of a logic function. This result is significant since past methods for efficient computation of the spectral coefficients required the logic function to be represented as a set of product terms covering the function or as an ordered binary decision diagram (OBDD).

The ability to compute the spectrum directly from a structural representation is important since recent efficient methods for spectrum computation rely upon binary decision diagrams (BDDs) as input. Unfortunately, it has been proven that some Boolean functions exist that may not be represented in BDD form unless the number of vertices is proportional to $2^{n/2}$, where n is the number of input variables in the function[1]. In these cases, the methods for the computation of the spectrum of a Boolean function fail to be efficient due to the large size of the BDD.

The interest in computing the spectrum of a combinational logic function in structural form arises because many logic synthesis systems describe the function in terms of a logic diagram, net-list, or PLA table after the minimization phase [2]. Following minimization, the technology mapping task of the system must then deal with a structural form of representation as input. Most technology mapping subsystems consist of a partitioning task followed by a matching or library binding phase. Although the library binding problem is co-NP complete, a significant amount of work has been done to streamline this aspect of the technology mapping portion of logic synthesis systems [3] [4] [5] [6] [7]. In particular, many efforts have utilized the properties of NPN-equivalence which classify circuits into disjoint sets where any one member may be used to provide the output of another member by possible inversion and permutation of some inputs, and perhaps, inverting the output. The theory of NPN-equivalence and its relation to certain spectral coefficients has been studied extensively [8] [9] [10]. Therefore, the ability to efficiently determine spectral coefficients from a structural representation of a circuit will allow spectral-based Boolean matching techniques to be easily included in this class of existing logic synthesis systems.

Another area of application of the use of logic diagrams to compute spectral coefficients is verification. This problem is the determination of whether or not two representations of a Boolean function are equivalent. The most common use of verification tools is to check that a specification of a digital logic circuit is equivalent to a proposed circuit. Typically, the specification is provided in the form of a functional description of a Boolean expression such as a binary decision diagram or behavioral hardware description language (HDL) and the proposed circuit is in the form of a net-list or schematic diagram. Verification methods form an integral part of modern day synthesis systems since they are used to determine "design correctness". It has been proven that the problem of determining if two logic specifications are exactly equivalent is a member of the set of problems whose solutions are NP-complete [11]. For this reason, the methodology of determining the solution for the verification problem is approached by two different means. "Formal verification" techniques guarantee the equivalence of two logic functions and therefore are costly to compute. The less costly methods employ heuristics or statistical methodologies for non-exhaustive verification solutions. It is the statistical methodologies that may be improved by using spectral coefficients. By determining that a spectral coefficient is equal when computed from two different representations of a Boolean function, the probability of their equivalence increases. Since the verification process typically involves at least one of the representations in structural form, the spectral calculation method presented here can greatly improve the efficiency of the overall statistical verification process.

Closely related to the synthesis and verification problems is the analysis of a circuit for fault detection. The theory provided in [10] provides a large amount results concerning the determination of the detection of various classes of faults based upon the spectral coefficients

of a Boolean function. These results provide motivation for the ability to compute spectral coefficients directly from a structural representation of a given Boolean function since fault detection analyses are normally carried out for existing circuitry.

The remainder of this paper is organized as follows. Section 2 will briefly review the properties of spectral coefficients that are relevant to the development of the new computation technique. Section 3 will provide the details of the computation of output probabilities of logic circuits. The new technique for computation of the spectral coefficients is presented in section 4. In Section 5 a discussion of the implementation of this method and the associated computational complexity is discussed. The last section will provide conclusions and future related areas of research.

2 Properties of Spectral Coefficients

Spectral techniques have generally not received much attention due to the large complexity associated with obtaining the spectral vector. However, several research groups have recently developed new techniques for the computation of spectral values that are much more efficient than the direct implementation of a vector-matrix product [12] [13] [14] [15] [16] [17] [18]. In particular, we have developed a methodology that allows a single coefficient to be computed using the ordered binary decision diagram (OBDD) [19] of the candidate circuit as input [12] [13].

