Efficient Spectral Coefficient Calculation Using Circuit Output Probabilities

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

Many problems in the field of digital logic may be solved more efficiently in the spectral domain than in the Boolean domain. However, the primary drawback of spectral techniques is the large complexity associated with the calculation of the spectrum of a Boolean function. We present a new method for the computation of a spectral coefficient that has a complexity equal to $O(\mid E \mid)$ where $\mid E \mid$ is the number of edges in a binary decision diagram characterizing the circuit. This result is especially significant for techniques that require the calculation of only a few spectral coefficients since it allows the computations to be accomplished very efficiently and does not require storage resources for a large number values. Furthermore, this method holds for any general spectral transform and does not require the transformation matrix to be recursively defined or sparse.

1 Introduction

There have been many applications proposed and developed using spectral methods for logic circuits. Some of these include logic synthesis [1] [2] [3] [4] [5] [6] [7] , partitioning techniques [8] [3] [9] [10] [11] , testing [12] [13] [14] [15] , function classification [16] [2], and others. The application of spectral based methodologies to digital logic analysis have been studied and developed since the mid-1970's in an attempt to use the vast amount of results that have been very effective in areas such as signal processing and systems analysis. One of the chief reasons that these techniques have not found widespread use and acceptance is the large complexity associated with the computation of the spectrum of a Boolean function. Most digital logic analysis techniques in use today employ heuristic rules that typically have a much smaller complexity than that required for the calculation of a combinational logic circuit spectrum. However, these rule based methods generally perform very poorly, or not at all, for certain subsets of problems. Even when well known spectral computation techniques such as those proposed in [17] for the 'Fast-Fourier Transform' and later extended to transforms suitable for digital logic circuits (the 'Fast-Walsh Transform') [18] are used, the resulting algorithm still suffers from a complexity that is exponential with respect to the number of input variables of the Boolean function. Furthermore, these methods are based upon using recursively defined transformation matrices and are not suitable for generalized transformation matrices. In this paper, we propose an efficient algorithm for the calculation of the spectrum of a Boolean function that does not require the transformation matrix to be recursively defined or sparse, and, has a complexity of the order of the number of edges in a binary decision diagram (BDD) [19] [20].

As mentioned before, researchers have developed efficient spectral coefficient calculation schemes in the past. In particular, a recent method has been proposed that utilizes "integer valued" BDDs [21] [22] [23]. Although this method computes the resulting transform vector in a very compact method by representing it as an integer valued BDD, the determination of each individual spectral coefficient requires a separate evaluation of the BDD. Furthermore, the size of the integer valued BDDs can become exponentially large if the transformation matrices used are not sparse or recursively defined. Our method compares favorably with this approach since we compute a spectral coefficient in linearly bounded time but we also require an exponential number of these calculations to form the entire spectral vector. Further, our method does not require the transformation matrix to be recursively defined or sparse to preserve the efficiency of the computation.

Another methodology allows for the computation of transform coefficients directly from a representation of a Boolean function as a set of disjoint cubes [24] [11]. Unfortunately, as the number of inputs to the Boolean function grows, the corresponding set of disjoint cubes can become extremely large. Our method has the advantage that the function to be transformed can be represented in very compact manner and does not require a large set of product terms.

In this paper, we show that a single spectral coefficient may be computed with a complexity of the order of the number of edges in a BDD. The formulation of this technique requires the use of probability expressions for the output of the circuit to be synthesized. Circuit output probability expressions (OPEs) have been used in the past in areas such as testing [25], analysis [26], and verification [27]. This paper discusses the use of output probabilities to compute spectral coefficients in an efficient manner. To that end, we first develop a new algorithm to compute circuit output probabilities.

The primary reason for the reduction in computation complexity is due to the fact that the output probabilities may be computed efficiently using a BDD representation of the logic circuit. The formulation of the output probability expression requires exponential resources if the Boolean equations are transformed using algebraic methods. However, when the circuit is represented in BDD form, the formulation can be accomplished with $O(\mid E \mid)$ complexity where $\mid E \mid$ represents the number of edges in a BDD.

The remainder of this paper is organized as follows. Section 2 will briefly review the properties of output probability expressions for logic circuits and a new algorithm for calculating circuit output probabilities using BDDs is presented. In section 3, a method for the calculation of spectral coefficients using circuit output probabilities is given. Next, in section 4, we present the algorithm for efficient computation of spectral coefficients and discuss some optimizations to minimize the required complexity. An example spectral coefficient computation using this algorithm is also given. Finally, conclusions will be presented in section 6.

