SPECTRAL BASED NUMERICAL METHODS
FOR COMBINATIONAL LOGIC SYNTHESIS

Approved by:

Dr. V. S. S. Nair

Dr. Dan I. Moldovan

Dr. Eric C. Lin

Dr. John D. Provence

Dr. Carlos E. Davila

Dr. Mandyam D. Srinath

SPECTRAL BASED NUMERICAL METHODS
FOR COMBINATIONAL LOGIC SYNTHESIS

A Dissertation Presented to the Graduate Faculty of the

School of Engineering and Applied Science

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Computer Engineering

by

Mitchell Aaron Thornton

(B.S., Oklahoma State University, 1985)
(M.S., University of Texas at Arlington, 1990)
(M.S., Southern Methodist University, 1993)

August 4, 1995

ACKNOWLEDGMENTS

There are many people who have provided support and encouragement during the time I pursued my graduate studies. I would like to acknowledge all of my Professors and in particular my advisor, V. S. S. Nair. His optimism and encouragement provided me with considerable motivation for the completion of this work, but above all, I value his friendship. I would also like to thank all the other members of my advising committee for their valuable comments and suggestions which helped to improve this dissertation considerably.

My family provided support for this work in many ways. The sacrifices that my wife, Misty, and my son, Micah, endured during the time period this work was undertaken are gratefully acknowledged and a testament to their support for me. The instillation of the value of education and quest for knowledge in my life must be attributed to my parents, Dennis and Mary Ann Thornton. They always encouraged me to achieve my dreams and this work is a product of that encouragement.

Finally, I would like to acknowledge the financial support provided by Southern Methodist University in the form of a teaching assistantship and that provided by a grant from the National Science Foundation under contract MIP-9410822. Without this support, this dissertation would not have been possible.

Thornton, Mitchell Aaron

B.S., Oklahoma State University, 1985
M.S., University of Texas at Arlington, 1990
M.S., Southern Methodist University, 1993

Spectral Based Numerical Methods
for Combinational Logic Synthesis

Advisor: Assistant Professor V. S. Sukumaran Nair

Doctor of Philosophy degree conferred August 4, 1995

Dissertation completed May 17, 1995

Automated computer-aided design (CAD) methods for the synthesis of digital logic circuits are typically employed to meet the demands imposed on todays' chip designers. As modern technology matures, the size and functionality of integrated circuits tends to increase proportionally. The use of CAD techniques allows the designer to expend more effort in the conceptual and behavioral specification portion of the design process since the tedious task of translating the behavioral description of a circuit into a structural one is handled by the automated synthesis system. Unfortunately, the state of modern automated synthesis technology is still in its infancy.

The development of CAD systems that are more mature and therefore capable of solving a more general class of problems is an area of research whose results are responsible for the enormous growth the electronic design automation (EDA) industry has recently enjoyed. The most common approach in this research area is to expand upon the principles used in current automated design tools which are typically rule based systems. The research discussed in this dissertation describes the results of an investigation into the use of spectral based numerical techniques to perform digital logic synthesis.

The first phase of this research validates the approach of using spectral quantities by describing a new method for their calculation. In the past, spectral methods were judged to be inappropriate for automated logic synthesis systems since the necessary calculations were far too time consuming to be exploited in an actual CAD system. With the advent of an efficient method for computing the spectra, numerically based techniques provide a viable alternative to the methods that are commonly employed in the commercial CAD systems in use today.

The second phase of this research describes three ways that numerical methods may be used in the implementation of automated logic synthesis systems. Altogether new approaches are developed as well as methods to augment existing CAD systems. The experimental results of these approaches show that numerical methods are capable of solving many problems in the area of logic synthesis in an efficient and timely manner. Further, the results show that the spectral approach offers superior synthesis solutions for certain classes of circuits.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DEDICATION

To Misty Dawn

CHAPTER 1

INTRODUCTION

Computer aided design (CAD) is now the standard approach for almost all aspects of the design of digital electronic circuitry. The huge advances in very large scale integration (VLSI) production techniques have provided motivation for the development and use of automated design systems in order to handle the increasing complexity in modern designs. Although CAD methods are used in the design of most integrated circuits produced today, the underlying principles that form a basis for the tools are still evolving. One example is the transition from schematic entry systems which were popular in the early 1980's to the use of hardware description languages (HDLs) that are common in todays' design environment.

The overall objective of a CAD tool is the transformation of a high level specification, subject to given design constraints, into a description of circuitry suitable for production. This transformation is usually divided into a number of intermediate steps. One of these steps is referred to as the logic synthesis step. This portion of the CAD tool is the focus of the research results described in this dissertation.

1.1. Modern Logic Synthesis Techniques

Most logic synthesis tools generally decompose the process into 3 subsequent steps which are illustrated in Figure 1.1. The logic synthesis system input is a description of the function to be synthesized. It usually is in the form of a Boolean equation, set of covering cubes, or, an initial structural representation. The first step minimizes the number of covering cubes, and ultimately the area required for the realization. The next two steps are often referred to collectively as the technology mapping portion. The technology mapper first decomposes the minimized description into a set of small interconnected subfunctions, and then it maps them to a corresponding set of library cells. This latter step is usually called Boolean matching or library binding.

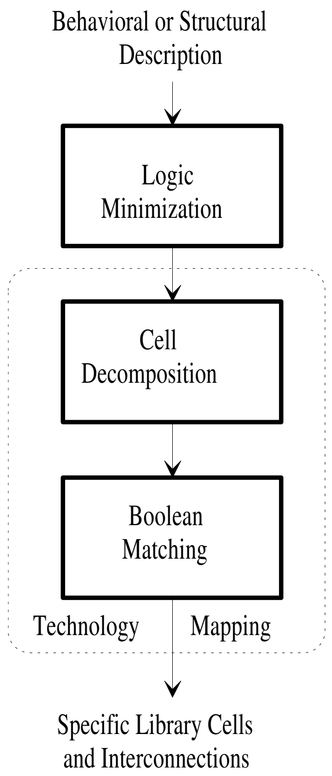


Figure 1.1. Processing flow of a typical logic synthesis tool

To date, the most successful logic synthesis systems have employed heuristics that guarantee a pseudo-optimal result since the logic synthesis problem is NP-hard [1]. These systems generally use algebraic manipulation of Boolean equations, or, graph algorithms applied to an initial circuit structure. Since these methods are based on heuristics in order to make the problem tractable, there is much room for improvement. In particular, as optimization constraints change some approaches may offer advantages over others. For instance, a system developed to optimize the required substrate area may yield circuits with large critical paths, and hence, poor timing characteristics.

1.1.1. Synthesis Tools Using Don't Care Sets

One of the most popular and commonly used tools for area optimization is $ESPRESSO$ [2] [3]. This tool is based on the Quine-McCluskey (QM) method for logic minimization [4] [5]. The QM algorithm is well suited for computer implementation, however it has an exponential complexity with respect to the number of input variables. By clever exploitation of the don't care set of cubes, a set of rules can be obtained that may be applied to the function under consideration. The application of these rules usually allows for a drastic reduction in the computational requirements as compared to the QM method and results in a good two-level minimization.

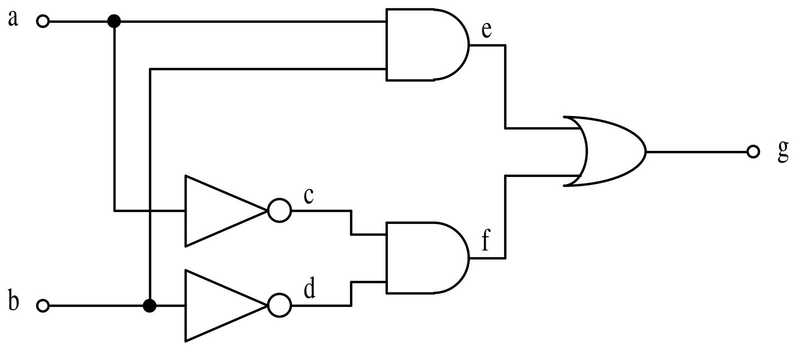
The $ESPRESSO$ algorithm is comprised of a series of functions that employ various heuristics. The first function, $EXPAND$, replaces each original cube by a set of all distinct cubes of one dimension higher provided that the replacement cubes do not intersect the complement of the function. Next, the procedure $IRREDUNDANT$ is applied. $IRREDUNDANT$ attempts to separate the the relatively essential cubes from all others. After the application of $IRREDUNDANT$, the $REDUCE$ operation is invoked which attempts to determine as small a set as possible of the relatively essential cubes that cover the function. These operations and others are applied repeatedly until no further minimization is achieved.

The minimization of area can be enhanced further by representing a logic function as a multi-level circuit instead of the two-level result such as that supplied by *ESPRESSO*. This enhancement is at the cost of increasing the critical path length since the two-level circuit always has a path length no larger than 2 multi-input logic gates. A popular multi-level logic synthesis algorithm is *MIS* [6]. This technique uses an initial representation of the structure of the circuit as input. This representation is "flattened" into an equivalent two-level representation where *ESPRESSO*-like heuristics are applied. Next, the reduced two-level representation is decomposed into small subcircuits which are subsequently matched to a given cell library [7].

1.1.2. Synthesis Tools Using Permissible Functions

The basic paradigm used in the creation of *ESPRESSO* and *MIS* was the exploitation of don't care sets to provide area minimization. Another popular synthesis tool paradigm is the use of the concept of permissible functions. The first synthesis tools to use this concept were the *TRANSDUCTION* and *SYLON* systems [8] [9]. Permissible functions have also been used recently to develop synthesis tools by other researchers [10] [11].

In order to outline the concept of permissible functions, consider the circuit shown in Figure 1.2. In the circuit in Figure 1.2 each internal node as well as the primary inputs and output are labeled with a unique variable. The possible logic levels for each of these nodes is given in the accompanying table. Each internal node can be viewed as a logic subcircuit. With this viewpoint, a permissible function



a	b	c	d	e	f	g
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

Figure 1.2. Logic circuit and truth table before permissible function substitution

is any function that has the same output vector as that given in the table. Using this definition, the permissible function for node f is $\{1, 0, 0, 0\}$. Therefore, a two-input NOR gate is a permissible function for f, resulting in an equivalent circuit as shown in Figure 1.3. In more complex circuits, the intermediate output vectors generally include don't care terms allowing more flexibility in the choice of permissible functions.

1.1.3. Other Approaches for Logic Synthesis

Although the two most popular logic synthesis paradigms have been described above with some of the specific tools referenced, other less popular synthesis methodologies have been developed as well. Since the logic synthesis problem can be viewed

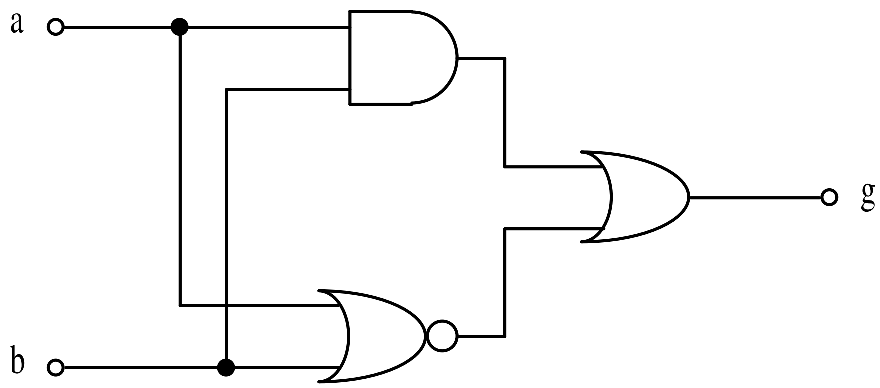


Figure 1.3. Logic circuit after permissible function substitution

as an optimization problem, various researchers have attempted to apply linear programming methods, notably that in [12].

Another approach where considerable research has been accomplished is the use of spectral methods for digital logic. These methods usually suffer from high complexity restricting their usefulness to very small circuits. The spectral approach is the subject of this dissertation since the research described here has yielded an efficient method for the computation of Boolean spectra using numerical techniques.

1.2. Spectral Based Synthesis Methods

The great success of spectral techniques for signal analysis and linear systems analysis and design prompted researchers to look for ways to apply these methods to digital systems. The pioneering work of Karpovsky [13] and Lechner [14] are generally regarded as the basis of subsequent work in this field.

The principles of spectral methods have been applied to many areas in digital systems engineering. Some of these include synthesis [13] [15] [16] [17] [18] [19], partitioning techniques [13] [14][20] [21] [22], testing [23] [24] [25] [26], function classification [16] [27], and others. It has been shown that certain problems such as disjoint decomposition [20] [21] and function classification [16] cannot be solved with less complexity in the Boolean domain than in the spectral domain.

Spectral methods have not enjoyed wide acceptance due to the large complexity required to compute the spectra using past methods. Even when fast spectral computation methods such as those developed by Cooley and Tukey to compute the discrete Fourier and later the more applicable Walsh transforms are applied [28] [29], the calculation of the spectrum of a Boolean function still imposes exponential complexity. This large complexity arises because the $O(NlgN)$ complexity of the Cooley-Tukey method translates to $O(n2^n)$ complexity for digital circuits since $N = 2^n$ where n is the number of circuit inputs.

Most of the transformation matrices used in the past were orthogonal matrices such as those constructed using Walsh functions, or, the Reed-Muller type. This allowed the resulting spectrum to be unique for a given function, and easy translation from the spectral to Boolean domains. Also, it allowed the Cooley-Tukey methods to be applied resulting in a modest savings in the computations.

By definition, the spectrum of a Boolean function is obtained by multiplying a transformation matrix by the function's output vector [16]. The following example illustrates an example calculation of a spectrum of a Boolean function whose truth table is given in Figure 1.4. The Rademacher-Walsh spectrum of this function is obtained by using a transformation matrix whose rows are composed of the Walsh functions with a Rademacher ordering. For transformations using Walsh matrices, the logic "0" values are represented using the integer "1" and the logic "1" values are represented by the integer "1" The spectrum is computed as shown in Equation 1.

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

Figure 1.4. Truth table of the function to be transformed using the RW matrix

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -4 \\ 4 \\ 0 \\ 0 \\ 4 \\ 4 \\ 0 \end{bmatrix} \tag{1}$$

The usefulness of the spectral data is that it provides global information about the nature of the function with respect to its inputs as opposed to the local information provided by a truth table. A notable advantage of spectral methods is that information regarding the XOR operation is readily available which is often not the case when methods such as those described in the preceding subsection are applied.

1.3. Impact and Contributions of this Research

The results described in this work renew the interest in the use of spectral methods since an efficient technique for the computation of the spectra is developed. By reducing the average complexity for computing the spectral coefficients, new techniques for logic synthesis are formulated. Past spectral based methodologies required the entire spectrum to be used. Since an individual coefficient may be computed very efficiently with the method presented here, emphasis is given to the development of methods that require only small subsets of coefficients.