In this section, the ideas used to develop the efficient spectral computation technique in [12] [13] are used to create a similar computational method using a structural representation of a Boolean function as input. This result is important since it will allow a spectral coefficient to be computed directly from a diagram of a circuit without resorting to forming a Boolean expression, the circuits' output vector, or, the OBDD corresponding to a circuit diagram. Also, a simplification is noted for the method proposed in [12] [13] that allows the spectral coefficients to be computed by reducing the calculations to one-half those required in these initial works. This simplification applies to the method using OBDDs given in [12] [13] as well as the new method described here that utilizes structural representations.

Before the method for computing the spectral coefficients is given, a brief review of the various forms of spectral coefficients will be described with some key concepts pointed out. The following notation and definitions will be used in the remainder of this paper:

- Small case variables such as x_0, x_1, etc. denote Boolean variables that have logic values of "1" or "0".

- Upper case variables such as X_0, X_1, etc. denote the probability that the corresponding lower case Boolean variables are equal to a logic "1" value. These quantities are real and exist in the interval $[0, 1]$.

- The operator symbol, "+", will refer to the Boolean OR function or the addition of real numbers depending upon the context of the equation in which it is used.
- The operator symbol, "$\cdot$", will refer to the Boolean AND operation. The absence of an operator between two adjacent values in a Boolean equation implies the presence of the "$\cdot$" operator.
- The operator symbol, "$\times$", will refer to the multiplication of two real values. The absence of an operator between two adjacent values in a real-valued equation implies the presence of the "$\times$" operator.
- The operator symbol, "$\oplus$", will refer to the Boolean XOR operation.
- The operator, "$\wp\{\}$", denotes the probability transform operator whose argument is a Boolean function. It will yield the probability that its argument is a logic "1". Unless otherwise noted, it is assumed that the input variables to the Boolean function are equally likely to be "1" or "0".
- $C_f[f_c]$ refers to the spectral coefficient of function f with respect to the constituent function, f_c, as originally defined by Chow.
- $S_f[f_c]$ refers to the spectral coefficient of function f with respect to the constituent function, f_c, as is more commonly defined for general spectral coefficients of a Boolean function.

There are varying definitions of the spectrum of a Boolean function. These different spectra are classified based upon the transformation matrices that are used for their computation. As discussed in [20], the transformation matrices may be viewed as a collection of "constituent" functions whose output vectors are used as row vectors in the transformation matrix. Most of the commonly used transformation matrices always include constituent functions that correspond to the constant function (either $f_{c0} = 0$, or, $f_{c0} = 1$) and functions that are equal to each primary input ($f_{ci} = x_i$ for $i = 1 \ldots n$). The particular subset of spectral coefficients that correspond to these $n + 1$ constituent functions are referred to as the Chow parameters [8]. As an example, the following transformation matrix could be used to compute the Chow parameters of a three-input function:

$$\begin{array}{r} 1 \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Notice that the constituent functions corresponding to each row vector of the transformation matrix are shown to the left of the matrix. As an example of the calculation of the Chow parameters, consider the circuit whose logic diagram is shown in Figure 1.

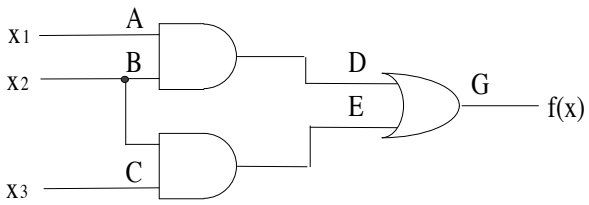

Figure 1: Logic Circuit Example for Chow Parameter Computation

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 3 \\ 2 \end{bmatrix} \tag{1}$$

The computation in Equation 1 is representative of the original definition of the Chow parameters as given in [8]. In this form, the Chow parameters yield information regarding the total number of minterms present in the circuits canonical sum-of-products (SOP) form of its logic equation (given by the coefficient corresponding to $f_{c0} = 1$), and, the number of minterms in which each primary input, x_i, appears in an uncomplemented form (given by each respective $f_{ci} = x_i$).