2 Output Probabilities of Combinational Logic Circuits

This section will discuss the computations of circuit output probabilities by briefly reviewing two methods used to compute OPE expressions and then by directly computing a circuit output probability using BDDs. Also, an example of a BDD for a specific logic function is presented. In the discussion presented in the remainder of this paper, the following notation is used:

- 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".

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 [25].

In [25], an algorithm is given to compute the OPE directly from a Boolean expression. This method requires the function to be expressed in a canonical sum-of-products (SOP) form and then each product is replaced by an expression for the probability that the product is at logic "1". The canonical SOP form must be used since it is necessary for one and only one product term to be at logic value "1" for a given input to preserve independence. The rules in table 1 are used to determine the probability expression for each product in the canonical SOP form.

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

This algorithm has a complexity that is exponential with respect to the number of input variables since it requires the formulation of the canonical SOP Boolean function.

As an example of this method, consider the function defined by the truth table in Figure 1:

The canonical SOP form for this function is given in equation 1:

$$f(x) = \overline{x}_3 x_2 x_1 + x_3 x_2 \overline{x}_1 + x_3 x_2 x_1 \tag{1}$$

The resulting OPE using the rules in table 1 is given in equation 2:

$$F(X) = X_2 X_1 + X_3 X_2 - X_3 X_2 X_1 \tag{2}$$

A more efficient algorithm for the computation of the OPE of a Boolean function is also given in [25]. This method requires the function to be represented as a logic diagram. In

x_3	x_2	x_1	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figure 1: Truth Table of Example Function

this 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 equation 1.

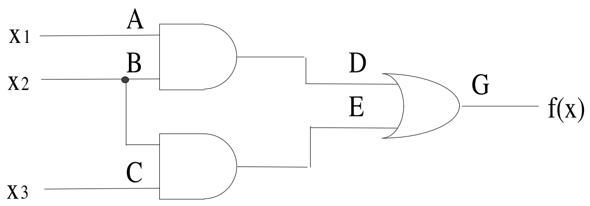

Figure 2: Example of a Binary Decision Diagram

Using the variables assigned to each interconnection and the rules in Table 1, the OPE can be derived as follows.

First, apply the rule for the AND operator:

$$D = AB \tag{3}$$

$$E = BC \tag{4}$$

Next, using the rule for the OR operator:

$$G = AB + BC - AB^2C \tag{5}$$

Finally, the idempotence property rule is employed:

$$G = AB + BC - ABC \tag{6}$$

Notice that the idempotence property is particularly useful since it allows all exponents to be dropped during the formation of the equations. Since we have an expression where the output label is a function only of the primary input labels, we have obtained the OPE. Equation 6 is exactly the same as equation 2. This technique has a complexity of $O(I)$, where I is the number of interconnections in the logic diagram since each interconnection is visited once in the formation of the OPE.

Although the OPE algorithm based upon circuit diagrams is efficient with respect to the size of the circuit, many times it is desired to compute the spectral coefficients of a circuit before it is realized. In particular, spectral based synthesis algorithms typically use some compact representation of the function as input. One compact way of describing a Boolean function is to utilize its BDD, which provides the motivation for computing a circuit output probability using a BDD description as input. For the purposes of computing spectral coefficients, it is sufficient to compute the output circuit probability for the case where the input variables are all equally likely to be "1" or "0". Thus it is not necessary to compute the OPE and then evaluate it for the case where all $X_i = 0.5$ since this probability may be computed directly from the BDD.

A BDD is a graphical representation of a Boolean logic circuit that consists of nodes representing input variables and function output values. These nodes are interconnected by directed edges with the initial node and internal nodes representing function input variables and the terminal nodes representing function output values. Each internal node and the initial node has two directed edges pointing to another node, one of the edges is activated if the input variable is at logic value "1" and the other is activated if the logic variable is at logic value "0". A complete discussion of BDDs may be found in [28] [19] [20]. In [20], some restrictions were placed upon the formation of BDDs that allowed several efficient algorithms to be defined for their manipulation. We will also use the form of BDDs described in [20].

As an example of a BDD, consider the function defined in equation 7:

$$f(x) = x_1 x_3 \overline{x}_6 + x_1 \overline{x}_3 x_4 \overline{x}_6 + x_1 \overline{x}_3 \overline{x}_4 \overline{x}_5 + \overline{x}_1 x_2 x_4 \overline{x}_6 + \overline{x}_1 x_2 \overline{x}_4 \overline{x}_5 + x_1 \overline{x}_2 \overline{x}_5 \tag{7}$$

This function would require a truth table with 2^6 entries to be completely specified. The BDD representation of this function in Figure 3 is quite compact however.

