Efficient Calculation of Spectral Coefficients and Their Applications

Mitchell A. Thornton, *Member, IEEE*, and V. S. S. Nair, *Member, IEEE*

***Abstract*—Spectral methods for analysis and design of digital logic circuits have been proposed and developed for several years. The widespread use of these techniques has suffered due to the associated computational complexity. This paper presents a new approach for the computation of spectral coefficients with polynomial complexity. Usually, the computation of the spectral coefficients involves the evaluation of inner products of vectors of exponential length. In the new approach, it is not necessary to compute inner products, rather, each spectral coefficient is expressed in terms of a measure of correlation between two Boolean functions. This formulation coupled with compact BDD representations of the functions reduces the overall complexity. Further, some computer aided design applications are presented that can make use of the new spectrum evaluation approach. In particular, the basis for a synthesis method that allows spectral coefficients to be computed in an iterative manner is outlined. The proposed synthesis approach has the advantage that it can accommodate a wide variety of constituent gates, including XOR gates, and complex subfunctions for realizing the circuits.**

I. Introduction

THE spectral information of a Boolean function yields information regarding the correlation between the input variables and the output of the function. The exploitation of the spectral data provides a sound mathematical basis for logic function synthesis [11], [22]. However, the primary drawback of spectral techniques is the large complexity associated with the calculation of the spectrum of a Boolean function. The results of the research discussed in this paper provide a new method for computing the spectrum of a function based upon its output probability. By computing the spectrum in this manner, the complexity of the calculations are greatly reduced since the coefficients are obtained without evaluating inner products. The use of circuit output probabilities in other endeavors in the field of digital systems engineering have yielded encouraging results. Some of these areas are verification [21], analysis [23], and testing [30].

Recently, some efficient spectral coefficient calculation schemes have been developed by other researchers. In particular, a method has been proposed that utilizes "integer valued" binary decision diagrams (BDD's) to represent the resulting spectral vector [6]–[8]. This technique requires the particular transformation matrix to be recursively defined or sparse in order to generate the matrix product as a series of recursive computations. The complexity of the spectral computation method presented here is comparable with the integer valued BDD approach since a spectral coefficient is computed in polynomially bounded time without the requirement that the transformation matrix be recursively defined or sparse to preserve the efficiency of the computation.

Further, by using the approach in [6]–[8], some resulting integer valued BDD's may be very large, particularly those with many different spectral coefficient values. This provides the motivation for the development of algorithms to compute a single coefficient. In [16], the method using integer valued BDD's described above has been extended to allow for the computation of a subset of coefficients. This technique allows for individual rows of the transformation matrix to be represented as integer valued BDD's and they are multiplied using the method in [7] resulting in an integer valued BDD representing the subset of coefficients corresponding to the particular rows of the transformation matrix chosen. Our method can be more efficient for the computation of a single spectral coefficient since it is only necessary to compute the output probabilities of two BDD's and then compute the spectral coefficient directly instead of recursively building a resultant integer valued BDD.

Another fairly recent methodology allows for the computation of transform coefficients directly from a representation of a Boolean function as a set of disjoint cubes [14], [41]. 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 is represented in a very compact manner requiring no product terms in the representation.

In addition to the presentation of the new method for computing the spectral coefficients, a discussion of some applications using this method is provided. In particular, a spectral based logic synthesis algorithm is outlined. This synthesis approach offers advantages in the synthesis of digital logic circuits since it does not require a specific transformation matrix to be used. As a result, the XOR gate is allowed to be integrated into the resulting design as well as other arbitrary gates or logic functions. In the past, most spectral synthesis algorithms could only achieve this capability if the circuit to be designed was partitioned into linear and nonlinear subcircuits [41]. The synthesis approach presented here is very general in that any set of subfunctions may be used to realize the desired function. A constituent subfunction may include simple gates or more complex functions such

Manuscript received December 5, 1994; revised April 18, 1995. This work was supported in part by the National Science Foundation under Grant MIP-9410822. This paper was recommended by Associate Editor M. Fujita.

M. A. Thornton is with the Department of Computer Systems Engineering, University of Arkansas, Fayetteville, AR 72701 USA.

V. S. S. Nair is with the Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275 USA.

IEEE Log Number 9413203.

as AND-OR-INVERT (AOI), multiplexors, and small look-up tables.

Many synthesis systems provide only a minimized Boolean function as output or utilize Boolean expressions in the intermediate formation of the output netlist. The synthesis approach that is outlined here differs in that no intermediate Boolean expressions are utilized. Thus, this method does not rely on symbolic algebraic manipulation algorithms. The computations are performed using graph algorithms to manipulate BDD's and floating point arithmetic to compute the circuit output probabilities and spectral coefficients.

The remainder of this paper is organized as follows: Section II will provide a brief review of circuit output probability expressions (OPE's), and it will introduce a new algorithm for the computation of a circuit output probability directly from a BDD representation. Next, in Section III, the relationship between circuit output probabilities and spectral coefficients is mathematically derived. This derivation shows how the spectral coefficients may be computed directly from the circuit output probabilities thus eliminating the need to compute an inner product. Following the derivation, some applications of the use of spectral coefficients are described in Section IV. In particular, a spectral-based synthesis algorithm is outlined and its relative merits are discussed. The complexity of the spectral coefficient calculation method is analyzed in Section V. The resulting impact on the complexity of other spectral based applications is also discussed. Finally, Section VI will provide experimental results obtained by using the ISCAS85 benchmarks circuits as examples followed by some conclusions.

II. Output Probabilities of Combinational Logic Circuits

This section provides a discussion of circuit output probabilities by briefly reviewing two methods used to compute OPE expressions described in [30]. Following the review, a new method is developed that computes a circuit output probability using a BDD description as input. 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 operation or the addition of real numbers depending upon the context of the equation in which it is used.
- The operator symbol, "·" will refer to the Boolean AND operation. The absence of an operator between two adjacent variables in a Boolean equation implies the presence of the · operator.
- The operator symbol, "×" 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 × 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 yields 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."

TABLE I
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	$x_i \cdot x_i$	X_i

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

Fig. 1. Truth table of example function for OPE computation.

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 [30].

In [30], 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 I are used to determine the probability expression for each product in the canonical SOP form.

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 form of the Boolean function. As an example of this method, consider the function defined by the truth table in Fig. 1.

The canonical SOP form for this function is given in (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 I is given in (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 [30]. This method

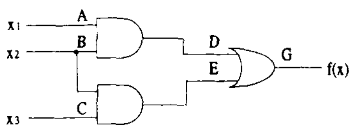


Fig. 2. Logic diagram of the example circuit for OPE computation.

requires the function to be represented as a logic diagram. In this technique, each primary input, each internal interconnection, and the output is assigned a unique variable name. Using the rules in Table I, 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 Fig. 2 that is a realization of (1).

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

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 (6) is an expression where the output label is a function only of the primary input labels, the OPE has been obtained. 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 desirable 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 BDD's may be found in [1], [4], [25]. In [4], some restrictions were placed upon the formation of BDD's that allowed several efficient algorithms to be defined for their manipulation.