Since the Chow parameters were originally studied for the purposes of function classification, other researchers found that a modified version of these parameters was more convenient for the analysis of linear threshold functions [9]. The modified version is usually referred to as the "modified Chow parameters" [10] and they range in value of $[-2^{n-1}, 2^{n-1}]$. This form has the desirable feature that the coefficient corresponding to a particular x_i is always zero-valued if that x_i is a redundant input. However, the fact that a particular coefficient is computed to be 0 does not necessarily imply that the corresponding x_i is redundant. This property is formally proven in the work [9].

Many common transformation matrices are more general in that they contain only the real values of 1 and -1. This allows the -1-valued intermediate products to accumulate thus providing more information about the function being transformed. Typically, the logic "1" values are replaced with -1 and the logic "0" values are replaced with +1. For this reason,

it is often the case in the literature that the Chow parameters are computed in this more general way. As an example, the Chow parameters computed in Equation 1 would become:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ -2 \\ 2 \end{bmatrix} \tag{2}$$

It is useful to derive the algebraic equations that transform the Chow parameters from their original definition to the form in Equation 2. Note that the 0^{th}-ordered spectral coefficient in Equation 1 corresponds to the constituent function, $f_{c0} = 1$, while the same spectral coefficient in Equation 2 corresponds to $f_{c0} = 0$. In order to define algebraic relationships between the form of the Chow parameters in Equation 1 and those in Equation 2 we will denote them as $\{C_f[f_{c0}], C_f[f_{c1}], C_f[f_{c2}], \ldots C_f[f_{cn}]\}$ and $\{S_f[f_{c0}], S_f[f_{c1}], S_f[f_{c2}], \ldots S_f[f_{cn}]\}$ respectively. The relationships are given as:

$$S_f[f_{c0}] = 2^n - 2C_f[f_{c0}] \tag{3}$$

$$S_f[f_{ci}] = 4C_f[f_{ci}] - 2C_f[f_{c0}] \tag{4}$$

In fact, Equations 3 and 4 hold for more complex constituent functions as well. Therefore any arbitrary spectral coefficient may be computed using Equation 4 by replacing f_{ci}, with any general constituent function, f_c, that corresponds to a higher ordered spectral coefficient. For example, the set of 2^n spectral coefficients that are formulated using all possible non-inverted, single-variable input XOR functions and the Chow parameters are commonly known as the Walsh spectrum coefficients. By defining a single coefficient in terms of its constituent function, f_c, any generalized spectral coefficient may be computed using the technique described in this paper.

By carefully examining the way in which the Chow parameters are computed using the original definition as illustrated in Equation 1, it is seen that their value depends upon the number of times the constituent function and the function to be transformed are simultaneously equal to a logic "1" value. This quantity is denoted as N_{m1}. Further, if the percentage of times that both functions simultaneously equal logic "1" (denoted by p_{m1}) were known, then $N_{m1} = 2^n \times p_{m1}$.

3 Output Probabilities of Boolean Circuits

The main idea behind the spectral coefficient calculation scheme is the relationship between the output probability expression (OPE) of a circuit and a spectral coefficient. OPEs were first introduced in [21] where they were used to perform analysis for the effectiveness of random testing schemes. In [12], we derived the relationship between a circuit output probability and a spectral coefficient by using the values p_{m1} and N_{m1} discussed in the preceding section.

The OPE of a combinational logic circuit is an algebraic expression that expresses the probability that the circuit output is a logic "1" given the probabilities that the input variables have the value of logic "1". It is possible to compute the OPE for a given circuit by transforming its Boolean equation representation or by calculating the OPE from a schematic diagram representation [21]. The rules for transforming a Boolean equation to its equivalent OPE are given in Table 1. In this table, the rules are given for 2-input logic gates only, but the expressions for functions with more than 2-inputs or with different gate types may be easily derived.