The BDD-based algorithm for the calculation of the output circuit probability does not have the exponential complexity of the algebraic method nor does it require a circuit diagram description of the Boolean function. Only the functionality of the circuit is required which can be expressed in a very compact manner using BDDs. In the remainder of this paper, we will utilize the form of BDD as defined in [20] and we will occasionally refer to some of the BDD algorithms cited there as well.

The following lemma expresses an important result concerning the BDD of a logic function.

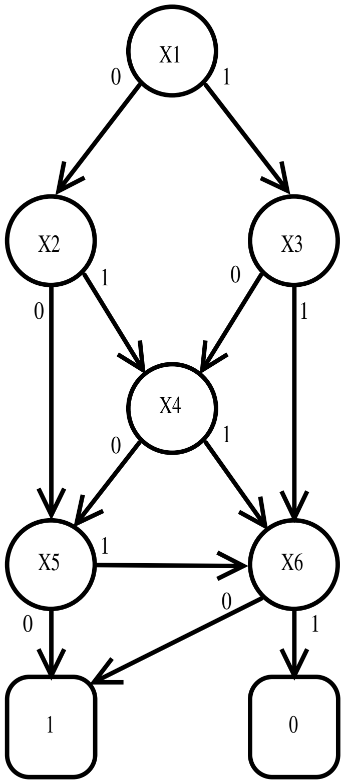

Figure 3: Example of a Binary Decision Diagram

Lemma 1 *For any one particular combination of input variables, at most one path will be activated between the input node and node j where j is any node in the BDD other than an input node.*

Proof:

If possible, let there be more than one path activated between the input node and node j. This implies that at least one of the nodes between the input node and j has both of its outgoing arcs activated for the given input condition which is an impossibility in a BDD. Therefore, there is at most one path activated for a given input condition. □

It should be noted that a path may not exist between the input node and j for certain input conditions.

The algorithm for computing a circuit output probability using the BDD of the circuit and assuming that all inputs are likely to be "1" or "0" is described by the following steps:

Probability Assignment Algorithm

STEP 1: Assign probability $= 1$ for the input node.

STEP 2: If the probability of node $j = P_j$, assign a probability of $\frac{1}{2}P_j$ to each of the outgoing arcs from j.

STEP 3: The probability, P_k, of node k is the sum of the probabilities of the incoming arcs.

Lemma 2 *In the probability assignment algorithm, the probability P_k is the probability that there exists a path from the input node to the node k.*

Proof: In the probability assignment algorithm given in the preceding, P_k is calculated as the sum of the probabilities of reaching node k through various paths from the input node. From lemma 1 all these paths are disjoint and therefore represent disjoint probability events. Thus, P_k is the probability of reaching node k from the input node over all possible input variable combinations. □

This BDD based algorithm for the computation of circuit output probabilities involves the traversal of the BDD from the input node to the terminal nodes. This enables the output probability of a combinational logic circuit to be computed with a complexity equal to $O(\mid E \mid)$, where $\mid E \mid$ is the number of edges or interconnections in the BDD. During the traversal of the BDD, a probability is assigned to each node. This is the probability that the node is reached for a given set of input variable probabilities of the function. Each node probability is a member of a probability space containing 2^n experiments. The node probabilities have the desirable feature of depending only upon their immediate predecessor node probabilities.

As an example, consider the Boolean function,

$$f(x) = \overline{x}_1\overline{x}_2 + x_3 \tag{8}$$

The truth table for equation 8 is given in Figure 4:

The corresponding BDD is given in Figure 5:

It is easily seen from the truth table that the probability that the output is a "1" is $\frac{5}{8}$. Using the algorithm above, each node in Figure 5 is labeled with the probability that it is reached, and it is seen that the terminal "1" node does indeed have the value $\frac{5}{8} = 0.625$.

As mentioned before, this algorithm is applicable only to BDDs that are formulated with restrictions on the variable orderings similar to those first presented in [20]. The reason for this constraint is to ensure that no infeasible paths are utilized in the node probability calculations. For example, if a node corresponding to variable x_i is the input node and this

x_3	x_2	x_1	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Figure 4: Truth Table of Example Function

node is also present internally in the graph, the straight forward application of the probability calculation would include the possibility of assuming x_i is at logic "1" on the input node and it is at logic "0" on the internal node. This is clearly an infeasible path. To eliminate infeasible paths, it is sufficient to constrain all parent nodes to have an index value less than their children nodes.