The efficient spectral calculation technique is also very general in that any arbitrary transformation matrix may be used. This allows transformation matrices other than those used in the past to be investigated. The alternative matrices need not be orthogonal, or even square, since Cooley-Tukey algorithms are not used to generate the spectrum. Also, the freedom to use arbitrary transformation matrices allows problems such as Boolean matching to be customized to a specific transformation matrix corresponding to a given cell library.

The use of spectral coefficients in logic synthesis systems allow complex logic cells to be included in the resulting design. In particular, the detection and inclusion of the XOR operator is usually difficult to achieve since most synthesis algorithms rely upon algebraic manipulations or graphical techniques. The properties of the XOR operation are easily detected and exploited when spectral methods are applied. This result is applicable to standard cell as well as field programmable gate array (FPGA) design solutions since many manufacturers are producing FPGAs with basic logic blocks that include the XOR gate.

1.4. Organization

The remainder of this dissertation is organized as follows. Chapter 2 provides a discussion of the development of the efficient spectrum calculation technique. Chapters 3 and 4 present new algorithms for the synthesis of multi-level circuits that are particularly well suited to the efficient computation method. Chapter 5 presents the results that allow the efficient spectral computation method to be extended to the case of the Reed-Muller (RM) form.

Chapter 2 is devoted to the development of the efficient method for the computation of the spectral coefficients. This work provides the necessary background and results that allow the synthesis methods developed in later chapters to be practical. In addition to the development of the technique, a brief review of the relevant properties of binary decision diagrams (BDDs) is presented since the BDD representation is crucial to the technique.

In Chapter 3, a spectral based algorithm is described that uses the paradigm of subfunction independence. This technique attempts to realize a circuit by successively computing subcircuits with as much degeneracy as possible thereby allowing the subcircuits to be minimized by the disregard of redundant inputs. In addition, this method uses a small set of spectral coefficients and is capable of strictly enforcing specified timing constraints.

In Chapter 4, an alternative technique is described using a maximum correlation paradigm. The maximum correlation paradigm is useful because it allows the transformation matrix to be customized for a particular cell library. The customization results in combining the logic minimization and library binding tasks into a single operation.

Chapter 5 describes how the efficient calculations can be applied to the realization of the Reed-Muller (RM) form of digital circuits. Classic RM circuit realization using spectral methods requires the use of modulo-2 arithmetic for the computation of the coefficients. By developing the algebraic relation between modulo-2 and real valued arithmetic, it is shown that the efficient spectral calculation technique may be applied to Reed-Muller circuit synthesis.

Finally, conclusions and future research areas are given in Chapter 6. The future research areas include extensions to the methods developed here in addition to the use of spectral computations in other areas of CAD of digital systems.

CHAPTER 2

EFFICIENT CALCULATION OF THE SPECTRUM OF A BOOLEAN FUNCTION

An efficient algorithm for the calculation of the spectrum of a Boolean function is developed and presented in this chapter. In addition to providing efficiency in the computations, the method is very general with no restrictions on the form of the transformation matrix. Unlike other methods, the transformation matrix is not required to be recursively defined or sparse. The efficiency of the technique is due to the fact that it has complexity of the order of the number of edges in a binary decision diagram (BDD) [30] [31].

Although new approaches for efficient spectral coefficient calculation schemes have recently been proposed by other researchers, the method presented here has many advantages in comparison. In particular, a recent method has been proposed that utilizes "integer valued" BDDs [32] [33] [34]. 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. Even for transformation matrices that are sparse or capable of being defined recursively, this method can still generate extremely large integer valued BDDs if the spectrum contains many dissimilar values. The method proposed here compares favorably with this approach since a single spectral coefficient is computed in time proportional to the number of edges in a BDD. Further, this method does not require the transformation matrix to be recursively defined or sparse to preserve the efficiency of the computation.

Another recently proposed methodology allows for the computation of transform coefficients directly from a representation of a Boolean function as a set of disjoint cubes [35] [22]. Unfortunately, as the number of inputs to the Boolean function grows, the corresponding set of disjoint cubes can become extremely large. The method presented here has the advantage that the function to be transformed can be represented in very compact manner (as a BDD) and does not require a large set of product terms.

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 [36], analysis [37], and verification [38]. This chapter discusses the use of output probabilities to compute spectral coefficients in an efficient manner. To that end, a new algorithm to compute circuit output probabilities is developed.

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 $O(\|E\|)$ complexity where $\|E\|$ represents the number of edges in a BDD.

2.1. The Concept of Constituent Functions

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 the row vector in the transformation matrix 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. As an example, the transformation matrix in Figure 2.1 has the corresponding constituent functions to the left.

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

Figure 2.1. Transformation matrix and corresponding constituent functions

Throughout the remainder of the dissertation, the following notation and definitions will be used to describe the development and implementation of the spectral algorithms.

- n is the number of input variables of a Boolean function.
- 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".

- 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 $S_f[f_c(x)] = N_m - N_{mm}$ [39].

- $R_f(x)$ 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:

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

The spectral coefficient values may be interpreted as correlation measures between the constituent functions and the transformed function. The actual relationship between a spectral coefficient and a coefficient of correlation is given in the following lemma.

Lemma 1 *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 provided by the definition, the coefficient of correlation is given by Equation 2 as:

$$C = \frac{1}{2^n} \sum_{i=0}^{n-1} [R_f(m_i) \times R_{fc}(m_i)] \tag{3}$$

Where, m_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 $\sum_{i=0}^{n-1}[R_{f(m_i)} \times R_{f_c(m_i)}]$ 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 2:

$$C = \frac{1}{2^n} S_f[f_c(x)] \tag{4}$$

Hence, $S_f[f_c(x)]$ is directly proportional to C with a constant proportionality coefficient of $1/2^n$. □

Similar results can be proven for other definitions of spectral coefficients. For instance, the Reed-Muller transform [40] [41] can be defined using a vector of values where each component is the number of matching logic "1" outputs (calculated as $\wp\{f \cdot f_c\} \times 2^n$) between the function to be transformed and a constituent function.

2.2. Output Probabilities of Boolean Circuits

This section discusses the computations of circuit output probabilities by briefly reviewing two methods used to compute output probability expressions (OPEs) and then by directly computing a circuit output probability using BDDs. Also, an example of a BDD for a specific logic function is presented.

The OPE of a combinational logic circuit is an algebraic relation 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 [36].

In [36], 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. Each product term 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 set of inputs. This constraint serves to preserve statistical independence. The rules in Table 2.1 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 Boolean function.

Table 2.1.–Rules for transforming Boolean operations to probability expressions

Boolean Operation	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

A more efficient algorithm for the computation of the OPE of a Boolean function is also given in [36]. This method requires the function to be represented as a logic diagram. In this formulation, each primary input, each internal interconnection, and the output is assigned a unique variable name. Using the rules in Table 2.1, 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.

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 in a behavioral or functional form 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 [30] [31] [42]. In [31], some restrictions were placed upon the formation of BDDs that allowed several efficient algorithms to be defined for their manipulation. Specifically, it was required that the BDDs be formed as "ordered binary decision diagrams" (OBDDs). This means that for any given path in the BDD, a graph node corresponding to a particular input is only encountered once, and, that subsequent input nodes have an index value greater than their predecessors. The OBDD form is used in this development to ensure that each input node in a given path is only encountered once thus ensuring statistical independence. As an example of an OBDD, consider the function defined in Equation 5.

$$f(x) = x_1 x_3 \overline{x}_6 + x_1 \overline{x}_3 x_4 \overline{x}_6 + x_1 \overline{x}_3 \overline{x}_4 \overline{x}_5 + \overline{x}_1 x_2 x_4 \overline{x}_6 + \overline{x}_1 x_2 \overline{x}_4 \overline{x}_5 + x_1 \overline{x}_2 \overline{x}_5 \tag{5}$$

This function would require a truth table with 2^6 entries to be completely specified. The BDD representation of this function in Figure 2.2 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, the OBDD form of BDD as defined in [31] is used some of the BDD algorithms cited there are occasionally referenced as well.

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

Lemma 2 *For any one particular combination of primary input values, 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. $\square$

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

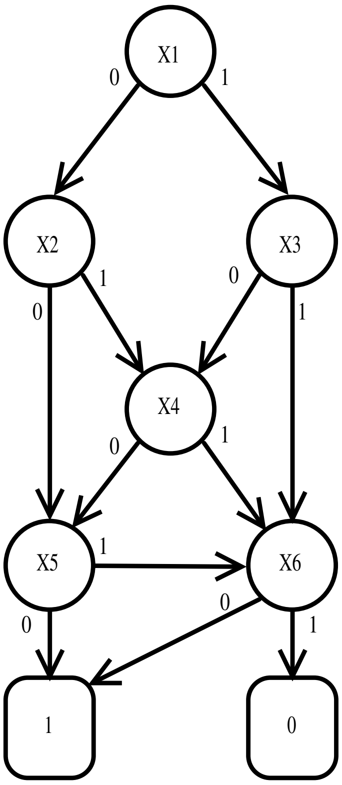


Figure 2.2. Example of a binary decision diagram

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

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 3 *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 2 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 $O(\|E\|)$, where $\|E\|$ 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.

2.2.1. OPE Calculation Using Logic Equations as Input

As an example of the OPE calculation method using logic equations as input, consider the function defined by the truth table in Figure 2.3.

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 2.3. Truth table for example OPE calculation using a logic diagram

The canonical SOP form for this function is given in Equation 6.

$$f(x) = \overline{x}_3 x_2 x_1 + x_3 x_2 \overline{x}_1 + x_3 x_2 x_1 \tag{6}$$

The resulting OPE using the rules in Table 2.1 is given in Equation 7.

$$F(X) = X_2 X_1 + X_3 X_2 - X_3 X_2 X_1 \tag{7}$$

2.2.2. OPE Calculation Using Logic Diagrams as Input

The OPE can be computed from a structural representation of a Boolean function such as a netlist or logic diagram as well as a functional representation. As an example, consider the logic diagram illustrated in Figure 2.4 that is a realization of the Boolean equation:

$$f(x) = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2x_3x_4 + \overline{x}_1\overline{x}_2x_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3\overline{x}_4 + x_1\overline{x_2}x_3\overline{x}_4 \tag{8}$$

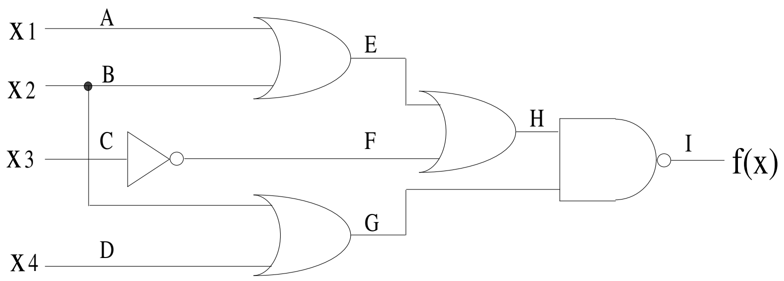


Figure 2.4. Logic circuit example for OPE computation

Using the variables assigned to each interconnection as shown in Figure 2.4 and the rules in Table 2.1, the OPE can be derived as follows. First, the rule for the Inversion operator is applied:

$$F = 1 - C \tag{9}$$

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

$$E = A + B - AB \tag{10}$$

$$G = B + D - BD \tag{11}$$

$$H = E + F - EF \tag{12}$$

Simplifying the equation for H by substituting equations 9 and equation 10 yields:

$$H = 1 - C + AC + BC - ABC \tag{13}$$

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

$$I = 1 - HG \tag{14}$$

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

$$I = 1 - X_2 - X_4 + X_2X_4 + X_3X_4 - X_1X_3X_4 - X_2X_3X_4 + X_1X_2X_3X_4 \tag{15}$$

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

2.2.3. OPE Calculation Using Binary Decision Diagrams as Input

As an example of the OPE calculation, consider the Boolean function,

$$f(x) = \overline{x}_1\overline{x}_2 + x_3 \tag{16}$$

The truth table for equation 16 is given in Figure 2.5 and the the corresponding BDD is given in Figure 2.6.

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 2.5. Truth table for example OPE calculation using a logic diagram

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 2.6 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 [31]. 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. To eliminate infeasible paths, it is sufficient to constrain all parent nodes to have an index value less than their children nodes.

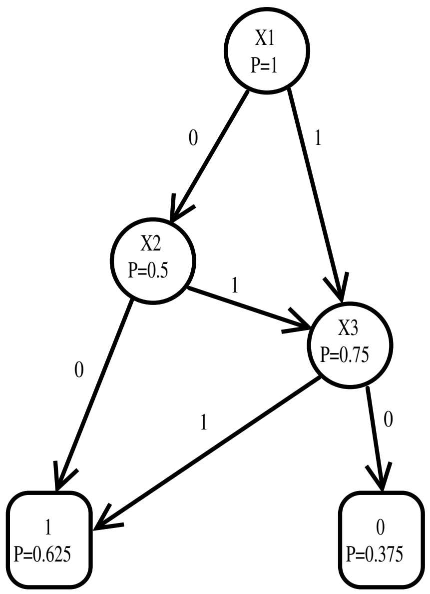


Figure 2.6. Output probability calculation example

2.3. The Relationship Between Output Probabilities and the Spectra

This section will develop some relevant properties of spectral coefficients that are used in the derivation of the efficient spectral calculation algorithm. Two useful properties of spectral coefficients are provided in the following two Lemmas that first appeared in [39].