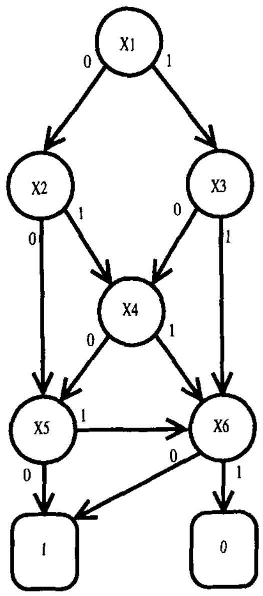


Fig. 3. Example of a binary decision diagram.

As an example of a BDD, consider the function defined in (7)

$$\begin{aligned} f(x) = {} & x_1x_3\overline{x}_6 + x_1\overline{x}_3x_4\overline{x}_6 + x_1\overline{x}_3\overline{x}_4\overline{x}_5 \\ & + \overline{x}_1x_2x_4\overline{x}_6 + \overline{x}_1x_2\overline{x}_4\overline{x}_5 + x_1\overline{x}_2\overline{x}_5. \end{aligned} \tag{7}$$

This function would require a truth table with 2^6 entries to be completely specified since there are 6 primary inputs. However, the BDD representation of this function in Fig. 3 is quite compact.

The BDD-based algorithm for the calculation of the output circuit probability does not have the exponential complexity of the algebraic method mentioned above 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 BDD's. In the remainder of this paper, we will utilize the form of BDD as defined in [4] 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.

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. □

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

Fig. 4. Truth table of example function for probability assignment algorithm.

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

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

Probability Assignment Algorithm:

1) Assign probability $= 1$ for the input node.
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.
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: Consider a single parent node, j, and one of its child nodes, k. Given that j has been reached initially, the probability that an arc from j to k is activated for a given set of input values is $\frac{1}{2}$ since every input or internal node in a BDD has two exiting arcs. The probability that the parent node, j, has been reached is P_j. Thus, the probability that the node k has been reached given that j has been reached is the conditional probability, $P_{k|j} = \frac{1}{2} P_j$. From Lemma 1, it is shown that for a given set of input conditions there exists only one path from the input node to the node, k. Therefore, the overall probability that node k is reached, P_k, is the sum of all conditional probabilities that the incoming arcs to k are activated. □

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 of the probability assignment algorithm, consider the Boolean function expressed in (8)

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

The truth table for (8) is given in Fig. 4 and the corresponding BDD is given in Fig. 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 Fig. 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 BDD's that are formulated with restrictions on the variable orderings similar to those first presented in [4]. 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 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 since it does not exist for a fixed set of function inputs. To eliminate infeasible paths, it is sufficient to constrain all parent nodes to have an input variable index value less than that of their children nodes.

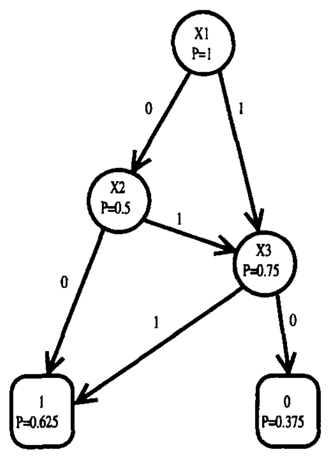


Fig. 5. Output probability calculation example.

III. 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 [20]. Although this is not necessarily 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 a constituent function, $f_c(x)$. With this viewpoint, the constituent function is a Boolean function whose output vector is identical to a 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 various 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.

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

$$f(x) = \overline{x}_1\overline{x}_3 + x_1\overline{x}_2 x_3 + \overline{x}_1 x_2 + x_2\overline{x}_3. \tag{9}$$

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

Fig. 6. Truth table of the example function for spectrum computation.

$$\begin{array}{r} 0 \\ x_1 \\ x_2 \\ x_3 \\ x_1+x_2 \\ x_1+x_3 \\ x_2+x_3 \\ x_1+x_2+x_3 \end{array}\left[\begin{array}{rrrrrrrr} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{array}\right]$$

Fig. 7. Transformation matrix for example spectrum calculation.

The truth table for this function is shown in Fig. 6 and the transformation matrix to be used is shown in Fig. 7.

The resulting spectral vector is given in (10)

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

These spectral 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. All of the mathematical details of this formulation may be found in [37].

A. 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. These definitions 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, $f_c(x)$.
- $R_f(x)$ 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 discrete functions and is defined as

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

- p_m is the percentage of matching outputs between a constituent function and a function to be transformed.
- p_{m0} is the percentage of 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 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 [39].

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

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

Proof: The maximum possible absolute value of a spectral coefficient occurs when a row of the matrix is equal to the function output vector or when each component of the vector is the negative of the corresponding entry in the transform matrix row. Hence, the maximum possible absolute value of the spectral coefficient is $|S_F[F_c(x)]| = 2^n$ indicating 100% positive or negative correlation between $F(x)$ and $\overline{F_c(x)}$. Indeed in this case, either $F(x) = F_c(x)$ or $F(x) = \overline{F_c(x)}$. Each mismatch present in the function output vector and the corresponding matrix row entry always produces a product value of -1. Therefore, N_{mm} mismatches result in a negative partial sum of $-N_{mm}$. The only other possibility is a match which is the complement of mismatches and always produces a product value of +1. Since the spectral coefficient for $F(x)$ and $F_c(x)$ is the difference between the number of matches, N_m, and the number of mismatches, N_{mm}

$$\begin{aligned} S_F[F_c(x)] &= N_m - N_{mm} \\ &= N_m - [2^n - N_m] \\ &= 2N_m - 2^n. \end{aligned}$$

Likewise, substituting N_{mm}

$$\begin{aligned} S_F[F_c(x)] &= N_m - N_{mm} \\ &= [2^n - N_{mm}] - N_{mm} \\ &= 2^n - 2N_{mm}. \end{aligned}$$

Hence, $S_F[F_c(x)] = 2^n - 2N_{mm} = 2N_m - 2^n$. □

Lemma 4: The following property of spectral coefficients holds

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

Proof: Let the number of mismatches between the inverse of the constituent function, $\overline{F_c(x)}$, be denoted by N'_{mm} and the corresponding matches denoted by N'_m.

Thus, $N'_m = N_{mm}$. Using this fact and the results from Lemma 3

$$\begin{aligned} S_F[F_c(x)] &= 2^n - 2N_{mm} \\ &= 2^n - 2N'_m \\ &= -(2N'_m - 2^n) \\ &= -S_F[\overline{F_c(x)}]. \end{aligned}$$

□

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 (1) 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 ith 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 $N_m - N_{mm}$. From the results of Lemma 3, $S_f[f_c(x)] = N_m - N_{mm}$. 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 [17], [38], 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.