3 Calculation of Spectral Coefficients

By definition, the spectrum of a Boolean function is obtained by multiplying a transformation matrix by the function's output vector [2]. Although this is generally not the way coefficients are calculated in practice, this definition is convenient for analyzing spectral transforms. The result of the vector-matrix product is termed a spectral vector and it is composed of elements that are referred to as spectral coefficients.

The type of information that the spectral coefficients yield depends upon the form of the transformation matrix. One way to interpret the meaning of each spectral coefficient is to view it as a measure of correlation between two Boolean functions. These two Boolean functions are the function being transformed, $f(x)$, and the constituent function, $f_c(x)$. With this viewpoint, the constituent function is a Boolean function whose output vector is identical to the row vector in the transformation matrix that is used to generate a specific spectral coefficient. Thus, a transformation matrix may be represented as a collection of constituent functions each of whose output vectors are identical to the different row vectors of the transformation matrix.

The following example illustrates an example calculation of a spectrum of a Boolean function. In this example, all logic "1" values are replaced by the integer value, -1, and all logic "0" values are replaced by the integer value, 1.

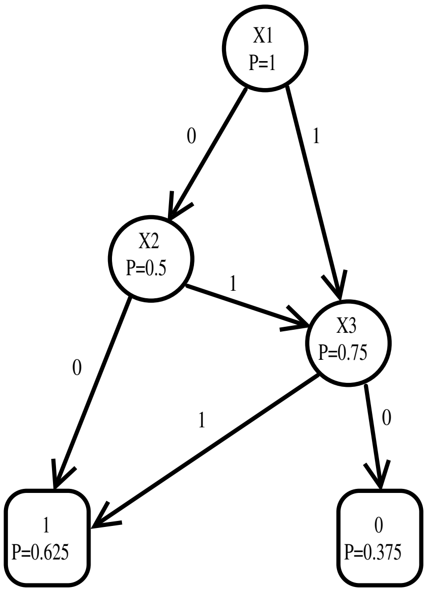

Figure 5: Output Probability Calculation Example

Example 1 *Example of the calculation of the spectrum of a Boolean function*

$$f(x) = \overline{x}_1\overline{x}_3 + x_1\overline{x}_2x_3 + \overline{x}_1x_2 + x_2\overline{x}_3 \tag{9}$$

The truth table for this function is given in Figure 6:

x_1	x_2	x_3	f
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

Figure 6: Truth Table of the Example Function

The transformation matrix to be used is:

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

Thus the resulting spectral vector is calculated as:

$$\underline{S^T} = [-2, -2, 2, -2, 2, -2, 2, 2, -2] \tag{10}$$

These coefficient values may be interpreted as correlation measures between the constituent functions shown to the left of the transformation matrix and the transformed function. For example, the last coefficient in the spectral vector indicates that the constituent function, $x_1 + x_2 + x_3$, has a correlation measure of -2 with the function that was transformed, $\overline{x}_1\overline{x}_3 + x_1\overline{x}_2x_3 + \overline{x}_1x_2 + x_2\overline{x}_3$. The relationship between a spectral coefficient and a coefficient of correlation is formally developed in the following subsection.

3.1 Relevant properties of spectral coefficients

This section will develop some relevant properties of spectral coefficients that are used in the derivation of the algorithm presented in the following section. The following definitions and notations are used in the remaining sections of this paper:

- n is the number of input variables of a Boolean function.
- N_m is a positive integer that has a value equal to the number of outputs of $f(x)$ that are identical to those of $f_c(x)$ (number of matches) for all possible common input combinations.
- N_{mm} is a positive integer that has a value equal to the number of outputs of $f(x)$ that differ from those of $f_c(x)$ (number of mismatches) over all possible common input combinations.
- $S_f[f_c(x)]$ is the spectral coefficient associated with the function, $f(x)$, and the constituent function, $f_c(x)$. A common definition of $S_f[f_c(x)]$ is $S_f[f_c(x)] = N_m - N_{mm}$ [7].