Lemma 4 *For a given function $f(x)$ and a given constituent function $f_c(x)$ the resulting spectral coefficient is given by:*

$$S_f[f_c(x)] = 2^n - 2N_{mm} = 2N_m - 2^n \tag{17}$$

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 $f_c(x)$. 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 and $f(x)$ 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 5 *The following property of spectral coefficients holds:*

$$S_f[f_c(x)] = -S_f[\overline{f_c(x)}] \tag{18}$$

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 4:,

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

□

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 efficiently 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} is the number of minterm terms in a canonical SOP form of $f \cdot f_c$ and N_{m0} is the number of maxterm 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 [43]. 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.

$$N_m = N_{m1} + N_{m0} \tag{19}$$

$\square$

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

Lemma 7

$$N_m = 2^n[1 + \wp\{f \cdot f_c\} - \wp\{f + f_c\}] \tag{20}$$

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":

$$p_{m0} = 1 - \wp\{f + f_c\} = 1 - \frac{1}{2^n}(N_{m0}) \tag{21}$$

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":

$$p_{m1} = \wp\{f \cdot f_c\} = \frac{1}{2^n}(N_{m1}) \tag{22}$$

Substituting Equations 21 and 22 into Equation 20 and observing that $p_m = p_{m1} + p_{m0}$:

$$N_m = 2^n[p_{m0} + p_{m1}] = p_m 2^n \tag{23}$$

Thus, the definition of N_m is satisfied and the proof is complete. $\square$

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

Theorem 1

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

Proof:

From Equation 17:

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

From Equation 20:

$$N_m = 2^n[1 + \wp\{f \cdot f_c\} - \wp\{f + f_c\}] \tag{26}$$

Substituting 26 into 25 and simplifying:

$$S_f[f_c(x)] = 2^n[1 + 2(\wp\{f \cdot f_c\} - \wp\{f + f_c\})] \tag{27}$$

$\square$

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

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

Proof: From Theorem 1,

$$S_f[f_c(x)] = 2^n[1 + 2(\wp\{f \cdot f_c\} - \wp\{f + f_c\})] \tag{29}$$

Substituting Equations 21 and 22 into Equation 29:

$$S_f[f_c(x)] = 2^n[2(p_{m1} + p_{m0}) - 1] \tag{30}$$

From the definition of p_m:

$$S_f[f_c(x)] = 2^n[2p_m - 1] \tag{31}$$

$\square$

This formulation allows a single spectral coefficient to be computed by performing output probability calculations on the two compositions functions,$f \cdot f_c$ and $f + f_c$. Almost all applications that use spectral coefficients require those corresponding to the constituent functions, $f_{ci} = x_i$, for all i. These coefficients may be obtained by computing only the output probability for one composition function if the 0^{th}-ordered spectral coefficient has been previously computed.

To develop this simplification, it is first noted that the elementary property of probability theory given in Equation 32 holds.

$$\wp\{f + f_c\} = \wp\{f\} + \wp\{f_c\} - \wp\{f \cdot f_c\} \tag{32}$$

Substituting this expression into Equation 27 results in the expressions given in Equations 33 and 34.

$$S_f[f_c(x)] = 2^n[1 + 4\wp\{f \cdot f_c\} - 2\wp\{f\} - 2\wp\{f_c\}] \tag{33}$$

$$S_f[f_c(x)] = 2^n[1 - 4\wp\{f + f_c\} + 2\wp\{f\} + 2\wp\{f_c\}] \tag{34}$$

The expression in Equation 33 holds for any general f_c. However when f_c is equal to a single primary input, the $\wp\{f_c\}$ term is always equal to 1/2 since all primary inputs will be at logic "1" and logic "0" the same number of times. This fact also holds true for all constituent functions used to form the Walsh transforms (with the exception of the 0^{th} ordered coefficient). This simplifies the calculation by requiring only the formation and evaluation of the output probability for the composition function, $f \cdot f_c$. Since the term, $\wp\{f\}$ is related to the 0^{th} ordered spectral coefficient, the expression to calculate the coefficients for this class of constituent functions is given in Equation 35.

$$S_f[f_{ci}] = 2^n(4\wp\{f \cdot f_c\} + 1) - S[0] \tag{35}$$

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 the function to be transformed, $f(x)$. For example, in the synthesis algorithm described in Chapter 4 a method for synthesizing a function by decomposing it into a collection of much simpler constituent functions is given. The decomposition is accomplished by using the information contained in the corresponding spectral coefficients.

2.4. Efficient Spectral Calculations Using Output Probabilities

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 2.2. 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 $O(\|E_{comp}\|)$ where E_{comp} is the number of edges present in the BDDs of the two composition functions.

If the algorithm $APPLY$ proposed in [31] is used to form the composition function BDDs, 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 and many times $\|E_f\| > \|E_{f_{comp}}\|$. 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:

$$p_{m1} = \wp\{f(x) \cdot f_c(x)\} \tag{36}$$

$$p_{m0} = \wp\{\overline{f(x)} \cdot \overline{f_c(x)}\} \tag{37}$$

$$S_f[f_c(x)] = 2^n[2(p_{m1} + p_{m0}) - 1] \tag{38}$$

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

Efficient Spectral Coefficient Computation Algorithm

1: Formulate the BDDs for the two composition functions using the *APPLY* algorithm.

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

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

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 $O(\|E_f\| \times \|E_{f_c}\|)$.

2.4.1. Applicability to Various Types of Circuit Descriptions

This algorithm has allowed the computation of spectral coefficients for some well known benchmark circuits to occur for the first time. Previous methods were impractical due to both the storage and computation time required. As stated in [44] empirical evidence has indicated that most Boolean functions of practical importance may be represented with OBDDs that do not have an exponential number of nodes. For this reason, the spectral computation described here is very applicable. However, there do exist some functions for which the number of nodes is exponential regardless of the variable ordering chosen [45] [46]. For these functions, the method presented here is no worse than other methods for computation of the spectral coefficients, but presents no savings in computation as well.

This technique may be applied to alternative representations of Boolean functions also. In fact, it may be applied to any representation that may be used to directly compute an OPE. As stated in [36], an OPE may be computed directly from a canonic SOP form where all the product terms are minterms, but the worst case complexity is exponential since an exponential number of minterms may be present in the canonic form. The reason that a sum of minterms is required is to ensure that the OPE transformation operations can be applied to each product term independently thereby providing statistical independence. If the statistical independence were not present, the idempotence rule would have to be used. This concept can be extended further by allowing the calculation to occur using any SOP form as long as all of the cubes are disjoint. Two cubes are defined to be disjoint, if one of them does not cover the other and vice verse, or alternatively, if the SOP expression contains only prime implicants [47]. This would preserve the property of statistical independence.

However, such lists of covering cubes can also become exponentially long for some functions.

Another common representation is a structural one usually in the form of a netlist expressed in an HDL, standard such as EDIF, or in terms of a PLA description. Unfortunately, like the case when algebraic descriptions are used, the OPE must be formulated first and then the output probability is obtained by evaluating it. Unlike the use of algebraic descriptions there is no way to enforce statistical independence, thus the OPE must be generated in a symbolic fashion in order to apply the idempotence rule where needed. This method was in fact implemented and the results are provided in a following section of this paper, but the results were not encouraging. The one significant outcome of this experiment was that it is possible to compute spectral coefficients if the input circuit is in two-level form. This occurs because the formation of the OPEs at each internal interconnection remain very small until they are combined at the final output. In fact this method did successfully compute a spectral coefficient for a circuit containing well over 200 inputs. However, this technique generally performs poorly when the circuits become large (more than 20 inputs) and are structured in a multilevel arrangement.

2.4.2. Implementation Using a Functional Description of the Logic Circuit

An example of the application of this algorithm to a 3-input logic function in a functional form, a BDD, can now be given.

Example 1 *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 39.

$$f(x) = \overline{x}_1 \overline{x}_2 + x_3 \tag{39}$$

The constituent function for this example, $f_c(x)$, is given as in Equation 40.

$$f_c(x) = x_2 + x_3 \tag{40}$$

The BDD for Equation 39 is given in Figure 2.6. The BDD for the composition function, $f(x) \cdot f_c(x)$, is given in Figure 2.8 and the BDD for the composition function, $\overline{f(x)} \cdot \overline{f_c(x)}$, is given in Figure 2.7:

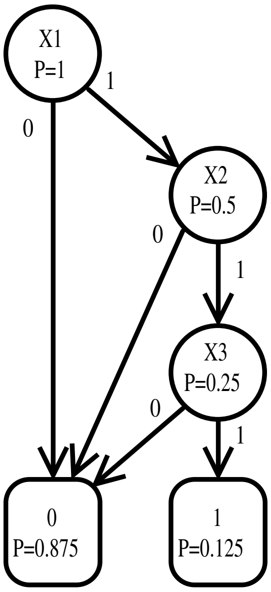


Figure 2.7. BDD of the composition function, $\overline{f(x)} \cdot \overline{f_c(x)}$

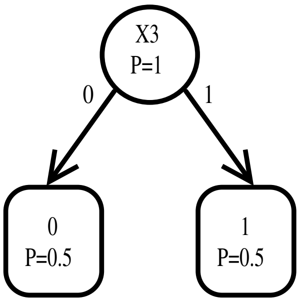


Figure 2.8. BDD of the composition function, $f(x) \cdot f_c(x)$

In order to compute the spectral coefficient determined by the constituent function given in Equation 40, 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:

$$p_{m1} = 0.5 \tag{41}$$

$$p_{m0} = 0.125 \tag{42}$$

Next, the spectral coefficient is computed as:

$$S_f[f_c(x)] = 2^3[2(0.5 + 0.125) - 1] = 2 \tag{43}$$

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 [28] and [29] are prohibited since the inclusive-OR based transform does not yield a sparse or recursively defined transformation matrix (the structure of this transformation matrix for 3-input variables is given in Figure 2.1).

Further, the application of the spectral calculation algorithm presented in [32] [33] [34] may result in the formation of a very large "integer-valued" BDD since the matrix is not sparse and cannot be recursively defined.

In order to evaluate this algorithm on circuits typically encountered in industry, this method was implemented and several standard benchmark circuits were used as input. 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 2 directed arcs leaving it.

The $ISCAS85$ benchmark circuits [48] 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. Tables 2.2 and 2.3 contain spectral coefficients for a selected output for each of the benchmark circuits. In addition to the 0^{th} and 1^{st} ordered coefficients, the sizes of the composite OBDDs are given thus providing a direct representation of the time complexity and hence, execution time 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 $f + f_c$.

Tables 2.2 and 2.3 also contains the number of inputs, n, and the netlist label of the output that was used to create the OBDD. The spectral coefficients $S(f_{c0})$ and $S(f_{ci})$ 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, $f_{c0} = 0$ and

$f_{ci} = x_i$. The specific netlist label for the input chosen as x_i is also present in the table.

Table 2.2.–A zeroth order spectral coefficient for each $ISCAS85$ netlist

Circuit	Output	n	$\|BDD\|$			$S(f_{c0})/2^n$
			f	$f \cdot f_c$	$f + f_c$	
$c432$	$421gat$	36	3970	3963	10	-7.068958×10^{-1}
$c499$	$od0$	41	3378	6307	6307	-9.921875×10^{-1}
$c880$	$878gat$	45	3101	2930	3100	-2.779270×10^{-1}
$c1355$	$1324gat$	41	3378	6307	6307	-9.921875×10^{-1}
$c1908$	66	33	71	63	12	7.690430×10^{-1}
$c2670$	308	122	219	219	216	9.338531×10^{-1}
$c3540$	409	49	36071	36071	36099	-2.162547×10^{-1}
$c5315$	658	67	66552	43486	43485	-5.000000×10^{-1}
$c6288$	$4946gat$	24	17058	14387	6722	-2.929688×10^{-3}
$c7552$	418	194	466	466	1	-9.999999×10^{-1}

Table 2.3.–A first order spectral coefficient for each $ISCAS85$ netlist

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

The set of spectral coefficients formed by using each primary input and the constant logic function, $f_{c0} = 0$, 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 2.4 and 2.5 contain the complete set of Chow parameters for the benchmark circuit $c432$.

Table 2.4.–The first 19 chow parameters for $ISCAS85$ circuit $c432$, output $421gat$

Constituent Function	$\|BDD\|$		Chow Parameter
f_c	$f \cdot f_c$	$f + f_c$	$S(f_{ci})/2^n$
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}

Many CAD applications require the use of constituent functions that are more complex than single primary inputs. In order to demonstrate that this method is applicable for more complex and generalized constituent functions, coefficients were

Table 2.5.–The last 18 chow parameters for $ISCAS85$ circuit $c432$, output $421gat$

Constituent Function	$\|BDD\|$		Chow Parameter
f_c	$f \cdot f_c$	$f + f_c$	$S(f_{ci})/2^n$
$x_{19} = 8gat$	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}

computed for various circuits and arbitrary constituent functions. The constituent functions selected are given by the following expressions:

$$f_{c1} = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$$

$$f_{c2} = \overline{x}_1\overline{x}_2x_5 + x_1\overline{x}_5 + x_2\overline{x}_5$$

$$f_{c3} = \overline{x}_1 \oplus x_4$$

$$f_{c4} = \overline{x_1 \oplus x_2 \ldots x_n}$$

Table 2.6 contains the coefficients and the sizes of the BDDs for the constituent functions given in the preceding. Table 2.7 gives the correspondence of the inputs x_i with the labeled inputs of the $ISCAS85$ circuits.

Table 2.6.–Spectral coefficients for various constituent functions and $ISCAS85$ circuits

Constituent Function	$ISCAS85$ Circuit	Circuit Output	$\|BDD\|$ $f \cdot f_c$	$f + f_c$	Scaled Spectral Coefficient
f_{c1}	$c880$	$878gat$	3045	3022	1.455054×10^{-12}
f_{c2}	$c432$	$421gat$	3659	3782	2.318131×10^{-2}
f_{c3}	$c7552$	276	208	208	-1.250000×10^{-1}
f_{c4}	$c3540$	369	16172	16174	8.553743×10^{-3}

Table 2.7.–Correspondence of labeled inputs with those in the constituent functions

Function	x_1	x_2	x_3	x_4	x_5
f_{c1}	$210gat$	$268gat$	$219gat$	$8gat$	$138gat$
f_{c2}	$4gat$	$1gat$	–	–	$24gat$
f_{c3}	4528	–	–	1492	–
f_{c4}	169	50	58	68	20

2.4.3. Implementation Using a Structural Description of the Logic Circuit

The OPE generation method using logic circuits as input was also implemented using the C language. The motivation for this implementation was to observe the size of the OPE polynomials in relation to the size of a given circuit. Further, while many Boolean functions may be represented in a very compact manner by using the BDD form, there does exist certain classes of functions that require an exponentially sized BDD to be represented. A commonly known example of this phenomenon is that of the multiplier circuit [46], others include the family of expressions given in [45]. Clearly, the method for computing spectral coefficients given here will not result in significant savings in time complexity when the size of the BDD is similar to the size of the truth table specifying the circuit.

As expected, the sizes of the OPEs grew quite rapidly and contained a number of OPE product terms of the order of the number of SOP product terms. It is necessary to symbolically compute the OPE in polynomial form since the idempotence property must be preserved when formulating the composition functions. This is not necessary when the composition BDDs are created since the OBDD form is used.

Of the 10 $ISCAS85$ benchmark circuits, the OPEs were only successfully computed for 1 circuit, $c2670$. The reason this occurred is because this circuit contains many loosely interconnected subcircuits with small depth. Therefore, the polynomials did not have a chance to grow extremely large. Practically all of the circuits had a few outputs with small critical paths and for those outputs the OPEs were also computed successfully. The interesting fact in this outcome is that $c2670$ has over 200 inputs. This fact would render the application of the definition useless for computing the spectrum since each coefficient would require the evaluation of a dot product of two vectors containing well over 2^{200} elements each.