B. Relevance of OPE's 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. 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"

$$p_{m0} = 1 - \wp\{f + f_c\} = 1 - \frac{1}{2^n}(N_{m1}). \tag{18}$$

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:

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

Proof: From Lemma 3

$$S_f[f_c(x)] = 2N_m - 2^n. \quad (22)$$

From Lemma 7

$$N_m = 2^n[1 + \wp\{f \cdot f_c\} - \wp\{f + f_c\}]. \quad (23)$$

Substituting (23) into (22) and simplifying

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

□

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

$$S_f[f_c(x)] = 2^n[2p_m - 1]. \quad (25)$$

Proof: From Theorem 1

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

Substituting (18) and (19) into (26)

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

From the definition p_m

$$S_f[f_c(x)] = 2^n[2p_m - 1]. \quad (28)$$

□

In order to use the governing relationship between the circuit output probability and the spectral coefficients given in (27) to formulate an algorithm, the following observations are made. The value p_m is obtained by using the BDD based output probability calculation algorithm presented in the previous section. 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 BDD's 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 $f2_{comp}(x) = \overline{f_c(x)} \cdot \overline{f(x)}$.

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

$$p_{m1} = \wp\{f(x) \cdot f_c(x)\} \quad (29)$$
$$p_{m0} = \wp\{\overline{f(x)} \cdot \overline{f_c(x)}\} \quad (30)$$
$$S_f[f_c(x)] = 2^n[2(p_{m1} + p_{m0}) - 1]. \quad (31)$$

The algorithm for the efficient computation of spectral coefficients is stated as the *efficient spectral coefficient computation algorithm.*

1) Formulate the BDD's for the two composition functions using the *APPLY* algorithm.

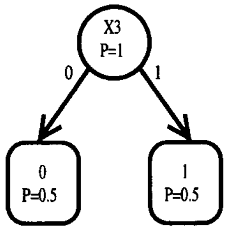


Fig. 8. BDD of the composition function, $f(x) \cdot f_c(x)$.

2) Use the output probability calculation algorithm to form the composition function BDD's.
3) Compute $p_{m1} = \wp\{f(x) \cdot f_c(x)\}$ and $p_{m0} = \wp\{\overline{f(x)} \cdot \overline{f_c(x)}\}$.
4) Compute $S_f[f_c(x)] = 2^n[2(p_{m1} + p_{m0}) - 1]$.

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

C. Example of the Efficient Spectral Coefficient Computation Algorithm

This subsection will provide 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 BDD's.

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

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

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

$$f_c(x) = x_2 + x_3. \quad (33)$$

The BDD for (32) is given in Fig. 5. The BDD for the composition function, $f(x) \cdot f_c(x)$ is given in Fig. 8, and the BDD for the composition function, $\overline{f(x)} \cdot \overline{f_c(x)}$ is given in Fig. 9.

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

$$p_{m1} = 0.5 \quad (34)$$
$$p_{m0} = 0.125. \quad (35)$$

Next, the spectral coefficient is computed as

$$S_f[f_c(x)] = 2^3[2(0.5 + 0.125) - 1] = 2. \quad (36)$$

Applying the definition of a transform to this problem would have resulted in computing the dot-product of two vectors with

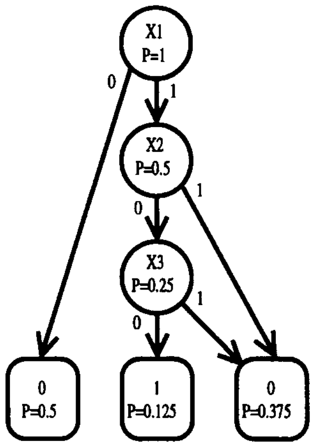


Fig. 9. BDD of the composition function, $\overline{f(x)} \cdot \overline{f_c(x)}$.

2^3 elements each. The use of "fast" algorithms proposed by [9], [32] 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 [6]–[8], may result in the formation of a very large "integer-valued" BDD since the matrix is not sparse and cannot be recursively defined.

IV. Applications of Spectral Coefficients

Many applications have been proposed and developed using spectral methods for logic circuits. Some of these include logic synthesis [13], [20], [22], [26], [31], [33], [39]-[41], testing [10], [18], [29], [34], function classification [5], [12], [20], and others. The application of spectral based methodologies to digital logic analysis has 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.

A. Function Decomposition

One of the chief reasons that spectral techniques have not found widespread use and acceptance is the large complexity associated with the computation of the spectrum of a Boolean function. For example, in the disjoint decomposition method [41], the functions were limited to about 20 inputs in order to keep the amount of computations manageable. The use of alternative, more efficient spectral computation algorithms such as the one presented in this paper could prove to be applicable to this problem.

B. Fault Detection Using Spectral Coefficients

The fault detection problem for digital logic circuits is becoming more important as the size of a typical circuit increases. There have been several methods proposed using spectral techniques. Most of these methods compute the spectral coefficients by performing an inner-product calculation using vectors with 2^n components. If the method requires the entire spectrum, 2^n of these inner-products are computed.

In the work presented in [34], a methodology was described where only 1 or 2 coefficients are used to determine the presence of a fault. This simplification along with the use of the efficient method for computing spectral coefficients proposed here results in a very efficient method for fault detection by verifying spectral coefficients. In fact, the method proposed for computing the coefficients as described here has the advantage that a single coefficient can be computed without determining the remaining $2^n - 1$ coefficients. Even methods that utilize the entire spectral vector such as the one in [29], could benefit substantially if a more efficient way of computing each coefficient is employed.

Another advantage is that the spectral coefficient computation method developed here could have used logic diagrams instead of BDD's to represent the circuits. This is because an efficient algorithm for computing circuit output probabilities using logic diagrams was given in [30]. The ideas presented in the preceding section that are used to relate output probabilities and spectral values were applied to structural representations of digital logic circuits in [36]. This could be a definite advantage in this area since the logic diagram may be more readily available than a BDD description of the circuit under analysis.

C. Function Classification Using Spectral Coefficients

The use of spectral coefficients has been proven to uniquely specify threshold functions. The work by Chow [5], resulted in a formal proof that $n + 1$ spectral coefficients are sufficient to define a specific threshold function. In terms of the definitions used in this paper, the constituent functions that are used to compute these coefficients are $f_c(x) = 0$ and $f_c(x) = x_i$, where each x_i is a primary input to the circuit. Many times this subset of spectral coefficients is referred to as the "Chow parameters."

Unfortunately, the Chow parameters do not uniquely specify the Boolean functions that are the subject of this paper. However, the Chow parameters can used to classify all possible Boolean functions into various subsets. Many results and further classification results have been developed based upon studying the properties of the functions in a particular subset [12]. Recently, it has been shown that some classes of functions may be classified by using a small subset of $2n + 2$ spectral coefficients [35]. Clearly, before any of these functions may be classified, it is necessary to compute their Chow parameters. The use of an efficient means for performing this computation would allow for faster classifications. As in the case of the spectral-based testing algorithms discussed above, the efficient means for computing the spectral coefficients described here may be superior to other methods since it is well-suited for computing a single coefficient at a time.