- $R_f(x)$ is a real-valued function that maps the output of a Boolean function, $f(x)$, from logic value "1" to -1 and logic value "0" to 1 for a given set of input values, x.
- C is a coefficient of correlation between two real valued functions and is defined as:

$$C = \frac{1}{2^n} \sum_{i=0}^{n-1} [R_f(x_i) \times R_{fc}(x_i)] \tag{11}$$

- p_m is the percentage of matching outputs between a constituent function and a function to be transformed.
- p_{m0} is the percentage of of matching outputs between a constituent function and a function to be transformed that are at a logic "0" value.
- p_{m1} is the percentage of of matching outputs between a constituent function and a function to be transformed that are at a logic "1" value.

Two useful properties of spectral coefficients are provided in the following two Lemmas that first appeared in [7]:

Lemma 3 *For a given function $f(x)$ and a given constituent function $f_c(x)$ the resulting spectral coefficient is given by:*

$$S_f[f_c(x)] = 2^n - 2N_{mm} = 2N_m - 2^n \tag{12}$$

Lemma 4 *The following property of spectral coefficients holds:*

$$S_f[f_c(x)] = -S_f[\overline{f_c(x)}] \tag{13}$$

The proofs of these Lemmas are provided in [29].

The following Lemma shows the relationship between a spectral coefficient and the correlation between two functions.

Lemma 5 *The spectral coefficient, $S_f[f_c(x)]$ is directly proportional to the coefficient of correlation between $f(x)$ and $f_c(x)$.*

Proof: As given in the definition above, the coefficient of correlation is given by equation 11 as:

$$C = \frac{1}{2^n} \sum_{i=0}^{n-1} [R_f(x_i) \times R_{fc}(x_i)] \tag{14}$$

Where, x_i, is the $i\underline{^{th}}$ unique minterm. Note that each product in the summation of the series is either 1 or -1. Thus we can replace $\sum_{i=0}^{n-1}[f(x_i) \times f_c(x_i)]$ with $N_m - N_{mm}$. By the definition given above, $S_f[f_c(x)] = N_m - N_{mm}$. Substituting $S_f[f_c(x)]$ into 11:

$$C = \frac{1}{2^n} S_f[f_c(x)] \tag{15}$$

Hence, $S_f[f_c(x)]$ is directly proportional to C with a constant proportionality coefficient of 2^n. □

Similar results can be proven for other definitions of spectral coefficients. For instance, the Reed-Muller transform [30] [31] can be defined as a vector of values where each component is the number of matching logic "1" outputs (calculated as $p_{m1} \times 2^n$) between the function to be transformed and a constituent function.

3.2 Relevance of OPEs to spectral coefficients

Since we can compute the spectral coefficients given the value N_m or N_{mm}, an efficient way to compute these quantities will in effect provide an efficient way to calculate the spectral coefficients. Furthermore, if we know the percentage of the matching outputs of a constituent function and the function to be transformed (denoted by p_m), we can easily compute $N_m = p_m 2^n$. This observation is the basis behind the algorithm to compute the spectral coefficients.

In order to determine p_m, we need to use logic equations that indicate when the outputs of the constituent function and the function to be transformed match. It is trivial to show that such logic equations can always be formed by using the logical AND of these two functions for the case when both output a "1", and, the logical NAND of these two functions when both output a "0". A formal definition of these types of functions follows:

Definition 1 *A function that is formed by taking the logical AND or NAND of a constituent function and a function to be transformed is called a 'composite function' and is denoted by* $f_{comp}(x)$.

Therefore, in order to compute the value p_m we only need to find the probability that both functions simultaneously output a logic "1" value (p_{m1}) and the probability that both functions simultaneously output a logic "0" value (p_{m0}). By forming the BDD of the two

$f_{comp}(x)$ functions, p_{m0} and p_{m1} are simply the probabilities that the terminal node of logic value "1" is reached.

In Lemma 6 an important result is given relating the spectral coefficients and the $f_{comp}(x)$ functions. This result is presented by using the concepts of canonical sum-of-products (SOP) and product-of-sum (POS) forms of Boolean expressions.

Lemma 6 *$N_m = N_{m1} + N_{m0}$, where N_{m1} = the number of minterms terms in a canonical SOP form of $f \cdot f_c$ and N_{m0} = the number of maxterms terms in a canonical POS form of $f_c + f$*

Proof: All Boolean expressions may be expressed by indicating the output value corresponding to each of its 2^n minterms (this is in fact a truth table). A canonical SOP form for a Boolean expression is the inclusive-OR of all minterms that produce a logic "1" output [32]. Hence, the number of minterms present in a canonical SOP expression represents the number of times the function output is at logic value "1".

Likewise, N_{m0} is equal to the number of maxterms in a canonical POS form of $f + f_c$ since this expression will be at logic "0" if and only if both f and f_c output "0" for a common set of inputs.