Table 1: Rules for Transforming Boolean Operations to Probability Expressions

FUNCTION	BOOLEAN EXPRESSION	PROBABILITY EXPRESSION
Inversion	$\overline{x}_1$	$1 - X_1$
OR	$x_1 + x_2$	$X_1 + X_2 - (X_1 \times X_2)$
XOR	$x_1 \oplus x_2$	$X_1 + X_2 - 2(X_1 \times X_2)$
AND	$x_1 \cdot x_2$	$X_1 \times X_2$
Idempotence Property	$x_1 \cdot x_1$	X_1

An algorithm for the computation of the OPE of a Boolean function using its logic diagram representation as input is also given in [21]. In this OPE calculation technique, each primary input, each internal interconnection, and the output is assigned a unique variable name. Using the rules in Table 1, each internal node is expressed as a function of the primary inputs. This step is performed through subsequent substitutions until an expression is derived for the output variable in terms of the primary input variables thus forming the OPE. As an example, consider the logic diagram illustrated in Figure 2 that is a realization of the Boolean equation:

$$f(x) = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2x_3x_4 + \overline{x}_1\overline{x}_2x_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3\overline{x}_4 + x_1\overline{x_2}x_3\overline{x}_4 \quad (5)$$

Using the variables assigned to each interconnection as shown in Figure 2 and the rules in Table 1, the OPE is derived as follows.

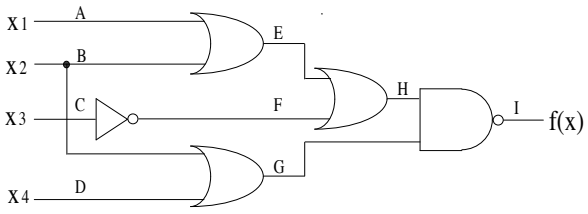

Figure 2: Logic Circuit Example for OPE Computation

First, the rule for the Inversion operator is applied:

$$F = 1 - C \tag{6}$$

Next, the rule for the Inclusive-OR operator is used:

$$E = A + B - AB \tag{7}$$

$$G = B + D - BD \tag{8}$$

$$H = E + F - EF \tag{9}$$

Simplifying the equation for H by substituting Equations 6 7 into Equation 9:

$$H = 1 - C + AC + BC - ABC \tag{10}$$

Finally, the rules for the AND and Inversion operations are used:

$$I = 1 - HG \tag{11}$$

This equation is simplified and the corresponding input probability variables are substituted resulting in the OPE:

$$I = 1 - X_2 - X_4 + X_2X_4 + X_3X_4 - X_1X_3X_4 - X_2X_3X_4 + X_1X_2X_3X_4 \tag{12}$$

Once the OPE has been computed, the probability that the output is equivalent to a logic "1" value is expressed as a function of the probabilities that the primary inputs are at a logic "1" value. For a fully specified Boolean function each primary input will be at logic "1" precisely 2^{n-1} times, hence the overall percentage of the time the logic function is equivalent to a logic "1" may be obtained by substituting 0.5 for all primary input probabilities.

4 Relation of Output Probabilities and Spectral Coefficients

From our earlier work [12], we have proven that the relationship between the OPE and a spectral coefficient is given by:

$$S_f[f_c(x)] = 2^n[1 + 2(\wp\{f \cdot f_c\} - \wp\{f + f_c\})] \tag{13}$$

Where f is the function whose spectral coefficient is desired, f_c is the constituent function used to compute the particular spectral coefficient, and, $\wp\{\}$ denotes the operator that transforms a Boolean expression to its equivalent OPE and then evaluates the OPE for all $X_i = 0.5$. Thus, the only requirement for computing the spectral coefficient is to form equivalent circuit diagrams for $f \cdot f_c$ and $f + f_c$.

However, this method for computing the spectral coefficients can be simplified by using the relationships given in Equations 3 and 4. The simplification arises because the values, $C[f_{c0}]$ and $C[f_{ci}]$ may be computed using only p_{m1}, therefore it is not necessary to form a circuit diagram (or OBDD) for the function $f + f_c$ (the percentage of times f and f_c simultaneously equal logic "0", p_{m0} is not needed). This result requires only the formation of a single equivalent circuit diagram (or OBDD) to compute a spectral coefficient. The key idea is to always compute the 0^{th} coefficient first since it is needed to compute the remaining values. Since the coefficients, $C[f_{c0}]$ and $C[f_{ci}]$ may be formulated in terms of p_{m1}, the use of Equations 3 and 4 allow $S[f_{c0}]$ and $S[f_{ci}]$ to be formulated in terms of p_{m1} as follows:

$$S[f_{c0}] = 2^n(1 - 2p_{m1}) \tag{14}$$

$$S[f_{ci}] = 2^n(4p_{m1} - 1) + S[f_{c0}] \tag{15}$$

Equations 14 and 15 show that it is only necessary to compute p_{m1} for each constituent function in order to determine the corresponding spectral coefficient. These equations are general in that any f_c may be used (hence any desired spectral coefficient may be obtained). The following example illustrates how a spectral coefficient may be computed using logic diagrams. If OBDDs are to be used, the reader is referred to reference [12] for a similar example.