D. Spectral Based Logic Synthesis

Most spectral synthesis methods developed in the past utilize orthogonal, recursively defined, transformation matrices. The synthesis method outlined here is capable of using any general transformation matrix and is dependent on the use of error functions in an iterative fashion. Fig. 10 illustrates the

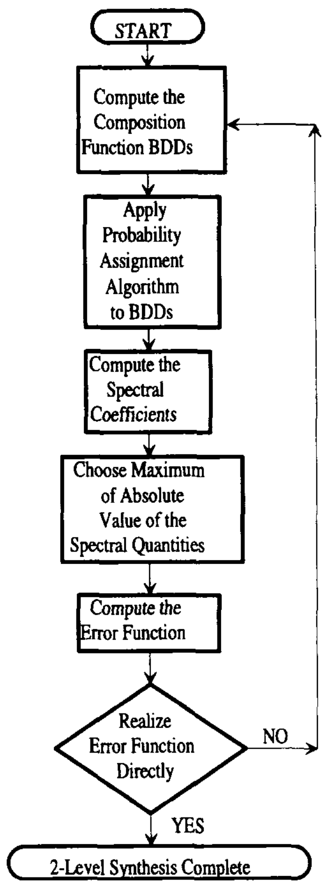

START
Compute the Composition Function BDDs
Apply Probability Assignment Algorithm to BDDs
Compute the Spectral Coefficients
Choose Maximum of Absolute Value of the Spectral Quantities
Compute the Error Function
Realize Error Function Directly
NO
YES
2-Level Synthesis Complete

Fig. 10. Flowchart of two-level synthesis technique.

overall structure of the proposed synthesis methodology. In fact, there are several ways that the use of spectral coefficients may be applied to the combinational logic synthesis problem. This paper will only outline the basic principles behind one of these approaches; later papers will present results from this method and others after they have been fully developed and implemented.

User supplied input consists of the BDD representation of the function to be synthesized and optionally, the maximum number of inputs per gate, N_{inp}, and preferences of the types of gates, $\{G_t\}$, to be used. The two optional parameters, N_{inp} and the set $\{G_t\}$ are used to determine the set of constituent functions, $\{F_c(x)\}$, that are used to construct the spectral vector.

The following list of procedural steps provides a detailed description of logic synthesis process depicted in Fig. 10.

Input Circuitry Iterative Logic Synthesis Algorithm:

1) Formulate the composition BDD's using $\{F_c(x)\}$ and $F(x)$.
2) Apply the Probability Assignment Algorithm to the composition BDD's.
3) Compute the spectral coefficients using (27).
4) Choose the largest (in magnitude) spectral coefficient.
5) Realize the function $F_c(x)$ that corresponds to the chosen coefficient in Sstep 4.
6) Compute the BDD representation of the error function, $e(x) = F_c(x) \oplus F(x)$.
7) If $e(x)$ indicates that there are w or fewer errors, go to Step 8. Otherwise iterate on the synthesis by going to Step 1 and use $e(x)$ as the next function to be synthesized.
8) Combine all the intermediate realizations of the various chosen $F_c(x)$ functions using the $\oplus$ operator and directly realize the function $e(x)$ for the remaining w or fewer errors.

This technique generates two-level tree-type circuits. For two-level realizations, each chosen $F_c(x)$ is realized in the first stage of the circuit with one multi-input logic gate. The second stage consists of a single combination gate that uses the outputs of all of the chosen constituent functions as its inputs. The circuits resulting from this synthesis technique are completely fan-out free (CFOF) and have the desirable property of requiring a set of test vectors equal to the number of primary circuit inputs to test all possible single stuck-at faults. As discussed in [13], the use of spectral design techniques for logic synthesis is known for the ability to produce easily tested circuits. The diagram in Fig. 11 indicates how the two-level circuit is constructed with each iteration.

The following theorem states the properties necessary to ensure the convergence of this synthesis algorithm.

Theorem 2: Any given Boolean function, $F(x)$, may be realized with the synthesis technique if the transformation matrix formed by using the output vectors of the constituent functions as row-vectors is of full-rank.

Proof: This proof is a statement that any N-vector can be produced as a combination of a subset of vectors from the set of vectors that are linearly independent over N-space. Each Boolean function to be realized is viewed as a N-vector with components from the binary field. The synthesis procedure described in the preceding text "chooses" a matrix row in each iteration (each row corresponds to a constituent function) to be "combined" with an appropriate combining operator. This process forms the output vector of the synthesized function as a combination of row vectors from the transformation matrix. Hence, if the transformation matrix contains at least N rows that span N-space, any function output vector can be realized by a finite number of combined transformation matrix rows. □

As with any synthesis method, this one can not realize some functions using a set of constituent functions that do not form a functionally complete set since the resulting transformation matrix will not be of full rank. For example, a function may not be realized if all constituent functions use only the AND operator.

E. Synthesis Method Example

In this section, an example of the synthesis technique is given. In the example, it is assumed that there are no restrictions on the number of inputs per constituent function and that only XOR, AND, and the AND-OR-INVERT (AOI) functions may be used.

Consider the realization of the function as shown in Fig. 6. Since a five input function is depicted, the set of constituent functions is chosen according to the constraints discussed above along with the knowledge that the function to be

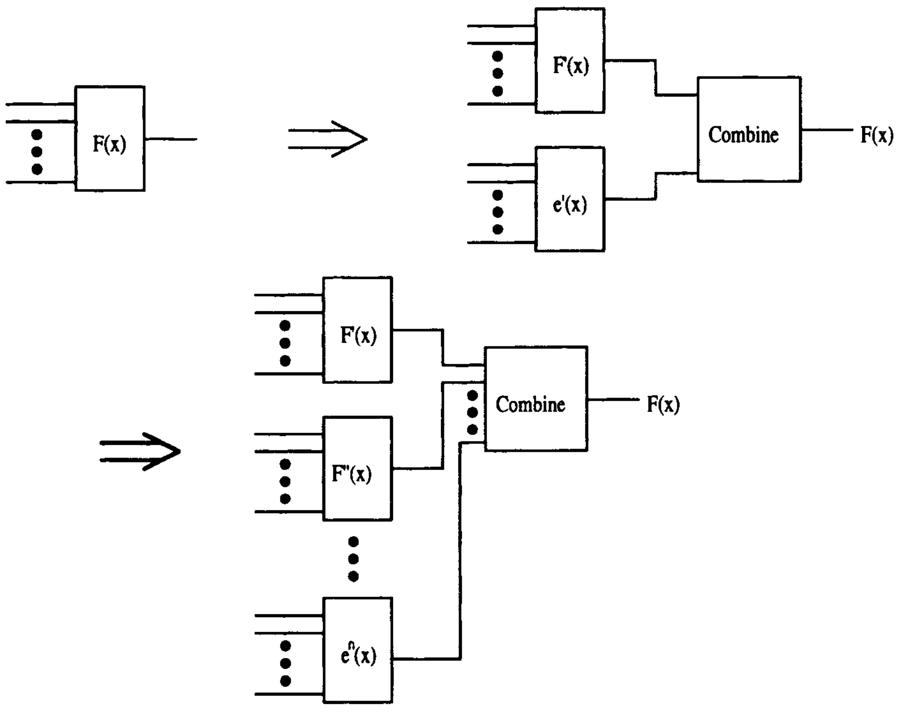


Fig. 11. Diagram of synthesis technique.