Since N_m is the number of times a constituent function, $f_c(x)$, and a function to be transformed, $f(x)$, have identical outputs for a common set of inputs.

$$N_m = N_{m1} + N_{m0} \tag{16}$$

□

The relationship between the output probability of a composition function and N_m is established in Lemma 7:

Lemma 7

$$N_m = 2^n[1 + \wp\{f \cdot f_c\} - \wp\{f + f_c\}] \tag{17}$$

Proof: $\wp\{f + f_c\}$ yields the probability that the function $f + f_c$ produces a logical "1". Therefore, $1 - \wp\{f + f_c\}$ is the probability that $f + f_c$ produces a logic "0". Since $f + f_c$ will output a "0" if and only if both f and f_c are at "0":

$$p_{m0} = 1 - \wp\{f + f_c\} = 1 - \frac{1}{2^n}(N_{m0}) \tag{18}$$

Likewise, $\wp\{f \cdot f_c\}$ yields the percentage of minterms of $f \cdot f_c$ that produce a logic "1" for the function, $f \cdot f_c$. Since $f \cdot f_c$ will output a "1" if and only if both f and f_c are at "1":

$$p_{m1} = \wp\{f \cdot f_c\} = \frac{1}{2^n}(N_{m1}) \tag{19}$$

Substituting 18 and 19 into 17 and observing that $p_m = p_{m1} + p_{m0}$:

$$N_m = 2^n[p_{m0} + p_{m1}] = p_m 2^n \tag{20}$$

Thus, the definition of N_m is satisfied and the proof is complete. □

Based on the results of the previous Lemmas, we can now prove that a spectral coefficient may be calculated based upon circuit output probabilities.

Theorem 1

$$S_f[f_c(x)] = 2^n[1 + 2(\wp\{f \cdot f_c\} - \wp\{f + f_c\})] \tag{21}$$

Proof:

From Lemma 12:

$$S_f[f_c(x)] = 2N_m - 2^n \tag{22}$$

From Lemma 17:

$$N_m = 2^n[1 + \wp\{f \cdot f_c\} - \wp\{f + f_c\}] \tag{23}$$

Substituting 23 into 22 and simplifying:

$$S_f[f_c(x)] = 2^n[1 + 2(\wp\{f \cdot f_c\} - \wp\{f + f_c\})] \tag{24}$$

□

Corollary 1 *A compact expression for $S_f[f_c(x)]$ is:*

$$S_f[f_c(x)] = 2^n[2p_m - 1] \tag{25}$$

Proof: From Theorem 1,

$$S_f[f_c(x)] = 2^n[1 + 2(\wp\{f \cdot f_c\} - \wp\{f + f_c\})] \tag{26}$$

Substituting equations 18 and 19 into equation 26:

$$S_f[f_c(x)] = 2^n[2(p_{m1} + p_{m0}) - 1] \tag{27}$$

From the definition p_m:

$$S_f[f_c(x)] = 2^n[2p_m - 1] \tag{28}$$

□

These results show that the calculation of spectral coefficients is translated to the problem of output probability calculations of the BDDs of composition functions. It should be noted that in most methods that utilize spectral techniques for digital logic circuits, $f_c(x)$ is much less complex than than the function to be transformed, $f(x)$. For example, in the synthesis algorithm proposed by us in [7], we present a method for synthesizing a function by decomposing it into a collection of much simpler constituent functions. The decomposition was accomplished by using the information contained in the corresponding spectral coefficients.

3.3 Complexity of the spectral computation algorithm

In order to implement these results to formulate an algorithm for the computation of a spectral coefficient, the following observations are made. The value p_m is obtained by using the BDD based output probability calculation algorithm presented in section 2. p_m is computed as the sum of p_{m0} and p_{m1} which are obtained by applying the output probability calculation algorithm to the BDDs formed by two composition functions denoted by $f1_{comp}(x)$ and $f2_{comp}(x)$. These composition functions are given by $f1_{comp}(x) = f_c(x) \cdot f(x)$ and $f2_{comp}(x) = \overline{f_c(x)} \cdot \overline{f(x)}$. Therefore, the values p_{m1} and p_{m0} are obtained with a complexity of $O(\mid E_{comp} \mid)$ where E_{comp} is the number of edges present in the BDDs of the two composition functions.