The conclusions of this experiment show that this method is better than the definition for relatively small circuits since a structural description is often more compact than its corresponding truth table. Also, this method is applicable to large circuits that have a structure similar to circuit $c2670$. In fact, since all of the circuits used to test this implementation were multilevel circuits, the results of the experiment may be misleading. The problem occurred when the intermediate polynomials grew very large. If the circuits were expressed in two-level form, the intermediate polynomials would be quite small and only the resulting OPE of the circuit would have a chance to grow too large. Therefore, this technique could be used as a viable alternative for circuits whose OBDDs are too large provided that the circuits are expressed in a two-level form.

CHAPTER 3

SYNTHESIS USING SPECTRAL BASED HEURISTICS

A synthesis technique using a subset of spectral coefficients is described in this chapter. The previous chapter contains the development of a methodology to compute a single spectral coefficient in an efficient manner, however most spectral based methodologies require the entire spectrum to be computed [15] [16]. Although, the computational method presented in the preceding chapter reduces the complexity from exponential to polynomial in terms of the number of primary inputs for most functions, the entire spectrum of a function still contains an exponential number of coefficients. This fact provides the motivation for developing a method that uses a subset of spectral coefficients to perform the synthesis. By using a subset of the coefficients, each coefficient may be efficiently calculated and the number of these computations is no longer exponential.

It has been shown that all 2^n spectral coefficients are required to uniquely represent a Boolean function when the Walsh family of transformation matrices are employed [13] [49]. Thus, a method that uses a subset of coefficients must necessarily employ heuristics since an exact solution cannot be obtained. The use of heuristics in the synthesis of logic functions is very common and has led to some of the most successful tools available today [3] [6]. The primary reason that heuristics are commonly used to solve the synthesis problem is that most logic functions typically encountered today have such a large output space that it is impractical to search for the absolute optimal solution in terms of some minimization criteria. In fact, it has been shown that the optimal solution in terms of minimal area (as measured by a minimal number of implicants in the algebraic expression) is NP-hard to obtain [3].

The method that is presented in this chapter is developed to produce multi-level circuits that may be optimized for area, device and interconnection minimization, timing, and, testability. The one other most popular optimization criteria is the minimization of power consumption. This algorithm does not incorporate that criteria since it functions at the gate level. Most low power design methods employed to date have relied on the use of efficient cells or architectural modifications such as reducing operating frequencies, or, reducing operating voltages [50]. These techniques are not applicable for gate level design other than ensuring the particular cell library used is a low power library.

The optimization for timing versus area presents a well known tradeoff. In order to ensure minimal delay, a two-level circuit composed of a maximally reduced set of implicants is the best that can be achieved in terms of critical path length. Alternatively, minimization of area and interconnections generally require a multilevel circuit so that intermediate term sharing between a set of reduced implicants may be exploited. As the number of levels of circuitry increase, so does the critical path length. The approach used in this method is to generate a multilevel circuit, but to allow the critical path lengths for particular variables to be controlled. This is reasonable since many real world design problems are specified by considering some valid input signals to be present at the inputs of the circuit before others. Hence, to ensure minimal overall circuit delay, it is necessary for the paths to be shorter for the inputs that arrive last. Further, by generating a multilevel circuit, area and interconnection resources are reduced since at each stage of the synthesis, the remaining portion of the function to be realized is chosen such that it exhibits maximal redundancy allowing for primary inputs to be discarded. Finally, testability is enhanced in this synthesis methodology by producing circuits that have an internal fanout value no greater than two with the option (at the cost of increased area count) of restricting it to one.

This method is also highly applicable for use in providing an initial circuit structure to other area minimizers. Many of the current popular minimizers require an initial form of the circuit as input [6] [8]. The final synthesized output can be adversely affected if a highly inefficient input circuit was provided. Further, the optimizations are usually performed by local changes over portions of the circuit, thus the various delays for input variables tend to remain relatively equivalent. The method described here can be used to provide an initial circuit with timing delays strictly minimized and required area loosely optimized. The resulting circuit may then be used as an initial representation for an area optimizer such as those described in the introductory chapter.

Another advantage of this synthesis method is that a purely functional description of the circuit is used for the input form. This allows the synthesis to be performed by automatically converting the logic relations expressed in an RTL description of system to OBDDs and then translating them to a gate level circuit. This relieves the designer from the task of manually converting the RTL level relations into initial circuit representations. Also, some of the current minimizers flatten the input circuit into a two-level form in order to apply the minimization techniques. It is not uncommon for designers to deal with circuits that are so large their two-level form requires more memory than is available. The method proposed here generates a multilevel circuit without resorting to first generating the two-level form.

The rest of this chapter is organized as follows. First, the synthesis method is presented and an examination of the optimization criteria will be provided. Next, the development of the method will be described in detail. The formulation of the spectral heuristics will be explained and the use of the decomposition methods at each stage of synthesis are discussed. Following the discussion of the philosophy behind the technique, implementation issues will be discussed including the program flow. Finally, some examples of this method are presented.

3.1. Description of the Synthesis Methodology

This synthesis technique produces a circuit by determining an output gate first and working back toward the inputs. The output gate is chosen by using the information contained in the subset of spectral coefficients commonly referred to as the Chow parameters. Based upon the properties of the Chow parameters, a set of heuristic rules are applied to choose the appropriate gate. The heuristics have been formulated such that the chosen gate will be maximally correlated to the entire function and hence the remaining portion of the function will be simplified. In order to take advantage of the efficient method for computing the spectral coefficients, the intermediate functions as well as the initial input function are represented in terms of OBDDs.

At each stage of the synthesis, once the output gate is chosen, at least one primary input is removed from each remaining intermediate function. Therefore, the size of the intermediate function always decreases by at least one half. The particular input that is chosen to be removed is determined by the optimization criteria. If timing optimizations are desired, the slower arriving inputs are removed first resulting in fewer gates in their propagation path. If area and interconnection minimizations are required, the input is chosen using the principle of maximal subfunction independence resulting in the intermediate functions being as simple as possible.

Since at least one primary input is discarded at each step in the processing flow of the synthesis technique, convergence is guaranteed. The criteria used in this method may not necessarily realize the absolute optimal circuit, but acceptable engineering solutions comparable with other popular logic synthesizers will result.

3.1.1. Optimization Criteria

This methodology supports optimization for area and interconnection minimization, delay minimization, and, testability. This section will describe how each of these criteria are included in the synthesis methodology. In addition, the tradeoffs between these various optimizations are also discussed.

In the past, area minimization has generally been measured as the number of implicants in a minimized cover of a function and the minimization of gates and interconnections has been measured as the total number of literals in a minimized cover [3]. The most common way of incorporating this type of optimization has been by using the concepts of "don't cares" [2] [6]. Another popular method that has been exploited by many researchers is the use of "permissible functions" [8]. The minimization criteria used in this implementation is the creation of intermediate functions that are as degenerate as possible. Since a single primary input is guaranteed to be removed at each stage of the synthesis, the resulting intermediate functions will contain at least one less primary input. However, if the intermediate functions also become degenerate, additional inputs may be discarded resulting in significantly simpler functions remaining to be realized. Further, the determination of which if any of the inputs are redundant is implicitly achieved since the intermediate functions are represented by OBDDs formed by applying the $RESTRICT$ operation on the original OBDD [44]. This occurs because an OBDD is defined as a BDD with a specific variable ordering that has been maximally reduced [44]. Thus, a redundant input cannot be contained in an OBDD.

In addition to the exploitation of intermediate function degeneracy, interconnection optimization is achieved through the structure that a circuit synthesized by this technique must have. Since each stage allows a single primary input to be discarded from the next synthesis step, a characteristic overall circuit structure results. Primary inputs are discarded through the use of the Shannon Decomposition [51] in most cases. Various forms of this decomposition formula imply the structure of each intermediate portion of the resulting circuit. Figure 3.1 depicts 3 possible forms for a single iteration of this synthesis technique. By choosing forms that incorporate a fanout of two, or, by even restricting those forms to no fanout, interconnections can be minimized and they are also local to the current area of the circuit that is being synthesized. This method does not result in gates near the input side of the circuit to directly drive gates near the output end of the circuit. Thus, gates that are not close together are decoupled within the resulting circuit minimizing interconnection complexities.

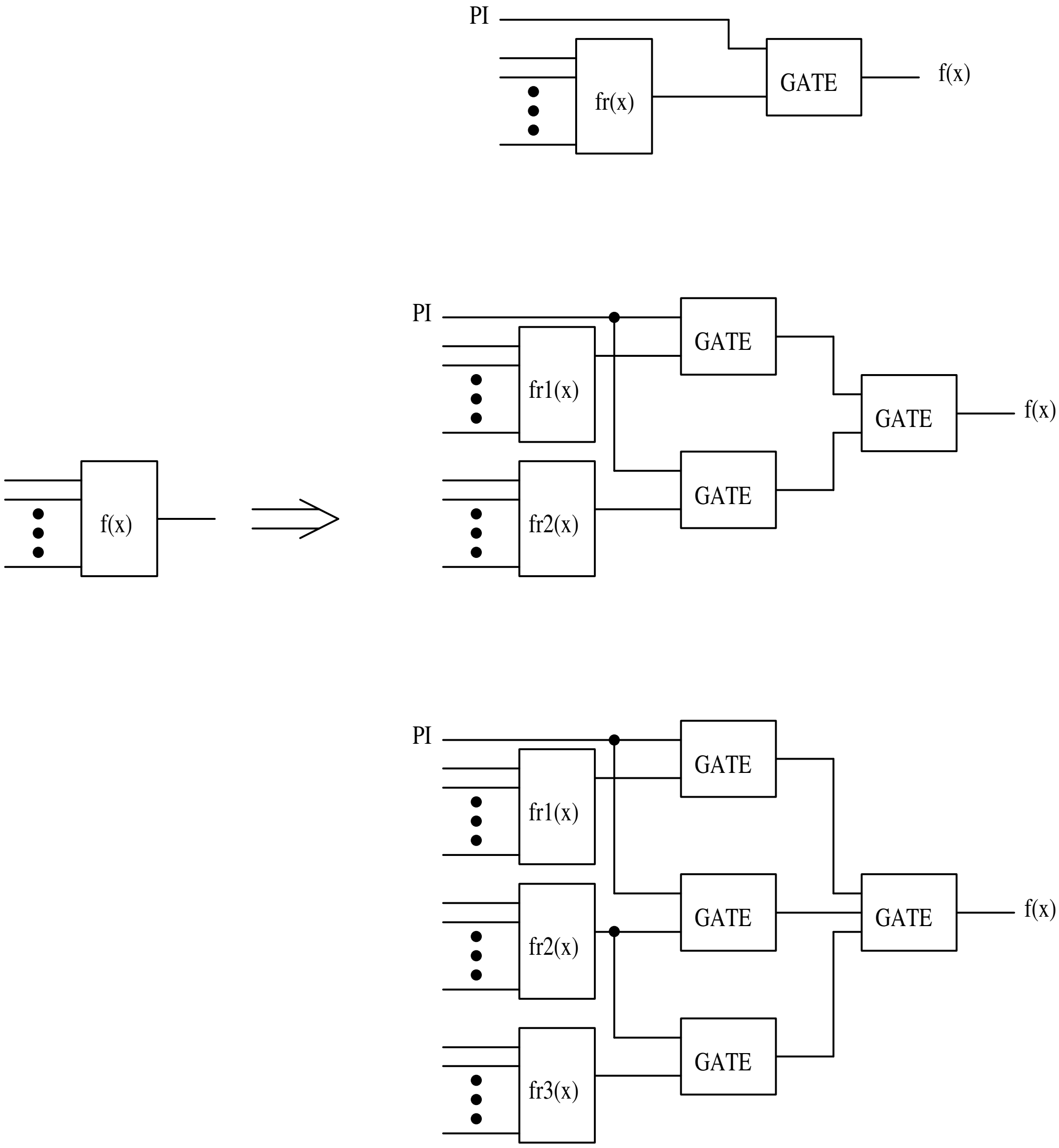


Figure 3.1. Diagram of a single iteration of the heuristic synthesis method

The timing optimization criteria results from discarding the primary inputs from each intermediate function. When the designer supplies the OBDD of the circuit to be synthesized, he must also supply timing information. Specifically, he must group the inputs into classes that are ranked by the speed at which they will appear at the inputs to the resulting circuit. Each class may contain a single input implying a strict timing order of arrival or, at the other extreme, a single class may be specified inferring that all signals will be present at the same time. The specification of these classes in effect dictate the delay versus area tradeoff in the final result. If a strict ordering is supplied, the synthesizer is forced to discard the primary inputs from the intermediate functions in a specific order. Therefore, area minimization is achieved only through the choosing of the particular output gate at each stage. It should be noted that the heuristics that are used to determine the output gates were developed with area minimization in mind so that a strict ordering does not necessarily cause the resulting circuit to be overly large, it just removes a degree of freedom by not allowing the synthesizer to choose the most prudent input to discard. Alternatively, if all inputs are in the same class, implying that all will arrive at the input of the circuit simultaneously, the synthesizer is allowed to choose the input to discard that will most likely result in an intermediate function with a high degree of redundancy as well as to choose which type of output gate to use. The practical way to specify the timing classes is to order only those inputs that are critical in terms of arrival time and to place all others in the same class. This will not only allow the synthesizer to enforce the timing criteria but also give it maximum freedom for minimizing the resultant area.

The third optimization criteria is that of testability. It is well known that completely fanout free (CFOF) circuits are highly testable since they only require a number of test vectors to detect any single stuck-at fault in the circuit equal to the number required to test for a single stuck-at fault at the primary inputs [52]. By choosing the decompositions at each stage of the synthesis to be such that no fanout is generated, the resulting circuit will have no internal fanout and be highly testable.

3.1.2. Spectral Heuristics for Decomposition

This synthesis methodology employs a set of heuristics based upon properties of the Chow parameters. The set of heuristics is used to choose the output gate at each level of synthesis. The basis for the heuristics for the choice of the output gate was the examination of the Chow parameters for all possible Boolean functions of two variables. It is essential that the correct gate be chosen when only two primary inputs remain in order to ensure that the synthesis algorithm terminates and does not oscillate when this terminal condition occurs. The algorithm is thus guaranteed to converge since a primary input is discarded at each intermediate stage and at the terminal stage when only two primary inputs remain a gate is guaranteed to be chosen that will result in termination of the algorithm. The following section will describe the details concerning the development of these heuristics and it will list them in a table.

3.1.3. Maximum Subfunction Independence

The other main idea in the implementation is the maximal redundancy test used to determine which primary input to discard. Since the function to be realized is in OBDD form, it is very efficient to apply the OBDD $RESTRICT$ operation for an input variable. The $RESTRICT$ operation returns a OBDD with a logic 1 or 0 substituted for all instances of a specified input variable. When the $RESTRICT$ operation is applied the returned OBDD will always depend on 1 less variable, and, in many cases several other inputs will also become redundant. The maximal redundancy algorithm computes the OBDDs for the restriction of each input and chooses the input that results in the most redundancy.