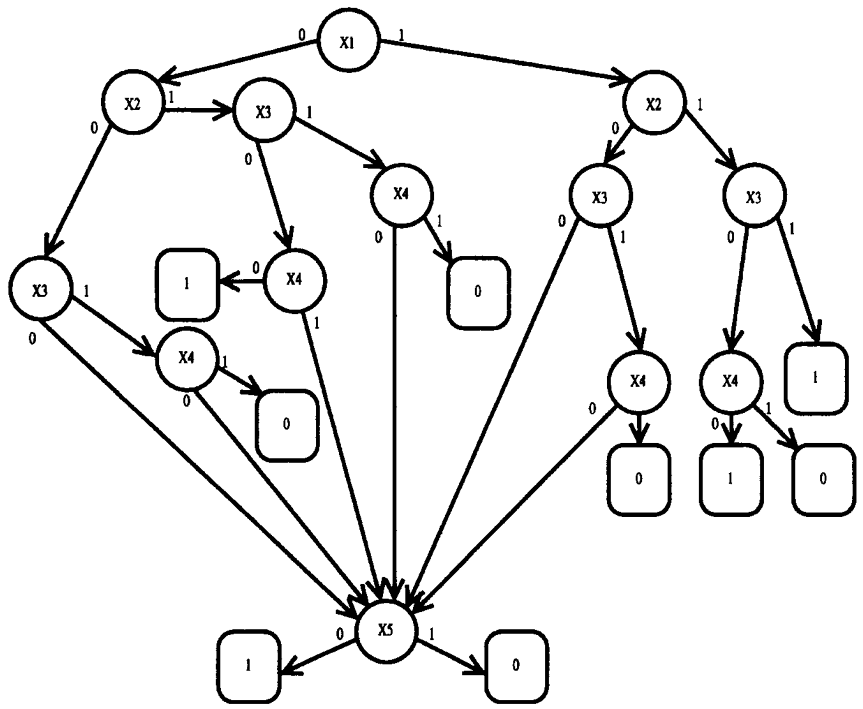


Fig. 12. BDD of function for synthesis example.

synthesized has five inputs. The set of constituent functions always includes functions that are equal to each component of the $\underline{x}$ vector and the $F_c(x) = 0$ function (i.e., the Chow parameters). These values are especially useful since they indicate the correlation between the output of the function with respect to each of its inputs.

First, the composition function BDD's are computed. Next, the spectral coefficients are computed using the probability assignment algorithm and (27).

In the first iteration, the maximum absolute valued spectral coefficient is 18 and corresponds to an AOI constituent function. Since the AOI constituent function, $F_c(x) = \overline{x_1x_2 + x_3x_4 + x_5}$, produced the largest spectral coefficient (in magnitude), it is chosen and the first portion of the circuit is realized as shown in Fig. 13.

The error function is computed with respect to an exclusive-OR operator since it is the most robust in terms of the possible operators available for providing the combining stage in the

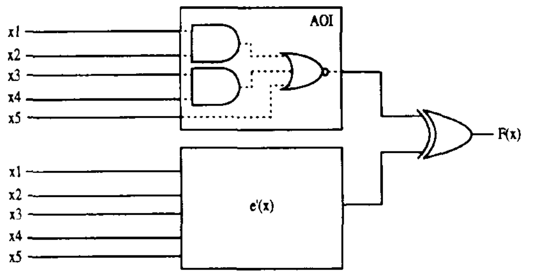


Fig. 13. First iteration of two-level synthesis of example "function."

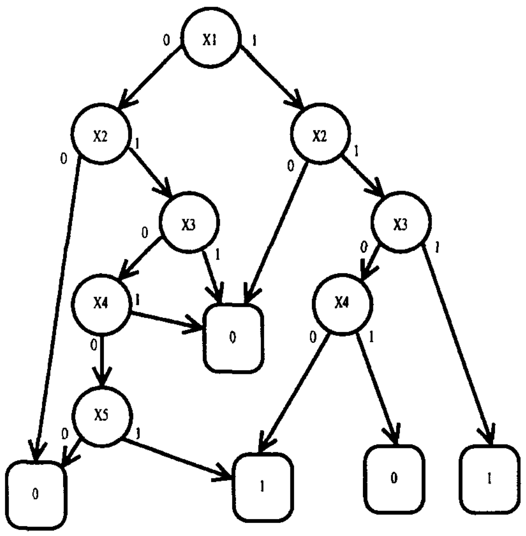


Fig. 14. BDD of the residual function after the first iteration.

circuit. This robustness is due the fact that an XOR can be used to change a 0 to 1 error as well as a 1 to 0 error. The following list describes the properties that determine which gate type may be used as an error operator. Since the XOR is capable of correcting all errors, it is used in this algorithm

1) XOR: $x \oplus 1 = \overline{x}$, errors may be $1 \rightarrow 0$ or $0 \rightarrow 1$.
2) AND: $x1 = x$ and $x0 = 0$, all errors must be $1 \rightarrow 0$.
3) OR: $x + 1 = 1$ and $x + 0 = x$, all errors must be $0 \rightarrow 1$.

After, the first iteration, the BDD of the error function is computed by using the *APPLY* algorithm with $F(x)$ and $x_1x_2 + x_3x_4 + x_5$ as inputs. The resulting BDD is shown in Fig. 14.

The synthesis algorithm requires 3 more iterations to completely realize the desired circuit. On the second iteration, the constituent function, $F_c(x) = x_1x_2x_3$, produces the largest spectral coefficient ($S_F[F_c(x)] = 26$) and is chosen as a term in the final circuit. The next iteration indicates $F_c(x) = x_2\overline{x}_3\overline{x}_4$ should be used since it has the highest valued spectral coefficient ($S_F[F_c(x)] = 22$). Finally, a single term remains, $\overline{x}_1x_2\overline{x}_3\overline{x}_4\overline{x}_5$, and it is chosen to directly realize the circuit. The complete circuit is given in Fig. 15.

V. Complexity Analysis

This section provides a discussion of the complexity of the various algorithms presented in this paper. First, the complexity of the probability assignment algorithm and spec-

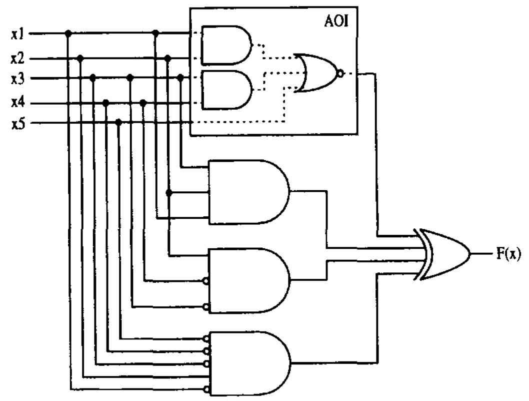


Fig. 15. Final circuit using the design process.

tral coefficient calculations are presented followed by the complexity of the synthesis approach.