In order to formulate p_{m1} for a corresponding constituent function, f_c, a theoretical logic circuit is formed by "ANDing" the constituent function logic circuit output with that of the circuit whose spectral coefficient is desired. This resulting "composite" logic diagram will yield a logic output of "1" only when both the original and constituent functions simultaneously output a logic "1". The determination of p_{m1} is obtained by computing the OPE of the composite circuit and then setting each variable equal to 0.5. It is not necessary to compute

the entire OPE for successive spectral coefficients of the same original circuit. Rather the OPE of the original circuit may be computed initially and stored. Then all that is required is the computation of the OPE of each constituent function and the two OPEs are combined using "AND-operation" rule given in Table 1. If the constituent functions are simple then the corresponding computation of each spectral coefficient is efficient.

As an example, consider the function in Figure 2 whose OPE was computed and given in Equation 12. First, the 0^{th}-ordered coefficient, $C[f_{c0}]$, is computed. By definition the corresponding constituent function is $f_{c0} = 1$. This means the composite circuit is formed by ANDing $f_{c0} = 1$ with the function to be transformed. Since $f \cdot 1 = f$, the value of p_{m1} in this case is obtained by simply evaluating the OPE of the original circuit with each input probability set equal to 0.5. The result is:

$$p_{m1} = 1 - 0.5 - 0.5 + 0.5^2 + 0.5^2 - 0.5^3 - 0.5^3 + o.5^4 = 0.3125 \tag{16}$$

Using Equation 14, the 0^{th}-ordered spectral coefficient is computed as:

$$S_f[f_{c0}] = 2^4[1 - 2(0.3125)] = 6 \tag{17}$$

To complete the example, the Chow parameter corresponding to $f_c = x_3$, will be computed. First the composite OPE will be formulated by ANDing the OPE for f_c (denoted by OPE_c) with that of the original function, f. This composite OPE (denoted by OPE_{comp}) is given by multiplying Equation 12 by $OPE_c = X_3$ and obeying the idempotence rule in Table 1. The result is:

$$OPE_{comp} = X_3 - X_2X_3 - X_1X_3X_4 + X_1X_2X_3X_4 \tag{18}$$

To determine the p_{m1} value associated with the constituent function, $f_c = x_3$, the value of 0.5 is used for all input probabilities in Equation 18. This substitution yields the value, $p_{m1} = 0.1875$. Using this quantity in Equation 15 yields the spectral coefficient:

$$S_f[f_{c3}] = 2^4[4(0.1825) - 1] + 6 = 2 \tag{19}$$

This example has demonstrated how spectral coefficients may be computed directly from their logic diagram description.

5 Implementation and Complexity

A directed acyclic graph (DAG) representing the topology of the circuit is used as input in this implementation. In the DAG, there are one or more edges entering each node in the graph that represent logic gate inputs and a singular directed edge leaving the node representing the logic gate output. Each node contains an identification field that indicates

the type of logic gate it represents. Before any spectral coefficients can be computed, it is necessary to compute the OPE of the logic circuit represented by the DAG. The OPE expression may be represented as a sum of real-valued product terms. In the worst case, there can be 2^n product terms since all possible combinations of the input variables may be used to formulate a single term. Fortunately, due to the idempotence rule in Table 1, there are no variables present in the OPE with a power greater than unity. In addition, most logic circuits have OPEs with significantly fewer product terms than the maximum possible number of 2^n. The number of product terms in an OPE for a particular logic function is similar to the number of minterms in a SOP form of a logic circuit, although there is no readily apparent relation between the two for a given circuit.