3.2. Development of the Technique

The input to the synthesis program is an OBDD representing the circuit to be synthesized. A queue is maintained that points to each intermediate OBDD to be synthesized. Initially, the OBDD of the entire function is placed in the queue. At each stage of the synthesis, an OBDD is popped from the queue. If the OBDD depends on 2 or more inputs, the Chow parameters are computed. Based upon the Chow parameter heuristics, an output gate is chosen. Once the output gate is chosen, the primary input to discard from the remainder functions must be obtained. If there is a timing optimization, the primary input corresponding to the largest arrival time is chosen. Otherwise, the maximal redundancy test is applied to choose the appropriate primary input to be discarded.

3.2.1. Formulation of the Heuristics

The heuristics were derived by observing the Chow parameters for all 16 possible Boolean functions of 2-variables and by exploiting the properties of spectral coefficients. The set of rules are organized in a hierarchical manner so that the rules providing the simplest residual OBDDs are chosen first. Table 3.1 contains the list of heuristics and rules used in the synthesis tool. The heuristics in Table 3.1 is incomplete and more will be added. The value σ is defined as the sum of the first order spectral coefficients as given in Equation 44 and f_{0i}, f_{1i} represent the functions expressed in Equations 45 and 46.

$$\sigma = \sum_{i=1}^{n} S(x_i) \tag{44}$$

$$f_0 = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \tag{45}$$

$$f_1 = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \tag{46}$$

Table 3.1.–Heuristics and rules for the synthesis methodology

MAIN HEURISTIC	SECONDARY HEURISTIC	FUNCTION CHOICE
$\|S(0)\| = 2^n$	$S(0) < 0$	$f = 1$
	$S(0) > 0$	$f = 0$
$\|S(x_i)\| = 2^n$	$S(x_i) < 0$	$f = \overline{x}_i$
	$S(x_i) > 0$	$f = x_i$
$\|S(x_i)\| = 2^n - \|S(0)\|$	$S(0) < 0$ and $S(x_i) < 0$	$f = \overline{x}_i + f_1$
	$S(0) < 0$ and $S(x_i) > 0$	$f = x_i + f_0$
	$S(0) > 0$ and $S(x_i) < 0$	$f = \overline{x}_i \cdot f_0$
	$S(0) > 0$ and $S(x_i) > 0$	$f = x_i \cdot f_1$
$S(0) > 0$	$\sigma < 0$	NOR
	$\sigma > 0$	AND
	$\sigma = 0$ and $S(x_1) \geq 0$	AND
	$\sigma = 0$ and $S(x_1) < 0$	NOR
$S(0) < 0$	$\sigma < 0$	NAND
	$\sigma > 0$	OR
	$\sigma = 0$ and $S(x_1) \geq 0$	OR
	$\sigma = 0$ and $S(x_1) < 0$	NAND
$S(0) = 0$	$S(x_i) = 0 \forall i$	XOR
	$\sigma > 0$	AND
	$\sigma < 0$	OR
	$\sigma = 0$ and $S(x_1) \geq 0$	XNOR
	$\sigma = 0$ and $S(x_1) < 0$	XOR

To illustrate how the heuristics were chosen consider the Boolean functions and their associated Chow parameters given in Table 3.2. There are 16 entries in Table 3.2 corresponding to all possible functions of 2 variables.

As an example, consider the function, $x_1 + x_2$. The zero order spectral coefficient, $S(0)$, is less than 0 and the sum of the first order coefficients, σ, is greater than

0. Therefore, whenever $S(0) < 0$ and $\sigma > 0$, the OR gate is chosen as the dominant function.

Table 3.2.–Chow parameters for all boolean functions of 2 variables

FUNCTION	CHOW PARAMETERS			σ	GATE
	$S(0)$	$S(x_1)$	$S(x_2)$		
0	4	0	0	4	constant 0
x_1x_2	2	2	2	6	AND
$x_1\overline{x}_2$	2	2	-2	2	NOR
x_1	0	4	0	4	literal x_1
$\overline{x}_1x_2$	2	-2	2	2	NOR
x_2	0	0	4	4	literal x_2
$x_1 \oplus x_2$	0	0	0	0	XOR
$x_1 + x_2$	-2	2	2	2	OR
$\overline{x_1 + x_2}$	2	-2	-2	-2	NOR
$\overline{x_1 \oplus x_2}$	0	0	0	0	XNOR
$\overline{x}_2$	0	0	-4	-4	literal $\overline{x}_2$
$x_1 + \overline{x}_2$	-2	2	-2	-2	OR
$\overline{x}_1$	0	-4	0	-4	literal $\overline{x}_1$
$\overline{x}_1 + x_2$	-2	-2	2	-2	OR
$\overline{x_1x_2}$	-2	-2	-2	-6	NAND
1	-4	0	0	-4	constant 1

3.2.2. Shannon Decomposition Forms

Once the dominant output gate is chosen, the function must be decomposed into two residual functions. The decomposition method used in this synthesis tool is based on variations of the Shannon decomposition formula when the chosen output gate is not the XOR or XNOR. The Shannon form was chosen since at least one primary input is guaranteed to be discarded. In order to accommodate the various output gate forms, the Shannon Decomposition Boolean formula was rearranged into several forms. Tables 3.3 and 3.4 list the various forms. In the tables, f_0 and f_1 are used as defined in Equations 45 and 46.

Table 3.3.–Shannon decomposition forms for AND/OR/INVERT dominance

TYPE	CANDIDATE FORM	RESIDUALS
OR	$\overline{x}_i f_0 + x_i f_1$	f_0, f_1
	$f_0 f_1 + \overline{x}_i f_0 \overline{f}_1 + x_i \overline{f}_0 f_1$	$f_0 f_1, f_0 \overline{f}_1, \overline{f}_0 f_1$
	$\overline{\overline{f}_0 + \overline{f}_1} + \overline{x}_i\overline{(\overline{f}_0 + f_1} + \overline{x_i f_o + \overline{f}_1)}$	$\overline{f}_0 + \overline{f}_1, \overline{f}_0 + f_1, f_0 + \overline{f}_1$
	$f_0 f_1 + [f_0 \overline{f}_1 \oplus x_i(f_0 \oplus f_1)]$	$f_0 f_1, f_0 \overline{f}_1, f_0 \oplus f_1$
	$\overline{\overline{f}_0 + \overline{f}_1} + [\overline{(\overline{f}_0 + f_1)} \oplus x_i(f_0 \oplus f_1)]$	$\overline{f}_0 + \overline{f}_1, \overline{f}_0 + f_1, f_0 \oplus f_1$
	$f_0 f_1 + [\overline{f}_0 f_1 \oplus \overline{x}_i(f_0 \oplus f_1)]$	$f_0 f_1, \overline{f}_0 f_1, f_0 \oplus f_1$
	$(\overline{\overline{f}_0 + \overline{f}_1}) + [(\overline{f}_0 + \overline{f}_1) \oplus \overline{x}_i(f_0 \oplus f_1)]$	$\overline{f}_0 + \overline{f}_1, f_0 + \overline{f}_1, f_0 \oplus f_1$
AND	$(x_i + f_0)(\overline{x}_i + f_1)$	f_0, f_1
	$(\overline{\overline{f}_0 \overline{f}_1})(x_i + \overline{\overline{f}_0 f_1})(\overline{x}_i + \overline{f_0 \overline{f}_1})$	$\overline{f}_0 \overline{f}_1, \overline{f}_0 f_1, f_0 \overline{f}_1$
	$(f_0 + f_1)(x_i + f_0 + \overline{f}_1)(\overline{x}_i + \overline{f}_0 + f_1)$	$f_0 + f_1, f_0 + \overline{f}_1, \overline{f}_0 + f_1$
	$\overline{\overline{f}_0 \overline{f}_1}[(x_i \oplus \overline{\overline{f}_0 f_1}) + (\overline{f_0 \oplus f_1})]$	$\overline{f}_0 \overline{f}_1, \overline{f}_0 f_1, \overline{f_0 \oplus f_1}$
	$(f_0 + f_1)\{[x_i \oplus (f_0 + \overline{f}_1] + (\overline{f_0 \oplus f_1})\}$	$f_0 + f_1, f_0 + \overline{f}_1, \overline{f_0 \oplus f_1}$
	$(\overline{\overline{f}_0 \overline{f}_1})[(\overline{x}_i \oplus \overline{f_0 \overline{f}_1}) + (\overline{f_0 \oplus f_1})]$	$\overline{f}_0 \overline{f}_1, f_0 \overline{f}_1, \overline{f_0 \oplus f_1}$
	$(f_0 + f_1)\{[\overline{x}_i \oplus (\overline{f}_0 + f_1)] + (\overline{f_0 \oplus f_1})\}$	$f_0 + f_1, \overline{f}_0 + f_1, \overline{f_0 \oplus f_1}$
NOR	$\overline{\overline{x}_i \overline{f}_0 + x_i \overline{f}_1}$	$\overline{f}_0, \overline{f}_1$
	$\overline{\overline{f}_0 \overline{f}_1 + \overline{x}_i \overline{f}_0 f_1 + x_i f_0 \overline{f}_1}$	$\overline{f}_0 \overline{f}_1, \overline{f}_0 f_1, f_o \overline{f}_1$
	$\overline{\overline{f_0 + f_1} + \overline{x}_i\overline{(f_0 + \overline{f}_1)} + x_i\overline{(\overline{f}_0 + f_1)}}$	$f_0 + f_1, f_0 + \overline{f}_1, \overline{f}_0 + f_1$
	$\overline{\overline{f}_0 \overline{f}_1 + [\overline{f}_0 f_1 \oplus x_i(f_0 \oplus f_1)]}$	$\overline{f}_0 \overline{f}_1, \overline{f}_0 f_1, f_0 \oplus f_1$
	$\overline{(\overline{f_0 + f_1}) + [(\overline{f_0 + \overline{f}_1}) \oplus x_i(f_0 \oplus f_1)]}$	$f_0 + f_1, f_0 + \overline{f}_1, f_0 \oplus f_1$
	$\overline{\overline{f}_0 \overline{f}_1 + [f_0 \overline{f}_1 \oplus \overline{x}_i(f_0 \oplus f_1)]}$	$\overline{f}_0 \overline{f}_1, f_0 \overline{f}_1, f_0 \oplus f_1$
	$\overline{(\overline{f_0 + f_1}) + [(\overline{\overline{f}_0 + f_1}) \oplus \overline{x}_i(f_0 \oplus f_1)]}$	$f_0 + f_1, \overline{f}_0 + f_1, f_0 \oplus f_1$
NAND	$\overline{(x_i + \overline{f}_0)(\overline{x}_i + \overline{f}_1)}$	$\overline{f}_0, \overline{f}_1$
	$\overline{(\overline{f_0 f_1})(\overline{x}_i + \overline{\overline{f}_0 f_1})(x_i + \overline{f_0 \overline{f}_1})}$	$f_0 f_1, \overline{f}_0 f_1, f_0 \overline{f}_1$
	$\overline{(\overline{f}_0 + \overline{f}_1)(\overline{x}_i + f_0 + \overline{f}_1)(x_i + \overline{f}_0 + f_1)}$	$\overline{f}_0 + \overline{f}_1, f_0 + \overline{f}_1, \overline{f}_0 + f_1$
	$\overline{(\overline{f_0 f_1})[\overline{f_0 \overline{f}_1} \oplus x_i(f_0 \oplus f_1)]}$	$f_0 f_1, f_0 \overline{f}_1, f_0 \oplus f_1$
	$\overline{(\overline{f}_0 + \overline{f}_1)[(\overline{f}_0 + f_1) \oplus x_i(f_0 \oplus f_1)]}$	$\overline{f}_0 + \overline{f}_1, \overline{f}_0 + f_1, f_0 \oplus f_1$
	$\overline{(\overline{f_0 f_1})[\overline{\overline{f}_0 f_1} \oplus \overline{x}_i(f_0 \oplus f_1)]}$	$f_0 f_1, \overline{f}_0 f_1, f_0 \oplus f_1$
	$\overline{(\overline{f}_0 + \overline{f}_1)[(f_0 + \overline{f}_1) \oplus \overline{x}_i(f_0 \oplus f_1)]}$	$\overline{f}_0 + \overline{f}_1, f_0 + \overline{f}_1, f_0 \oplus f_1$

When the output gate is chosen as the XOR or XNOR, a spectral based decomposition is attempted first. Usually the spectral based decomposition is very effective in terms of partitioning the primary inputs, however, in those cases where inferior decompositions are achieved, the Shannon decomposition rules in Table 3.4 are used.

The idea behind the spectral based decomposition method is to determine two partitioned subfunctions such that one depends only on highly correlated primary inputs while the other depends upon inputs with a small correlation. When the dominant gate is chosen to be the XOR, the function, f, is partitioned in the form as shown in Equation 47. The subfunction, g, is formed by evaluating f with all highly correlated inputs set to a logic 0 as given in Equation 48.

$$f = g \oplus h \tag{47}$$

$$g = f(0, 0, 0, \ldots, x_{n-i}, x_{n-i-1}, \ldots, x_n) \tag{48}$$

The choice of the highly correlated inputs is made by choosing those inputs whose corresponding spectral coefficients have a magnitude greater than $|2^{n-1}|$. Mathematically stated, the criteria for choosing the highly correlated inputs is given in expression 49. Once g is computed, the corresponding h subfunction is obtained directly by evaluating Equation 50.

$$|S_f[x_i]| > 2^{n-1} \rightarrow \ use\ 0\ for\ x_i \tag{49}$$

$$h = f \oplus g \tag{50}$$

Table 3.4.–Shannon decomposition forms
for XOR/XNOR dominance

TYPE	CANDIDATE FORM	RESIDUALS
XOR	$\overline{x}_i f_0 \oplus x_i f_1$ $f_0 \oplus x_i(f_0 \oplus f_1)$ $f_1 \oplus \overline{x}_i(f_0 \oplus f_1)$	f_0, f_1 $f_0, f_0 \oplus f_1$ $f_1, f_0 \oplus f_1$
XNOR	$\overline{\overline{x}_i \overline{f}_0 \oplus x_i \overline{f}_1}$ $\overline{\overline{f}_0 \oplus x_i(f_0 \oplus f_1)}$ $\overline{f}_1 \oplus \overline{x}_i(f_0 \oplus f_1)$	$\overline{f}_0, \overline{f}_1$ $\overline{f}_0, f_0 \oplus f_1$ $\overline{f}_1, f_0 \oplus f_1$

3.3. Implementation

This synthesis technique was implemented using the C programming language. The basic data structure used in the implementation is a first-in first-out (FIFO) queue that contains pointers to the intermediate OBDDs. Initially the queue is initialized to point to the OBDD representing the function to be realized. At each stage of the synthesis, the OBDD pointed to by the pointer at the top of the FIFO is operated upon and an output gate is chosen for implementation. If any remainder OBDDs are created, pointers corresponding to them are inserted at the tail of the FIFO. The synthesis is complete when the FIFO becomes empty indicating the entire circuit has been built.