A. Complexity of the Probability Assignment Algorithm

The BDD based algorithm for the computation of circuit output probabilities involves the traversal of a BDD from the input node to the terminal nodes. This enables the output probability of the function represented by the BDD to be computed with a complexity equal to $O(|E|)$, where $|E|$ is the number of edges or interconnections in the BDD.

B. Complexity of the Spectral Computation Algorithm

The necessary information for the calculation of the spectral coefficients is p_m. p_m can be conveniently determined as the sum of p_{m0} and p_{m1}. The values p_{m1} and p_{m0} are obtained with a complexity of $O(|E_{comp}|)$ where E_{comp} is the number of edges present in the BDD's of the two composition functions.

If the algorithm *APPLY* proposed in [4] is used to form the composition function BDD's, the resulting complexity is $O(|E_{f_c}|\,|E_f|)$, where $|E_{f_c}|$ is the number of edges in the BDD of the constituent function, $f_c(x)$, and $|E_f|$ 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. Since the bounding operation in the spectral coefficient calculation algorithm is the utilization of the *APPLY* algorithm to form the composition function BDD's, the total computational complexity is $O(|E_f| \times |E_{f_c}|)$.

C. Complexity of the Logic Synthesis Algorithm

In order to analyze the complexity of the synthesis algorithm it is convenient to consider the transformation matrix that could be used in lieu of the more efficient method for computing spectral coefficients provided in the preceding section. The matrix would consist of several row-vectors each of dimension 2^n. Thus, the computation of a single spectral coefficient would require 2^n scalar multiplications. Clearly, this is an exponentially bounded computation. However, if the output

TABLE II
A First Order Spectral Coefficient for each ISCAS85 Netlist

Circuit	Output	n	x_1	$\|BDD\|, f_c = x_1$			$S(f_c)/2^n$	
				f	$f \cdot f_c$	$f + f_c$	$f_c = 0$	$f_c = x_1$
$c432$	$421gat$	36	$4gat$	3970	3963	10	-7.068958×10^{-1}	-2.852917×10^{-1}
$c499$	$od0$	41	$id13$	3378	6307	6307	-9.921875×10^{-1}	-8.437500×10^{-1}
$c880$	$878gat$	45	$210gat$	3101	2930	3100	-2.779270×10^{-1}	2.411922×10^{-1}
$c1355$	$1324gat$	41	$92gat$	3378	6307	6307	-9.921875×10^{-1}	-8.437500×10^{-1}
$c1908$	66	33	952	71	63	12	7.690430×10^{-1}	4.923096×10^{-1}
$c2670$	308	122	69	219	219	216	9.338531×10^{-1}	3.890991×10^{-3}
$c3540$	409	49	213	36071	36071	36099	-2.162547×10^{-1}	-7.837453×10^{-1}
$c5315$	658	67	248	66552	43486	43485	-5.000000×10^{-1}	-7.827759×10^{-3}
$c6288$	$4946gat$	24	$273gat$	17058	14387	6722	-2.929688×10^{-3}	-2.441406×10^{-3}
$c7552$	418	194	150	466	466	1	-9.999999×10^{-1}	-1.257285×10^{-7}

circuit probability technique is used, the complexity is reduced from $O(2^n)$ to $O(|E_{f_c}|\,|E_f|)$ for each spectral coefficient.

Since there is a spectral coefficient computed for each member of set $\{F_c(x)\}$, the overall algorithm complexity will depend upon the set size. Suppose the constraint $N_{inp} = 2$ is imposed. This means that the resulting circuit must contain only 2-input logic gates. If all 16 possible 2-variable logic functions are present in the set $\{F_c(X)\}$, the total number of rows in the transformation matrix can be easily computed as shown in (37). This calculation simply considers all possible combinations of the primary inputs for a two-input gate. Since there are 16 total constituent functions, the number of combinations is multiplied by eight. The reason eight is used instead of 16 is because each member in the set of constituent functions has an inverse that is also in the set. Thus, by Lemma 4 the $S_f[f_c(x)]$ value for a particular $f_c(x)$ is simply the negative value of the spectral coefficient for $\overline{f_c(x)}$ so it is not necessary to compute the spectral coefficient for both

$$8\binom{n}{2} = 4(n)(n-1) = 4n^2 - 4n. \qquad (37)$$

Added to this value is $n + 1$ additional matrix rows for the computation of the Chow parameters [19], yielding a total number of rows equal to $4n^2 - 3n + 1 = O(n^2)$ in row-size complexity of the matrix.

Therefore, the total complexity of the one iteration of the synthesis algorithm is $O[n^2(|E_{f_c}|\,|E_f|)]$. A further observation is that an efficient variable ordering of a BDD can result in the number of edges being of order, $O(n)$, [15], [28]. Thus the total complexity of an iteration of the synthesis algorithm is $O(n^4)$ assuming efficient BDD orderings and equal complexity of the BDD's used to express $F(x)$ and each member in the set $\{F_c(X)\}$.

VI. Experimental Results from the Spectrum Computation Algorithm

The spectrum calculation algorithm was implemented using a popular OBDD package and by implementing the probability assignment algorithm using the C programming language. The probability assignment algorithm is similar to a "breadth-first search" approach except that instead of each node in the BDD being visited once, each traversal (or arc) in the graph is visited once. However, the complexity is still of the order of the number of nodes in the BDD since every nonterminal node has exactly two directed arcs leaving it.

The ISCAS85 benchmark circuits were used as inputs to this implementation to provide the experimental results. The netlists were parsed and an OBDD was created for each of them. Table II contains spectral coefficients for a selected output for each of the benchmark circuits. In addition to the 0th and 1st ordered coefficients, the sizes of the composite OBDD's are given thus providing a direct representation of the time complexity of this approach. The OBDD size columns are labeled $\|BDD\|$ and the number of nodes is given for the original circuit, f, and the composite functions, $f \cdot f_c$, and $f + f_c$.

Table II also contains the number of inputs, n, and the netlist label of the output that was used to create the OBDD. The spectral coefficients $S(f_c = 0)$ and $S(f_c = x_1)$ are scaled by 2^n for convenience thus they lie in the interval [-1, 1]. The two spectral coefficients are computed using the constituent functions, $f_c = 0$ and $f_c = x_1$. The specific netlist label for the input chosen as x_1 is also present in the table.

The set of spectral coefficients formed by using each primary input and the constant logic function, $f_c = 0$, are commonly referred to as the Chow parameters. This subset of the Walsh coefficients is particularly useful in many areas of spectral based CAD applications. Tables III and IV contain the complete set of Chow parameters for the benchmark circuit c432.

Many of the applications discussed in the preceding portion of this paper utilize constituent functions that are more complex than a single primary input. In order to demonstrate that this method is applicable for more complex and generalized constituent functions, coefficients were computed for various circuits and arbitrary constituent functions. Table V contains the constituent functions, the spectral coefficients, and the sizes of the resulting BDD's. Table VI gives the correspondence of the inputs x_i with the labeled inputs of the ISCAS85 circuits.