The most general form of the OPE of an arbitrary circuit is given in Equation 20.

$$\begin{aligned} OPE \quad &= a_0 + a_1X_1 + a_2X_2 + \cdots + a_nX_n \\ &+a_{12}X_1X_2 + a_{13}X_1X_3 + \cdots + a_{1n}X_1X_n \\ &+a_{23}X_2X_3 + a_{24}X_2X_4 + \cdots + a_{2n}X_2X_n \\ &\vdots \\ &+a_{123\ldots n}X_1X_2\cdots X_n \end{aligned} \tag{20}$$

From Equation 20, it is apparent that the maximum possible number of non-zero terms, T_{max}, is given by:

$$T_{max} = \sum_{i=0}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) = 2^n \tag{21}$$

In this implementation, the OPE is stored in the form of a linked list with each node in the list containing a "product term identifier" (PTI) indicating which input variable combinations are present, a non-zero coefficient, a_i, and a pointer to the next term in the expression. By storing only those terms with non-zero coefficients, 2^n memory locations are only used in the worst case. The PTI is implemented as a positive integer whose values have the range $[1, 2^{n+1} - 1]$. Using this range ensures that each PTI is unique since in binary form the PTI has bit $b_i = 1$ only if X_i is present in the product.

Once the OPE is formulated for the circuit whose spectral coefficients are desired (denoted as OPE_f), each coefficient in OPE_f is computed by multiplying the stored OPE of the constituent function (denoted by OPE_c) by OPE_f. Then the resulting composite OPE (denoted by OPE_{comp}) is evaluated with each $X_i = 0.5$ resulting in p_{m1} which is substituted into Equation 4 to yield the spectral coefficient. Therefore the amount of computation required to obtain each spectral coefficient has a worst case complexity of $O(2^n)$, since at most each node in the linked list will be visited to compute OPE_{comp} and evaluate it for all $X_i = 0.5$. Thus, the overall worst case computational complexity of this approach is $O(m2^n)$ for the computation of m spectral coefficients.

It is illuminating to point out that the methods proposed by Cooley and Tukey [22] originally used to compute the "Fast-Fourier Transform" and later extended for use for digital data [23], have the same complexity as the worst case of this approach if it is desired to compute 2^n coefficients. However, the approach in this paper has three main advantages over the Cooley-Tukey method. They are:

- The Cooley-Tukey approach always has complexity $O(n2^n)$ for computing the spectrum of a Boolean function while $O(n2^n)$ is a worst case complexity arising rarely in this approach due to the fact that most OPE_{comp} functions have fewer terms than 2^n.
- The types of transformations suitable for implementation using the methods of Cooley and Tukey are limited to those with transformation matrices that are recursively defined in order to develop a "butterfly" structure. In the method proposed here any transformation matrix may be used regardless of the size of the row-space or its structure.
- As many or as few spectral coefficients as desired may be computed with our technique. Thus, if fewer than 2^n coefficients are needed, the worst case complexity becomes $O(m2^n)$ where $m < n$.

The complexity of this implementation is comparable to those that compute spectral coefficients from a set of cubes covering the logic function [10] [18] [24]. However, it is quite possible that the number of product terms in the OPE of a given logic function may vary significantly from the corresponding number of SOP terms in a covering expression. Further, there have been other methods for computing spectral coefficients directly from OBDDs of logic functions with a complexity equivalent to the number of edges in the OBDD [12] [13]. However, it has been shown that for certain types of logic functions, the minimal possible number of edges in the corresponding OBDD is exponential with respect to the number of input variables in the function [1]. In these cases the method presented in this paper may provide clear advantages over other methods for computing spectral coefficients.

6 Conclusion

In this work we have provided a technique for computing a spectral coefficient directly from a structural representation of a Boolean function. This result is important since it can provide a more efficient method for computing spectral coefficients for certain classes of logic functions. In particular, those functions that have a small number of product terms in their OPEs, or, those that may not be compactly expressed in OBDD form. The ability to compute spectral coefficients efficiently is very important since they have many applications

in several different areas of digital systems engineering such as synthesis, design verification, and fault detection.