3.3.1. Processing Flow

The flowchart depicted in Figure 3.2 illustrates the structure of this tool. The primary data structures are the OBDDs used to represent the Boolean functions and a FIFO queue that contains pointers to the OBDDs. Each iteration operates on a single OBDD removed from the top of the queue and generates one or more simpler, residual OBDDs that are inserted at the tail of the queue.

Each residual OBDD results from the application of the $RESTRICT$ operation [31] which sets a given primary input to a constant logic value and returns the corresponding OBDD. This fact may be used to compute the Chow parameters of the function represented by the output OBDD of the $RESTRICT$ operation directly from those of the input OBDD. However, this requires extra storage since all OBDDs must have their associated Chow parameters stored. This modification proved to be useful only for those functions that required very large OBDDs for their representation since the probability assignment algorithm (PAA) is very efficient for compact OBDDs.

3.4. Examples and Results

Several of the $ISCAS85$ and $IWLS$ benchmark circuits were converted to OBDDs and synthesized using this technique. In order to determine the relative effectiveness of this technique, the $IWLS$ benchmarks were synthesized using the misII tool from Berkeley using a cell library identical to the one used in the method described here. Two different misII scripts were used to synthesize these benchmarks. The first script consisted of an $ESPRESSO$ minimization followed by the misII $simplify$ and $sweep$ commands. The second misII script omitted the $ESPRESSO$ simplification and consisted of the $collapse$, $simplify$, and $sweep$ commands. Table

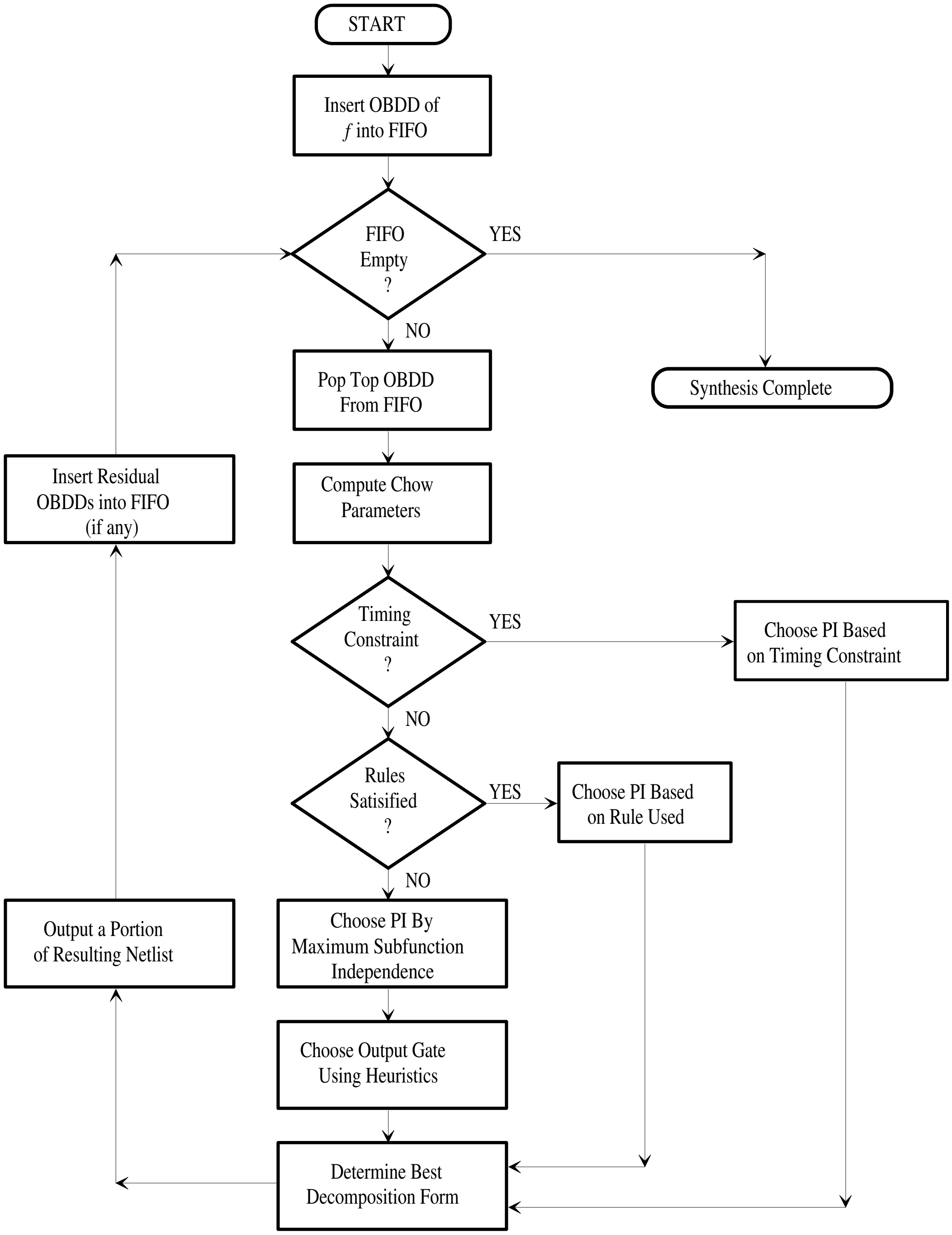


Figure 3.2. Flowchart of spectral based heuristic synthesis method

3.5 contains a summary of these results. The column labeled '*mis*II/*ESPRESSO* Size' contains the number of logic gates *mis*II required when the first script was used as input. The fifth column labeled '*mis*II Size' contains the number of gates the resulting circuit required when the *ESPRESSO* minimization was not used. Finally, the column labeled 'Heuristic Size' contains the number of logic gates in the resulting circuit using the synthesis method just described.

Table 3.5.–Comparison of spectral based heuristic method with *misII*

Circuit Name	Output Name	Number Inputs	*mis*II Size	*mis*II/*ESPRESSO* Size	Heuristic Method Size
*xor*5	*output*1	5	4	41	4
*rd*53	*output*1	5	11	12	9
*rd*53	*output*3	5	4	41	4
*con*1	*output*1	6	8	6	12
*con*1	*output*2	6	7	7	4

Table 3.6 contains a summary of the results obtained when significantly larger circuits are synthesized. The first column contains the name of the benchmark circuit. The remaining columns contain the name of the output, the number of primary circuit inputs, and, the resulting circuit size in terms of logic gates.

Table 3.6.–Experimental results of the spectral based heuristic logic synthesizer

Circuit	Output	Inputs	Size
*c*880	879*gat*	44	174
*c*880	863*gat*	36	113
*c*2670	401	11	20
*c*3540	399	26	60
*c*3540	364	25	29
*c*5315	1002	9	15
*c*5315	661	24	49

Some of the results are surprisingly good. This can be attributed to the high degree of redundancy present in the initial netlist. Since this method was designed to exploit redundancy through the principle of maximum subfunction independence, these results are expected and verify the usefulness of this approach.

CHAPTER 4

SYNTHESIS USING MAXIMUM CORRELATION

This chapter presents an iterative logic synthesis methodology based upon the paradigm of maximum correlation between the constituent and target functions. This approach allows the synthesis to be performed with generalized transformation matrices that do not require any special properties other than being of full rank. The method differs from previous work in that the synthesized circuit is allowed to be composed of any of several types of subfunctions including basic logic gates as well as more complex functions. Previous spectral based methods focused on the realization of the circuit with predominately a single type of logic gate as the basic building block [16].

The fact that any generalized matrix may be employed makes this approach especially attractive for use with existing cell libraries. The only requirement is that each cell in the library be represented by either its truth table or OBDD. Another major advantage of this approach is that the constituent functions may be as simple or as complex as the designer desires. For example, a 5-input AND-OR-Invert (AOI) circuit is as readily usable as a 2-input NAND gate. This gives great flexibility to the designer as well as allowing the method to be easily interfaced with existing cell libraries.

The following development of this technique is provided in the context of using linear algebra to obtain the coefficients. However, it may be implemented using

OBDDs and hence, the efficient methods for computing the spectra as described in Chapter 2. Once the necessary mathematical results are derived, a discussion of the implementation of the synthesis method is provided.

4.1. Description of the Synthesis Method

It has been shown that the Rademacher-Walsh transform provides Boolean function output correlation measures with respect to various combinations of input variables added together via modulo-2 arithmetic [53]. This result allowed a synthesis methodology to be formulated by using the XOR gate predominately. Unfortunately, this method required that the transformation matrix be restricted to the Walsh type since orthogonality was required. Since any general constituent function can also be correlated with a function to be realized as proven in Lemma 1, the method developed here utilizes a more general transformation without requiring orthogonality, and hence any arbitrary constituent function may be used.

The technique can be implemented by using truth tables or alternatively, OBDDs as input. The two methods of implementation result in synthesis methodologies with different characteristics. When truth tables are used as input, this method can produce either two-level or multi-level circuits. However, when OBDDs are used to represent the input functions, a two-level circuit is obtained.

The majority of automated synthesis systems developed thus far have used minimized algebraic expressions. This method is unique in that it does not require an intermediate Boolean function to be realized. Since there is no need for any symbolic manipulation, this technique is especially well suited for implementation as a computer program. Furthermore, the XOR gate is fully exploited as a potential candidate for incorporation into the design in addition to the AND and OR gates.

Most design techniques require symbolic manipulation of Boolean functions to achieve inclusion of the XOR gate. Indeed, it has been stated that algebraic techniques in general are not well suited for circuit realization using XOR circuits. The algebraic methods presented here are rigorously developed and are very well suited for XOR type circuits.

4.1.1. Processing Flow of the Synthesis Technique

The basic philosophy of this technique is to choose the constituent function that has the highest correlation with the function to be realized in an iterative fashion. Since the spectral coefficients provide a measure of correlation as given by Lemma 1, they can be used to determine the candidate constituent function.

Once a highly correlated constituent function has been chosen, it is used to realize a portion of the desired circuit with the remainder of the circuit being computed in truth table or OBDD form through the use of a "combining" operation. The "combining operator" allows the portion of the circuit not covered by the chosen constituent function to be explicitly formed as a new Boolean function and used in the next iteration of the process.

This method realizes the target circuit by constructing it from the input side initially and iteratively forming portions that become closer to the output. Figure 4.1 depicts the flow that occurs when a two-level realization is desired. In the first iteration, the maximally correlated constituent function (denoted by $F'(x)$ in Figure 4.1) is used to realize a portion of the circuit. Next, the remainder, or error function, $e'(x)$, is computed using some combining operator. The combining operator is generally a simple Boolean logic gate. The choice of the exact type of combining gate is discussed fully in a subsequent section. At this point, the system iterates with $e'(x)$ as the next function to be realized.

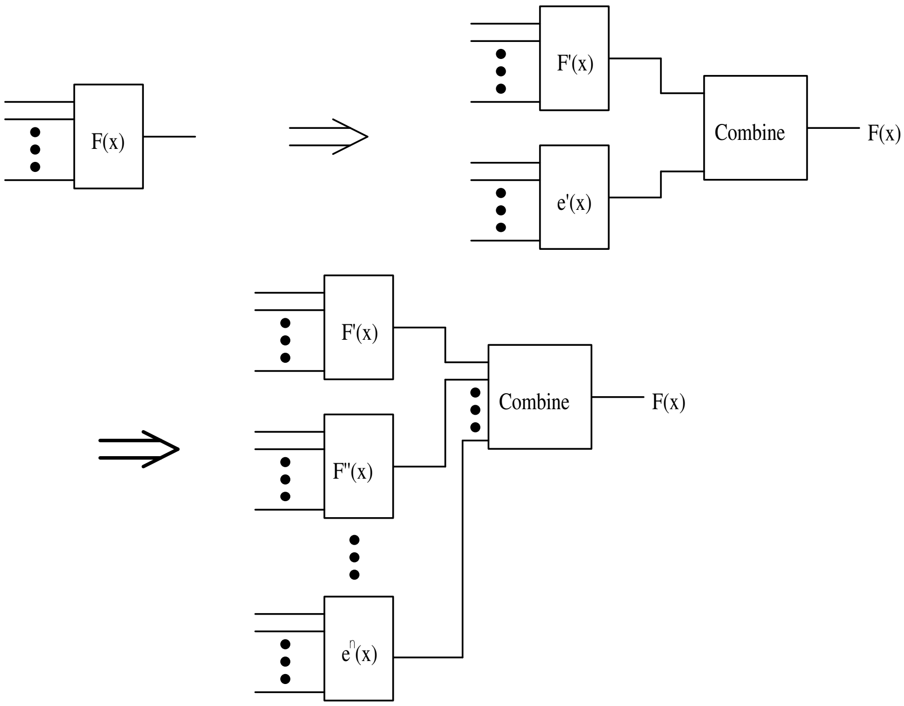


Figure 4.1. Diagram of the two level synthesis technique

At each step of the synthesis, the error functions become increasingly simplified since a constituent function is chosen that covers a maximum amount of the output space of the target function. If a multi-level solution is desired, the combining operator is chosen anew with each iteration. The processing flow for the multi-level case is depicted in Figure 4.2.

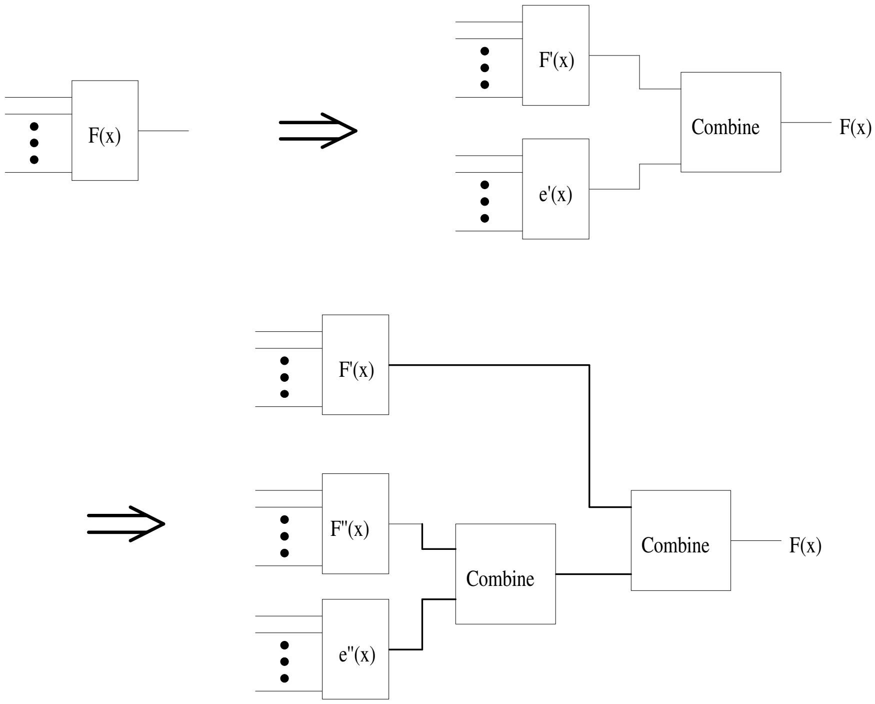


Figure 4.2. Diagram of multi-level synthesis technique

4.1.2. Mathematical Background of the Synthesis Technique

The notion of an output vector of a boolean function was described earlier. A qualitative discussion of the derivation of transformation matrices extends this notion such that the rows of the transformation matrix are viewed as output vectors of the constituent functions. The constituent functions are generally simple functions using only a single operator (although they need not be). For instance, if it is desired to compute the correlation between a constituent function, $x * y$, and a function to be transformed, one row of the transformation matrix would consist of the output vector of the constituent function. Hence, the spectral coefficient resulting from the dot product of this row and the function output vector provides a measure of correlation between the overall function and the constituent function, $x * y$. In fact, a measure of correlation with any arbitrary constituent function may be computed in this manner. Each correlation measure (or spectral coefficient) contains information regarding the exact number of matching outputs between the constituent function and the transformed function for a given common set of inputs.