VII. Conclusion

In this paper, we developed a new efficient method for computing the spectrum of a Boolean function using BDD's. The theoretical relationships between circuit output probabilities and spectral coefficients of Boolean functions were developed. BDD's were used to calculate the output probabilities which in

TABLE III
THE FIRST 19 CHOW PARAMETERS FOR ISCAS85 CIRCUIT c432, OUTPUT 421gat

Constituent Function	$\|BDD\|$		Chow Parameter
f_c	$f \cdot f_c$	$f + f_c$	$S(f_c)/2^n$
0	3970	3970	-7.068958×10^{-1}
$x_1 = 4gat$	3963	10	-2.852917×10^{-1}
$x_2 = 1gat$	3589	3335	2.433660×10^{-1}
$x_3 = 11gat$	3779	3653	-2.318131×10^{-2}
$x_4 = 17gat$	3647	1860	3.022123×10^{-2}
$x_5 = 24gat$	3780	3655	-2.318131×10^{-2}
$x_6 = 30gat$	3650	1862	3.022123×10^{-2}
$x_7 = 37gat$	3780	3659	-2.318131×10^{-2}
$x_8 = 43gat$	3656	1866	3.022123×10^{-2}
$x_9 = 50gat$	3780	3667	-2.318131×10^{-2}
$x_{10} = 56gat$	3668	1874	3.022123×10^{-2}
$x_{11} = 63gat$	3780	3683	-2.318131×10^{-2}
$x_{12} = 69gat$	3692	1890	3.022123×10^{-2}
$x_{13} = 76gat$	3780	3715	-2.318131×10^{-2}
$x_{14} = 82gat$	3740	1922	3.022123×10^{-2}
$x_{15} = 89gat$	3780	3779	-2.318131×10^{-2}
$x_{16} = 95gat$	3836	1986	3.022123×10^{-2}
$x_{17} = 102gat$	3844	3969	-2.318131×10^{-2}
$x_{18} = 108gat$	3963	1985	3.022123×10^{-2}

TABLE IV
THE LAST 18 CHOW PARAMETERS FOR ISCAS85 CIRCUIT c432, OUTPUT 421gat

Constituent Function	$\|BDD\|$		Chow Parameter
f_c	$f \cdot f_c$	$f + f_c$	$S(f_c)/2^n$
$x_{19} = 8gat$	2947	2947	1.474875×10^{-1}
$x_{20} = 21gat$	3649	3649	1.422319×10^{-2}
$x_{21} = 34gat$	3648	3648	1.422319×10^{-2}
$x_{22} = 47gat$	3646	3646	1.422319×10^{-2}
$x_{23} = 60gat$	3642	3642	1.422319×10^{-2}
$x_{24} = 73gat$	3634	3634	1.422319×10^{-2}
$x_{25} = 86gat$	3618	3618	1.422319×10^{-2}
$x_{26} = 99gat$	3586	3586	1.422319×10^{-2}
$x_{27} = 112gat$	3522	3522	1.422319×10^{-2}
$x_{28} = 14gat$	3971	1668	7.755330×10^{-2}
$x_{29} = 27gat$	2818	3074	-7.505239×10^{-3}
$x_{30} = 40gat$	2818	3010	-7.505239×10^{-3}
$x_{31} = 53gat$	2850	3010	-7.505239×10^{-3}
$x_{32} = 66gat$	2898	3042	-7.505239×10^{-3}
$x_{33} = 79gat$	2954	3090	-7.505239×10^{-3}
$x_{34} = 92gat$	3014	3146	-7.505239×10^{-3}
$x_{35} = 105gat$	3076	3206	-7.505239×10^{-3}
$x_{36} = 115gat$	3139	3268	-7.505239×10^{-3}

TABLE V
SPECTRAL COEFFICIENTS FOR VARIOUS CONSTITUENT FUNCTIONS AND ISCAS85 CIRCUITS

Constituent	ISCAS85	$\|BDD\|$		Spectral
Function	Circuit	$f \cdot f_c$	$f + f_c$	Coefficient
$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$	c432	3884	3821	4.859995×10^{-4}
	c880	3045	3022	1.455054×10^{-2}
	c7552	214	214	0.000000×10^{0}
$\bar{x}_1\bar{x}_2x_5 + x_1\bar{x}_5 + x_2\bar{x}_5$	c432	3659	3782	2.318131×10^{-2}
	c880	3039	3020	-5.577102×10^{-2}
	c7552	211	211	2.351379×10^{-2}
$\bar{x}_1 \oplus x_4$	c432	1854	3654	-2.240873×10^{-2}
	c880	3038	3010	4.875052×10^{-2}
	c7552	208	208	1.250000×10^{-1}

turn were used to compute the individual spectral coefficients. The complexity of the new technique has been analyzed and the method has been implemented with experimental results given.

TABLE VI
CORRESPONDENCE OF LABELED INPUTS WITH THOSE IN THE CONSTITUENT FUNCTIONS

Circuit	Output	x_1	x_2	x_3	x_4	x_5
c432	$421gat$	$4gat$	$1gat$	$11gat$	$17gat$	$24gat$
c880	$878gat$	$210gat$	$268gat$	$219gat$	$8gat$	$138gat$
c7552	276	4528	1496	38	1492	1486

The capability to compute individual spectral coefficients expands the realm of applications of spectral-based CAD techniques. As an example we outline a synthesis approach that iteratively realizes various levels of the circuit based upon the spectral coefficients of the partially realized subcircuit. This type of approach for logic synthesis can be custom-tuned for the inclusion of the XOR's and various optimization parameters such as delay, area, testability, and power.

ACKNOWLEDGMENT

The authors would like to thank D. Long of AT&T Laboratories for making his BDD library package available and particularly for providing the program that converts the ISCAS85 netlists to OBDD's.

REFERENCES

[1] S. B. Akers, "Binary decision diagrams," *IEEE Trans. Comput.*, vol. C-27, pp. 509–516, June, 1978.

[2] R. L. Ashenhurst, "The decomposition of switching functions," in *Proc. Int. Symp. Theory Switching*, Apr. 1957, pp. 74–116.

[3] D. Bryan, "The ISCAS85 benchmark circuits and netlist format," *On-Line Documentation*, 1988.

[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, pp. 667–691, Aug. 1986.

[5] C. K. Chow, "On the characterization of threshold functions," *IEEE Special Pub. S.134*, pp. 34–38, 1961.

[6] E. M. Clarke, K. L. McMillian, X. Zhao, and M. Fujita, "Spectral transforms for extremely large Boolean function," in *Proc. IFIP WG 10.5, Workshop Appl. Reed-Muller Expansion in Circuit Design*, Sept. 1993, pp. 86–90.

[7] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transformations for large Boolean functions with applications to technology mapping," in *Proc. ACM/IEEE Design Automat. Conf.*, pp. 54–60, 1993.

[8] E. M. Clarke, X. Zhao, M. Fujita, Y. Matsunaga, R. McGeer, and J. Yan, "Fast Walsh transform computation with binary decision diagram," in *Proc. IFIP WG 10.5 Workshop Appl. Reed-Muller Expansion in Circuit Design*, Sept. 1993, pp. 82–85.

[9] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math Computat.*, vol. 19, pp. 297–301, 1965.