An area of future research is to develop methods to determine the complexity of a given Boolean function in terms of the size of its BDD, or, the number of product terms required in its OPE so that the most optimal method for the computation of its spectral coefficients may be chosen among the several that have been recently proposed.

References

[1] S. Devadas. Comparing Two-Level and Ordered Binary Decision Diagram Representations of Logic Functions. *IEEE Trans. on CAD/ICAS*, vol. CAD-12, no. 5:722–723, May 1993.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincintelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, Boston, Massachusetts, 1984.

[3] K. Keutzer. Dagon: Technology binding and local optimization by dag matching. *Proceedings of ACM/IEEE Design Automation Conference*, pages Miami Beach, Florida, 1987.

[4] H. Savoj, M. J. Silva, R. K. Brayton, and A. Sangiovanni-Vincentelli. Boolean matching in logic synthesis. *Proceedings of the European Design Automation Conference*, pages 168–174, 1992.

[5] K. C. Chen. Boolean matching based on boolean unification. *Proceedings of the European Design Automation Conference*, pages 346–351, 1993.

[6] J. R. Burch and D. E. Long. Efficient boolean function matching. *Proceedings of ICCAD*, pages 408–411, 1992.

[7] F. Mailhot and G. De Micheli. Algorithms for technology mapping based on binary decision diagrams and on boolean operations. *IEEE Trans. CAD/ICAS*, vol. 12, no. 5:599–620, May 1993.

[8] C. K. Chow. On the characterization of threshold functions. *IEEE Special Publication S.134*, pages 34–38, 1961.

[9] S. Muroga. *Threshold Logic and its Applications.* Wiley, New York, 1971.

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

[11] R. Wei and A. Sangiovanni-Vincentelli. Proteus: A logic verification system for combinational circuits. *Proceedings of the International Test Conference*, pages 350–359, 1986.

[12] M. A. Thornton and V. S. S. Nair. Efficient spectral coefficient calculation using circuit output probabilities. *Digital Signal Processing: A Review Journal*, (to appear) December 1994.

[13] M. A. Thornton and V. S. S. Nair. Applications and efficient computation of spectral coefficients for digital logic. *Technical Report*, 94-CSE-13, Feb. 1994.

[14] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang. Spectral transformations for large boolean functions with applications to technology mapping. *Proceedings of ACM/IEEE Design Automation Conference*, pages 54–60, 1993.

[15] M. Fujita, J. Yang, E. M. Clarke, X. Zhao, and P. McGeer. Fast spectrum computation for logic functions using binary decision diagrams. *Proceedings of the International Conference on Circuits and Systems*, 1994.

[16] E. M. Clarke, X. Zhao, M. Fujita, Y. Matsunaga, R. McGeer, and J. Yan. Fast walsh transform computation with binary decision diagram. *Proceedings of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 82–85, September 1993.

[17] E. M. Clarke, K. L. McMillan, X. Zhao, and M. Fujita. Spectral transforms for extremely large boolean functions. *Proceedings of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 86–90, September 1993.

[18] D. Varma and E. A. Trachtenberg. Design automation tools for efficient implementation of logic functions by decomposition. *IEEE Trans. on CAD*, vol. 8, no. 8:901–916, August 1989.

[19] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comp.*, vol. C-35, no. 8:677–691, August 1986.

[20] M. A. Thornton and V. S. S. Nair. An iterative combinational logic synthesis technique using spectral information. *Proceedings of the European Design Automation Conference*, pages 358–363, September 1993.

[21] K. P. Parker and E. J. McCluskey. Probabilistic treatment of general combinational networks. *IEEE Trans. Comp.*, vol. c-24:668–670, June 1975.

[22] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Computation*, vol. 19:297–301, 1965.

[23] J. L. Shanks. Computation of the fast walsh-fourier transform. *IEEE Trans. Comp.*, vol. C-18:457–459, May 1969.

[24] B. J. Falkowski, I. Schafer, and M. A. Perkowski. Calculation of the rademacher-walsh spectrum from a reduced representation of boolean functions. *Proceedings of the European Design Automation Conference*, pages 181–186, September 1992.