The transformations used in this synthesis method are linear and they are conveniently visualized as vector-matrix products although the implementation may employ the efficient methods described in Chapter 2. Multiplication and addition operations are performed over the field of integer numbers, not over the binary field. Using "1" and "-1" instead of "0" and "1" for the binary valued digits allows the zero-valued function outputs to accumulate in each spectral quantity.

In general, any set of constituent functions may be used for this technique as long as they form a functionally complete set of operators for Boolean algebra. A transform must be constructed from an orthogonal set of basis functions for the spectrum and original function to form a unique pair. Fortunately, this technique does not rely on transform pair uniqueness. Thus, transform matrices that are not necessarily orthogonal may be used. The transformation matrix must be full rank however, for convergence of this algorithm to be guaranteed. In this paper, we construct transforms using any arbitrary set of functions to form the constituent functions. In addition to choosing a set of basic circuits for the constituent functions, matrix rows are usually included that correspond to each component of the x vector and the $f_c(x) = 0$ function (i.e., the Chow parameters) in case of high correlation with regard to a single input to $f(x)$. The spectral coefficient $S_f(0)$ indicates the overall dominant operator of the function.

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 proposed synthesis technique if the transformation matrix used for the synthesis is of full rank.*

Proof: This proof is a statement that any N-order vector can be produced as a combination of a subset of vectors from a set that are linearly independent over N-space. Each Boolean function to be realized is viewed as a N-order 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 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. $\square$

The one caveat is that this synthesis method can not realize any arbitrary function 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.

4.1.3. Formation of the Transformation Matrix

The matrix is formed by choosing a set of constituent functions and using their output vectors to form the transformation matrix rows. Thus, if the designer wishes to optimize the resulting circuit for low power, he may choose low power constituent functions to form the matrix. Similarly, other desired optimizations may be incorporated by the judicious choice of the constituent functions used to form the transformation matrix.

4.1.3.1. Choosing the Constituent Functions

It is easy to add extensions to this technique to produce output with desired properties. For instance, if only two-input gates are desired, all constituent functions in the transformation matrix are restricted to functions of two-inputs only. Also, by restricting the constituent function operators to a certain type allows the resulting circuit to contain only those operator (gate) types.

If optimization for circuit speed is desired, each spectral coefficient may be weighted by the inverse of the corresponding constituent function delay. This would cause those constituent functions with the least delay and highest correlation to be used.

4.1.3.2. Cell Library Considerations

Existing standard cell logic libraries may be used with this synthesis technique. All that is required is the output vector of each standard logic cell to be used as a row in the transformation matrix. This allows the synthesis technique presented here to be easily interfaced to existing design environments without the need for changing anything other than the "synthesis engine" itself.

4.1.4. Rules for the Combining Gate

The error function may always be 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 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 other gate types that may be used as an error operator.

1. XOR : $x \oplus 1 = \overline{x}$, errors may be 1→0 or 0→1

2. AND : $x1 = x$ and $x0 = 0$, all errors must be 1→0

3. OR : $x + 1 = 1$ and $x + 0 = x$, all errors must be 0→1

4.2. Implementation of the Iterative Method

This technique can be implemented using OBDD, or, truth table descriptions of the constituent functions and the function to be realized as input. The truth table method has obvious disadvantages since it requires an exponential amount of storage. However, it does have the advantage that the type of errors ($1 \rightarrow 0$ or $0 \rightarrow 1$) can be easily seen thus allowing the combining operator to be chosen with each iteration resulting in a multi-level circuit. If the truth table method is combined with an appropriate pre-synthesis partitioning technique, a multi-level synthesis solution can be obtained without incurring excessively high cost in terms of required CPU time.

Implementation using the OBDD as input allows the efficient spectral computation method to be employed resulting in reduced CPU execution time. It is not easy to determine the types of errors at each iteration when OBDDs are used, hence the two-level circuit is the best choice in this case.

4.2.1. Implementation Using an OBDD Input

By implementing this technique using OBDD descriptions as input, the efficient spectral computation method can be employed. Figure 4.3 illustrates the overall flow of the synthesis methodology. User supplied input consists of the OBDD 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 choice of constituent functions). 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 Figure 4.3.

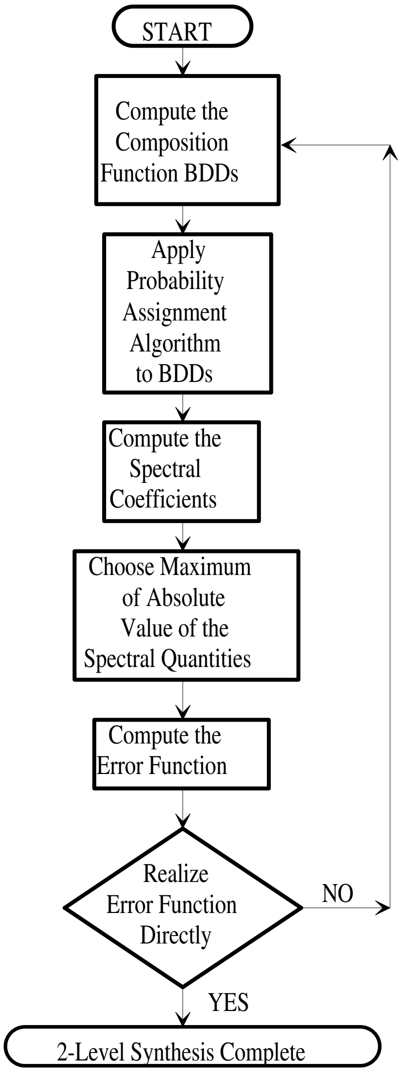


Figure 4.3. Flowchart of two-level synthesis technique

OBDD-Based Maximum Correlation Synthesis Algorithm

1: Formulate the composition OBDDs using $\{f_c(x)\}$ and $f(x)$.

2: Apply the Probability Assignment Algorithm to the composition OBDDs.

3: Compute the spectral coefficients using Equation 35.

4: Choose the largest (in magnitude) spectral coefficient.

5: Realize the function $f_c(x)$ that corresponds to the chosen coefficient in step 4.

6: Compute the OBDD 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 [15], the use of spectral design techniques for logic synthesis is known for the ability to produce easily tested circuits.

4.2.1.1. Example Using an OBDD Input

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 represented by the OBDD in Figure 4.4. The set of constituent functions includes functions that are equal to each component of the $\underline{x}$ vector and the $f_c(x) = 0$ function (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.

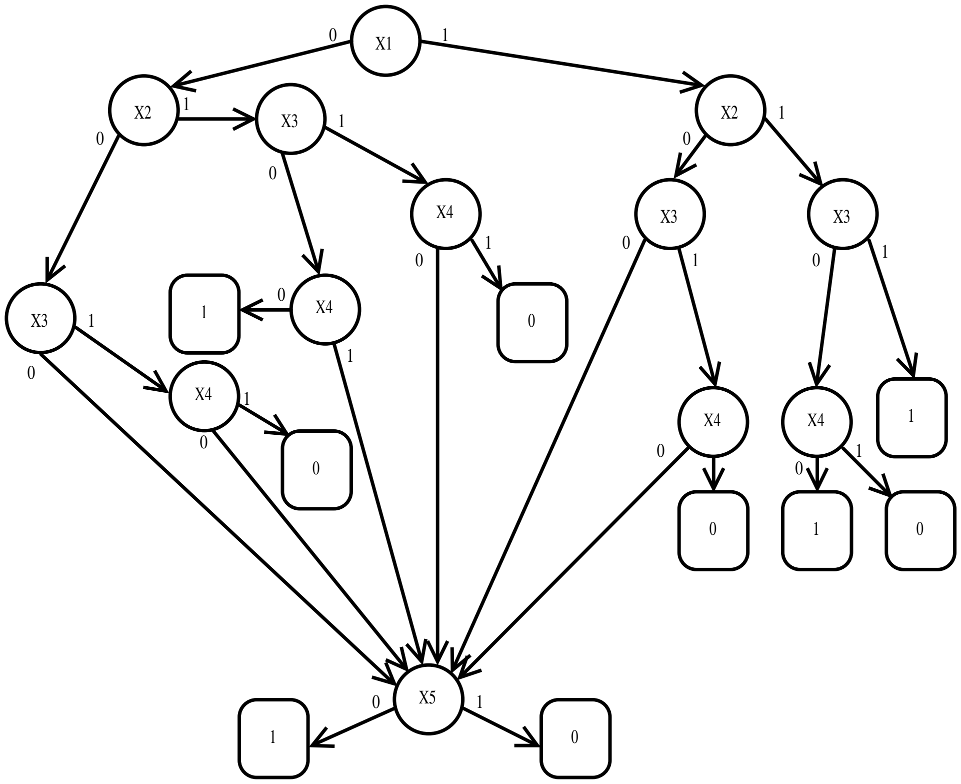


Figure 4.4. OBDD of function for synthesis example

First, the composition function OBDDs are computed. Next, the spectral coefficients are computed using the probability assignment algorithm and Equation 35.

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 Figure 4.5.

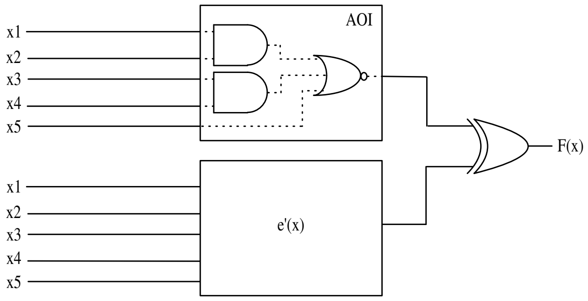


Figure 4.5. First iteration of two-Level synthesis of example "function"

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 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.

After, the first iteration, the OBDD of the error function is computed by using the *APPLY* algorithm with $f(x)$ and $\overline{x_1x_2 + x_3x_4 + x_5}$ as inputs. The resulting OBDD is shown in Figure 4.6.

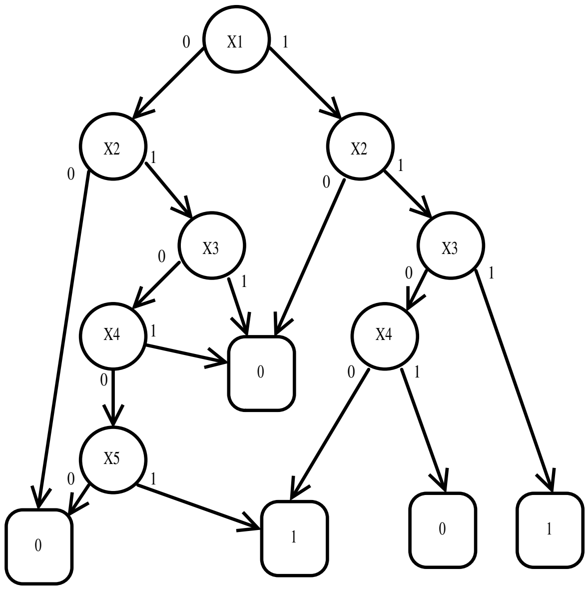


Figure 4.6. OBDD of the residual function after the first iteration

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 Figure 4.7.

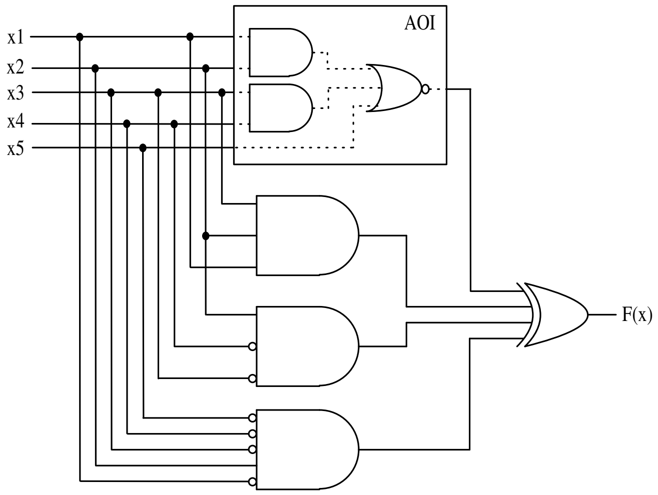


Figure 4.7. Final circuit using the design process

4.2.2. Implementation Using a Truth Table

The implementation of this technique using a truth table description has some advantages. If the function to be synthesized can be effectively described in terms of a truth table, the complexity of computing the spectral coefficients by invoking an inner product computation are usually not severe. Further, the resulting synthesized circuit can have a multi-level topology, allowing for the benefits of reduced fanin and term sharing to be exploited.

As in the OBDD implementation of this algorithm, the user provides the maximum number of inputs per gate, N_{inp}, and preferences of the types of gates to be used, $\{G_t\}$, in addition to the truth able description of the circuit. The two optional parameters, N_{inp} and the set $\{G_t\}$ are used to determine the set of constituent functions, $\{f_c(x)\}$ to be used in the formation of the transformation matrix.

The following list gives a detailed description of each synthesis step for the process.

Truth Table-Based Maximum Correlation Synthesis Algorithm

1: Convert the input truth table to 1's and -1's using a 1 to denote a logic "0" and a -1 to denote a logic "1".

2: Compute the transformation matrix using the constituent functions.

3: Compute the spectral coefficients via vector-matrix multiplication between the transformation matrix and the output vector of the function to be synthesized.

4: Choose the largest (in magnitude) spectral coefficient.