[10] T. Damarla, "Generalized transforms for multiple valued circuits and their fault detection," *IEEE Trans. Comput.*, vol. 41, pp. 1101–1109, Sept. 1992.

[11] M. Davio, J.-P. Deschamps, and A. Thayse, *Discrete and Switching Functions*. New York: McGraw-Hill, 1978.

[12] C. R. Edwards, "The application of the Rademacher–Walsh transform to Boolean function classification and threshold logic synthesis," *IEEE Trans. Comput.,*, vol. C-24, pp. 48–62, 1975.

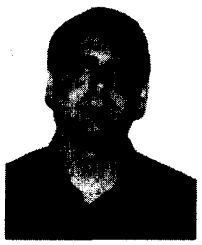
[13] ——, "The design of easily tested circuits using mapping and spectral techniques," *Radio Electron. Eng.*, vol. 47, no. 7, pp. 321–342, 1977.

[14] B. J. Falkowski, I. Schafer, and M. A. Perkowski, "Calculation of the Rademacher–Walsh spectrum from a reduced representation of Boolean functions," in *Proc. Europ. Design Automat. Conf.*, Sept. 1992, pp. 181–186.

[15] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluation and improvements of Boolean comparison method based on binary decision diagrams," *Proc. ICCAD*, pp. 2–5, 1988.

[16] M. Fujita, J. Yang, E. M. Clarke, X. Zhao, and P. McGeer, "Fast spectrum computation for logic functions using binary decision diagrams," in *Proc. Int. Conf. Circuits Syst.*, 1994.

[17] D. Green, *Modern Logic Design.* Reading, MA: Addison-Wesley, 1986.
[18] T. C. Hsiao and S. C. Seth, "An analysis of the use of Rademacher–Walsh spectrum in compact testing," *IEEE Trans. Comput.*, vol. C-33, pp. 934–937, Oct. 1984.
[19] S. L Hurst, "The application of chow parameters and Rademacher–Walsh matrices in the synthesis of binary functions," *Comput. J.*, vol. 16, no. 2, 1973.
[20] S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic.* Orlando, FL: Academic Press, 1985.
[21] J. Jain, J. Bitner, D. S. Fussel, and J. A. Abraham, "Probabilistic design verification," Tech. Rep., Univ. Texas, Austin, UT-CERC-TR-JAA91-01, Apr. 1991.
[22] M. G. Karpovsky, *Finite Orthogonal Series in the Design of Digital Devices.* New York: Wiley, 1976.
[23] S. K. Kumar and M. A. Breuer, "Probabilistic aspects of Boolean switching functions via a new transform," *Journal ACM*, vol. 28, no. 3, pp. 502–520, July 1981.
[24] R. Lechner, "Harmonic analysis of switching functions," *In Recent Developments in Switching Theory*, pp. 121–228, 1971.
[25] C. Y. Lee, "Representation of switching circuits by binary-decision programs," *Bell Syst. Tech. J.*, vol. 38, pp. 985–999, July 1959.
[26] A. M. Lloyd, "A consideration of orthogonal matrices, other than the Rademacher–Walsh types, for the synthesis of digital networks," *J. Electron.*, vol. 47, no. 3, pp. 205–212, 1979.
[27] M. Mano, *Digital Design.* Englewood Cliffs, NJ: Prentice-Hall, 1984.
[28] S. Milak, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic verification using binary decision diagrams in a logic synthesis environment," *Proc. ICCAD*, pp. 6–9, 1988.
[29] D. M. Miller and J. C. Muzio, "Spectral fault signatures for single stuck-at faults in combinational networks," *IEEE Trans. Comput.*, vol. C-33, pp. 765–768, Aug. 1984.
[30] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Trans. Comput.*, vol. C-24, pp. 668–670, June 1975.
[31] M. A. Perkowski, M. Driscoll, J. Liu, D. Smith, J. Brown *et al.*, "Integration of logic synthesis and high-level synthesis into the DIADES design automation system," in *Proc. 22nd IEEE Int. Symp. Circuits Syst.*, 1989, pp. 748–751.
[32] J. L. Shanks, "Computation of the Fast Walsh–Fourier transform," *IEEE Trans. Comput.*, vol. C-18, pp. 457–459, May 1969.
[33] M. Stanković, Z. Tošić, and S. Nikolić, "Synthesis of Maitra cacades by means of spectral coefficients," *IEE Proc.*, vol. 130, Pt. E, no. 4, pp. 101–108, July 1983.
[34] A. K. Susskind, "Testing by verifying Walsh coefficients," *IEEE Trans. Comput.*, vol. C-32, pp. 198–201, Feb. 1983.
[35] M. A. Thornton and V. S. S. Nair, "The analysis and classification of parity functions using a small subset of Walsh coefficient," *1995 IEEE Int. Symp. Multiple Valued Logic*, to appear.
[36] ——, "The computation of a Boolean function spectrum using a structural input," Tech. Rep., 94-CSE-40, 1994.
[37] ——, "Applications and efficient computation of spectral coefficients for digital logic," Tech. Rep., 94-CSE-13, Feb. 1994.
[38] ——, "A numerical method for Reed-Muller circuit synthesis," in *Proc. IFIP WG 10.5 Workshop Appl. Reed-Muller Expansion in Circuit Design*, Sept. 1993, pp. 69–74.
[39] ——, "An iterative combinational logic synthesis technique using spectral information," in *Proc. Europ. Design Automat. Conf.*, Sept. 1993, pp. 358–363.
[40] V. M. Tokmen, "Disjoint decomposability of multiple valued functions by spectral means," in *Proc. IEEE 10th Int. Symp. Mult. Valued Logic*, 1980, pp. 88–93.
[41] D. Varma and E. A. Trachtenberg, "Design automation tools for efficient implementation of logic functions by decomposition," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 901–916, Aug. 1989.



Mitchell A. Thornton (S'81–M'85) received the B.S. in electrical engineering and from Oklahoma State University, Stillwater, the M.S. degree in electrical engineering from the University of Texas, Arlington, the M.S. degree in computer science and the Ph.D. degree in computer engineering from Southern Methodist University, Dallas, TX, in 1985, 1991, 1993, and 1995, respectively.

From 1985 to 1990, he was employed at E-Systems, Inc., where he left as a Senior Electronic Systems Engineer. Currently, he is an Assistant Professor in the Computer Systems Engineering Department, University of Arkansas, in Fayetteville, AR. His research interests include logic synthesis, design verification, and computer arithmetic.

Mr. Thornton is a registered Professional Engineer in Texas and is a member of Eta Kappa Nu.

V. S. S. Nair (M'95) received the B.Sc.Engg. in electronics and communications from the University of Kerala. He received the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Illinois, Urbana, in 1988 and 1990, respectively.

Currently, he is an Assistant Professor in the Computer Science and Engineering Department at Southern Methodist University, Dallas, TX, where he holds a J. Lindsey Embrey Trustee Professorship in Engineering. His research interests include fault-tolerant computing and communication, VLSI systems, and software engineering.

Dr. Nair is a member of the ACM.