If the algorithm $APPLY$ proposed in [20] is used to form the composition function BDDs, the resulting complexity is $O(\mid E_{f_c} \mid\mid E_f \mid)$. Where $\mid E_{f_c} \mid$ is the number of edges in the BDD of the constituent function, $f_c(x)$, and $\mid E_f \mid$ is the number of edges in the BDD of the function to be transformed, $f(x)$. This bound is very good since for most transforms the constituent functions are very small as compared to the function to be transformed. In the general case however, constituent functions may be as complex as the function to be transformed, or, even more complex.

Thus, to form a spectral coefficient it is only necessary to apply the output probability algorithm to the BDDs of the composition functions and then compute the following:

$$p_{m1} = \wp\{f(x) \cdot f_c(x)\} \tag{29}$$

$$p_{m0} = \wp\{\overline{f(x)} \cdot \overline{f_c(x)}\} \tag{30}$$

$$S_f[f_c(x)] = 2^n[2(p_{m1} + p_{m0}) - 1] \tag{31}$$

The algorithm for the efficient computation of spectral coefficients is stated as:

Efficient Spectral Coefficient Computation Algorithm

STEP 1 Formulate the BDDs for the two composition functions using the $APPLY$ algorithm.

STEP 2 Use the output probability calculation algorithm to form the composition function BDDs.

STEP 3 Compute $p_{m1} = \wp\{f(x) \cdot f_c(x)\}$ and $p_{m0} = \wp\{\overline{f(x)} \cdot \overline{f_c(x)}\}$.

STEP 4 Compute $S_f[f_c(x)] = 2^n[2(p_{m1} + p_{m0}) - 1]$.

Since the bounding operation in this algorithm is the utilization of the $APPLY$ algorithm to form the composition function BDDs, computational complexity of this algorithm is $O(\mid E_f \mid \times \mid E_{f_c} \mid)$.

Next, an example of the application of this algorithm to a 3-input logic function is given.

Example 2 *Example of the efficient calculation of a spectral coefficient using output probabilities and BDDs.*

The function to be transformed, $f(x)$, is given by equation 32:

$$f(x) = \overline{x}_1\overline{x}_2 + x_3 \tag{32}$$

The constituent function for this example, $f_c(x)$, is given as:

$$f_c(x) = x_2 + x_3 \tag{33}$$

The BDD for equation 32 is given in Figure 5. The BDD for the composition function, $f(x) \cdot f_c(x)$ is given in Figure 7, and the BDD for the composition function, $\overline{f(x)} \cdot \overline{f_c(x)}$ is given in Figure 8:

In order to compute the spectral coefficient determined by the constituent function given in equation 33, the values p_{m1} and p_{m0} are computed using the output probability algorithm. The node probabilities are shown on the composition BDDs. These values are:

$$p_{m1} = 0.5 \tag{34}$$

$$p_{m0} = 0.125 \tag{35}$$

Next, the spectral coefficient is computed as:

$$S_f[f_c(x)] = 2^3[2(0.5 + 0.125) - 1] = 2 \tag{36}$$

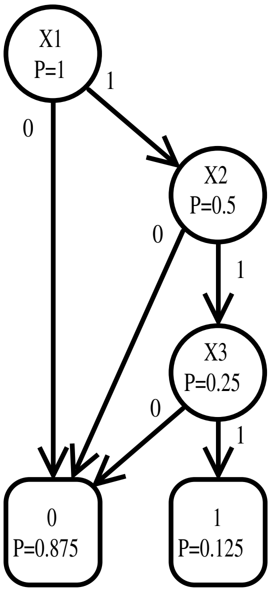

Figure 7: BDD of the composition function, $f(x) \cdot f_c(x)$

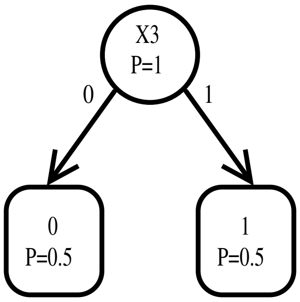

Figure 8: BDD of the composition function, $\overline{f(x)} \cdot \overline{f_c(x)}$

Applying the definition of a transform to this problem would have resulted in computing the dot-product of two vectors with 2^3 elements each. The use of "fast" algorithms proposed by [17] and [18] are prohibited since the inclusive-OR based transform does not yield a sparse or recursively defined transformation matrix (an example of this transformation matrix for 3-input variables is given in Example 1). Further, the application of the spectral calculation algorithm presented in [21] [22] [23] may result in the formation of a very large "integer-valued" BDD since the matrix is not sparse and cannot be recursively defined.

4 Conclusion

The theoretical relationships between circuit output probabilities and the spectrum of a Boolean function were developed. These relationships were utilized to form an efficient algorithm for the computation of a spectral coefficient of a Boolean function. The circuit output probabilities were computed using a BDD representation of the logic circuits resulting in an efficient computation method that does not require the circuit to be realized in schematic form. The resulting algorithm is efficient in terms of required computation time and storage. These results are especially significant for spectral based methodologies that require the calculation of a limited subset of coefficients rather than the entire spectral vector (which contains 2^n components).

The algorithm was applied to an example where the transformation matrix was neither sparse nor recursively defined. The results of the example illustrated how this method can yield a spectral coefficient very efficiently for a case where other known methods would require exponential resources.

References

[1] C. R. Edwards. The design of easily tested circuits using mapping and spectral techniques. *Radio and Electronic Engineer*, vol. 47, no. 7:321–342, 1977.

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

[3] M. G. Karpovsky. *Finite Orthogonal Series in the Design of Digital Devices.* John Wiley, New York, NY, 1976.

[4] A. M. Lloyd. A consideration of orthogonal matrices, other than the rademacher-walsh types, for the synthesis of digital networks. *J. Electronics*, vol. 47, no. 3:205–212, 1979.

[5] M. A. Perkowski, M. Driscoll, J. Liu, D. Smith, J. Brown, L. Yang, A. Shamsapour, M. Helliwell, B. Flakowski, and A. Sarabi. Integration of logic synthesis and high-level synthesis into the diades design automation system. *Proceedings of 22nd IEEE Int. Symp. on Circuits & Systems*, pages 748–751, 1989.

[6] M. Stanković, Z. Tošić, and S. Nikolić. Synthesis of maitra cacades by means of spectral coefficients. *IEE Proceedings*, vol. 130, Pt. E, No. 4:101–108, July 1983.

[7] 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.

[8] R. L. Ashenhurst. The decomposition of switching functions. *Proceedings of an International Symposium on the Theory of Switching*, pages 74–116, April 1957.

[9] R. Lechner. Harmonic analysis of switching functions. *in Recent developments in switching theory*, pages 121–228, 1971.

[10] V. M. Tokmen. Disjoint decomposability of multiple valued functions by spectral means. *Proceedings IEEE 10th Int. Symp. Mult. Valued Logic*, pages 88–93, 1980.

[11] 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.

[12] T. Damarla. Generalized transforms for multiple valued circuits and their fault detection. *IEEE Trans. Comp.*, vol. C-41, no. 9:1101–1109, September 1992.

[13] T. C. Hsiao and S. C. Seth. An analysis of the use of rademacher-walsh spectrum in compact testing. *IEEE Trans. Comp.*, vol. C-33, no. 10:934–937, October 1984.

[14] D. M. Miller and J. C. Muzio. Spectral fault signatures for single stuck-at faults in combinational networks. *IEEE Trans. Comp.*, vol. C-33, no. 8:765–768, August 1984.

[15] A. K. Susskind. Testing by verifying walsh coefficients. *IEEE Trans. Comp.*, vol. C-32, no. 2:198–201, February 1983.

[16] C. R. Edwards. The application of the rademacher-walsh transform to boolean function classification and threshold logic synthesis. *IEEE Trans. Comp.*, pages 48–62, 1975.

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

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

[19] S. B. Akers. Binary decision diagrams. *IEEE Trans. Comp.*, vol. C-27, no. 6:509–516, June 1978.

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

[21] 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.

[22] 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.

[23] 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.

[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.

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

[26] S. K. Kumar and M. A. Breuer. Probabilistic aspects of boolean switching functions via a new transform. *Journal of the ACM*, vol. 28, No. 3:502–520, July 1981.

[27] J. Jain, J. Bitner, D. S. Fussell, and J. A. Abraham. Probabilistic design verification. *Technical Report, University of Texas at Austin*, UT-CERC-TR-JAA91-01, April 1991.

[28] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, vol. 38:985–999, July 1959.

[29] M. A. Thornton and V. S. S. Nair. Iterative combinational logic synthesis techniques using spectral data. *Technical Report*, 93-CSE-8, 1993.

[30] D. Green. *Modern Logic Design.* Addison-Wesley, Reading, Massachusetts, 1986.

[31] M. A. Thornton and V. S. S. Nair. A numerical method for reed-muller circuit synthesis. *Proceedings of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 69–74, September 1993.

[32] M. Mano. *Digital Design.* Prentice Hall, Englewood Cliffs, New Jersey, 1984.