5: Realize the function $f_c(x)$ that corresponds to the chosen coefficient in step 4.

6: Compute the error function, $e(x) = f_c(x) * f(x)$ with respect to some operator, $*$.

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 3 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 $*$ operator and directly realize the function $e(x)$ for the remaining w or fewer errors.

This technique can be used to generate two-level and multi-level tree-type circuits. For two-level realizations, each chosen $f_c(x)$ can be realized in the first stage of the circuit with one multi-input logic gate. The second stage consists of a single combining gate that uses the outputs of all of the chosen constituent functions as its' inputs.

If a multi-level circuit is desired, the same design procedure is used but the error function computation is performed slightly differently. The difference is that a new combining gate is used at each iteration. This allows for changing the operator used to define the error function (i.e., the combination operator) at each iteration.

Note that the only real difference between the two-level and multi-level synthesis techniques is in the choice of new error operators at each iteration. This allows for greater flexibility when it is desired to use one gate type as much as possible since that type may be able to be used for the error operator in each iteration.

4.2.2.1. Example Using Truth Table Input

Examples of the truth table input form of this synthesis technique will now be given. In the first example, it is assumed that there are no restrictions on the number of inputs per logic gate and the types of logic gate are chosen to be the AND, OR, XOR and their complements. The first example will show how a two-level circuit can be synthesized. The second example shows how a multi-level circuit can be realized with the constraints that only two-input gates should be used and that the gate type should be OR as much as possible.

Consider the realization of the following function:

$$f(x) = \overline{x}_1\overline{x}_3 + x_1\overline{x}_2x_3 + \overline{x}_1x_2 + x_2\overline{x}_3 \tag{51}$$

First, the function truth table is computed. Figure 4.8 shows the contents of the truth table for the function given in Equation 51.

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 4.8. Truth table contents of the function to be synthesized using the truth table as input

Next, the transformation matrix is computed in terms of OR, AND, and XOR operations. Note that rows are not included for the NOR, NAND, and XNOR operations since by Lemma 5 the arithmetic sign of the spectral coefficients can be used to indicate when the complementary gates are needed. Also, for each gate type we compute $f_c(x)$ functions for all possible combinations of two or more inputs since we have no restrictions on the maximum number of inputs per gate.

The transformation matrix is computed as:

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

The resulting spectral coefficient vector is:

$$\underline{S_f^T}[\{f_c(x)\}] = [-2, -2, 2, -2, 2, -2, 2, -6, 2, -2, 2, 2, -2, -2, -2, -4] \tag{52}$$

Since the constituent function, $f_c(x) = x_1 \oplus x_2 \oplus x_3$, has the largest magnitude spectral coefficient, it is chosen and the first portion of the circuit is realized with an exclusive-NOR as shown in Figure 4.9. Next, the output vectors of the residual and error functions are computed and the are given in the table shown in Figure 4.10.

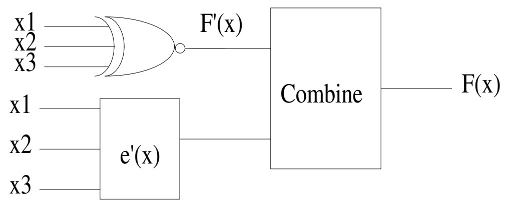


Figure 4.9. First iteration of two-level synthesis of example function

x_1	x_2	x_3	$f(x)$	$f'(x)$	$e(x)$
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

Figure 4.10. Truth table contents of the function and error function

Since the output vector for $e(x)$ indicates that there is only 1 disagreement between $f(x)$ and $f'(x)$, the terminal condition has been reached and the remaining term is realized directly. The complete circuit is given in Figure 4.11.

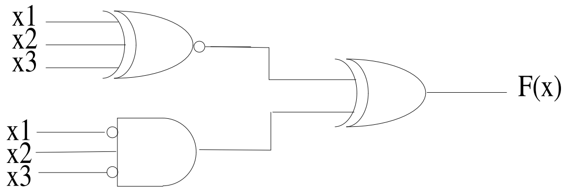


Figure 4.11. Final circuit using two-level design process

The design of a multi-level circuit with the restriction that all gates are two-input and must be OR type gates as much as possible is given for the second example. The OR operator cannot be used to form a functionally complete set of operators for Boolean algebra. Therefore, at least one other type of gate is needed for the synthesis. This example will also realize the function given in Equation 51.

The transformation matrix is computed in terms of the Chow parameters and two input OR expressions only. The spectral vector for the first iteration is given in Equation 53.

$$\underline{S_f^T}[\{f_c(x)\}] = [-2, -2, 2, -2, 2, -2, 2] \tag{53}$$

In this case, all spectral coefficients have equal magnitudes. The constituent function $f_c(x) = x_1 + x_2$ is arbitrarily chosen as a starting point for the synthesis. It is also desirable to use an OR operation for the error function operator since this circuit is to be realized with predominately OR-type gates. By examining the truth table it is seen there are 0 to 1 and 1 to 0 discrepancies which restrict the error operator to be of type XOR. Figure 4.12 illustrates the truth table in terms of the function to be realized and the error function.

x_1	x_2	x_3	$f(x)$	$e(x)$
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

Figure 4.12. Truth table contents of the function and error function for the second synthesis example

Next, the spectrum for $e(x)$ is computed, resulting in the following vector of spectral coefficients:

$$\underline{S_e^T}[\{f_c(x)\}] = [2, 2, -2, -4, -2, -2, -6] \tag{54}$$

The constituent function corresponding to -6 is chosen since it is a maximum (in magnitude) which is $f_c(x) = x_2 + x_3$. There is a discrepancy in only one place in the truth table, hence we can use a NOR to directly realize this term. The resulting circuit is given in Figure 4.13.

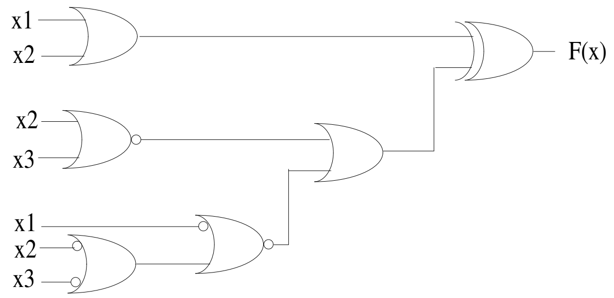


Figure 4.13. Final circuit using multi-level design process

4.2.3. Complexity of the Iterative Method

The section discusses the computational complexity of the method for both types of implementations. The implementation using the OBDD representations clearly reduces the computational complexity incurred in the calculation of each individual coefficient. In contrast to other spectral synthesis methodologies, it is not required to maintain the the entire spectral vector only the maximally correlated coefficient. Thus there are no storage problems encountered for maintaining a very large vector of spectral values.

4.2.3.1. Complexity Using an OBDD Input

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 OBDDs and the probability assignment algorithm are used, the complexity is reduced from $O(2^n)$ to $O(\mid E_{f_c} \mid\mid E_f \mid)$ for the computation of 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 Equation 55. 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 8. The reason 8 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 5 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.

$$8\begin{pmatrix} n \\ 2 \end{pmatrix} = 4(n)(n-1) = 4n^2 - 4n \tag{55}$$

Added to this value is $n+1$ additional matrix rows for the computation of the Chow parameters [53] 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 $O[n^2(\mid E_{f_c} \mid\mid E_f \mid)]$. A further observation is that an efficient variable ordering of a OBDD can result in the number of edges being of order, $O(n)$, [54] [55]. Thus the total complexity of an iteration of the synthesis algorithm is $O(n^4)$ assuming efficient OBDD orderings and equal complexity of the OBDDs used to express $f(x)$ and each member in the set $\{f_c(X)\}$.

4.2.3.2. Truth Table Method Complexity

Although it is readily apparent that this matrix grows in size with respect to the size of the x vector, as discussed in the preceding section the matrix space complexity is $O(n^2)$ if the design is restricted to two-input gates.

However, the computation of each coefficient has a complexity of $O(2^n)$. Therefore, the overall complexity when all possible two-input gates are used as constituent functions is $O(n^2 2^n)$. This exponentially bounded complexity precludes this method from being applied to functions of a larger size. However, if partitioning or decomposition methods are used, this technique can provide a viable synthesis solution.

CHAPTER 5

SYNTHESIS OF GENERALIZED REED-MULLER NETWORKS

As the complexity of VLSI circuitry increases, size and testability concerns tend to grow proportionally. The Exclusive-OR Sum Of Products (ESOP) form of a Boolean logic circuit provides two distinct advantages over the more traditional canonical forms. The ESOP form is similar to the Sum Of Products (SOP) representation with the difference that exclusive OR gates are used instead of inclusive OR gates. It has been shown that ESOP forms generally require fewer logic gates than the SOP forms [56]. For symmetric functions it has been proven that the ESOP form will never require more product terms than the number needed for the corresponding SOP realization [57]. Many field programmable gate arrays (FPGAs) are now being manufactured that utilize the XOR gate as a basic cell component requiring ESOP implementations to be considered when an FPGA implementation is desired. Another chief advantage of the ESOP form is that all single stuck-at faults can be tested with a minimal number of test vectors [58]. Finally, recent developments in layout technology have reduced the layout area required for the XOR gate so that it is comparable with other basic logic gates [59].

Previously, the automated synthesis of ESOP logic circuits was accomplished via symbolic Boolean algebra manipulations [60] [61], or the use of the Reed-Muller transforms [22] [40] [49]. The research results discussed in this chapter are used to develop an approach similar to the Reed-Muller transform technique, however, a more general mathematical framework is derived allowing the calculations to be performed over the field of real numbers. This allows the problem to be specified in a manner that is suitable for the application of the efficient spectral calculation technique discussed in Chapter 2. Specifically, it is shown that the Reed-Muller (RM) synthesis methodology can be reduced to solving a system of linear equations over the real-number field.

These theoretical results were used to implement an ESOP synthesis system in the form of a behavioral to structural translator using the $Verilog$ hardware description language (HDL). The use of automatic behavioral to structural translators in the design of digital circuits allow the designer to spend more time in the specification stage of the design cycle. This is particularly advantageous when the desired circuit is specified with a behavioral description in terms of an HDL. By using automatic behavioral to structural translation tools, the design process becomes closer to the goal of simply specifying a circuit and then allowing automated tools to perform the implementation. Another advantage is that the problem of error introduction during the manual translation of the behavioral to structural circuit description is eliminated allowing for faster design turn-around time and ultimately reducing the design cost.

5.1 Review of the RM Transform and Generalized ESOP Forms

Before the real-valued RM transform is developed, a brief review of ESOP forms is presented. The ESOP form can be closely related to the SOP form and the development of this relationship leads to the RM transform. A mathematically succinct development of the RM transform is given in [40]. In this development it is pointed out that the XOR and AND operations are identical to the additive and multiplicative operators in the modulo-2 finite field algebra, formally known as the Galois field algebra and denoted as GF(2). This coincidence is extremely useful since all the axioms of GF algebras may be used to develop new CAD techniques.

In order to derive the RM transform, it is noted that the modulo-2 multiplicative operator and the Boolean algebra multiplicative operator are both realized by the AND function. The additive operators for these two algebras differ however. The Boolean algebra uses the inclusive-OR operation while the modulo-2 algebra uses the exclusive-OR operation for addition. The relationship between these two algebraic systems can be derived using the axioms and postulates of Boolean algebra beginning with the relation given in Equation 56.

$$a + b = a \oplus b \oplus a \cdot b \tag{56}$$

An implicit result of this relationship leads to the following Lemma.

Lemma 8 : *The ESOP form is functionally complete and hence any arbitrary digital logic function may be realized in ESOP form.*

Proof: It is a well-known fact that the AND, OR, and NOT (inversion) operations form a functionally complete set. Each of these operations may be formulated in terms of the GF(2) additive and multiplicative operators (which were previously noted to be identical to the XOR and AND functions respectively) as:

$$a \cdot b = a \cdot b \tag{57}$$
$$a + b = a \oplus b \oplus a \cdot b \tag{58}$$
$$\overline{a} = 1 \oplus a \tag{59}$$

Therefore, by inference, the GF(2) additive and multiplicative operators form a functionally complete set. □

The relationships given in Lemma 8 may be used to develop a formal transformation method. All possible Boolean functions of one variable may be expressed as the relationship in Equation 60 where $d_i = 1$ or 0.

$$f(x_1) = d_0\overline{x}_1 + d_1x_1 \tag{60}$$

Using de Morgan's theorem and the relationships in Lemma 8, Equation 60 can be rewritten and is given in Equation 61.

$$f(x_1) = d_0 \oplus (d_0 \oplus d_1)x_1 \tag{61}$$

At this point it is useful to define two additional variables, $c_0 = d_0$, and $c_1 = d_0 \oplus d_1$. Substituting these definitions into Equation 61 yields the relationship given in Equation 62.

$$f(x_1) = c_0 \oplus c_1 \cdot x_1 \tag{62}$$

It is easy to see that the transformation from the d_i values to the c_i values is given by the relationship in Equation 63.

$$\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \end{bmatrix} \tag{63}$$

It is recognized that the matrix of coefficients in Equation 63 is the well known RM transformation matrix for Boolean functions of one variable. All operations in the RM transform relation given in Equation 63 are performed using modulo-2 arithmetic. The matrix of coefficients is referred to as G_1 where the subscript denotes the number of variables in the expression to be transformed. The RM transformation matrix can be conveniently defined in a recursive manner for functions of more than one variable as shown in Equation 64.

$$G_n = \begin{bmatrix} G_{n-1} & 0 \\ G_{n-1} & G_{n-1} \end{bmatrix} \tag{64}$$

This form of the RM transformation matrix yields a specific type of ESOP expression referred to as the complement-free ring-sum form since no input variables are complemented in the transformed expression. This is only one of the generalized RM (GRM) forms. In past literature the GRM forms were classified using a polarity number [40]. The complement-free ring-sum expression is equivalent to the polarity-$2^n - 1$ RM form where the polarity value refers to inputs that are present in the expression in inverted form.

For any general function, there exist 2^n different unique fixed polarity forms since there are 2^n different inverted and non-inverted primary input combinations possible for an n-input function. The polarity numbers are positive integers whose value lie in the interval, $[0, 2^{n-1}]$. The particular polarity number refers to which primary inputs are inverted. For example, the polarity-1 GRM expression contains only the x_1 primary input in non-inverted form while the polarity-3 form indicates that both x_1 and x_2 are present in a non-inverted form with the remaining $n-2$ inputs being complemented. There is a unique polarity-k expression for every possible Boolean function, hence there exists a unique RM transformation matrix for each polarity-k form as well. The derivation of the polarity-$2^n - 1$ form of RM transformation matrix has been given above, the transformation matrices corresponding to the other polarization numbers may also be easily derived and the details can be found in [40].

The uniqueness of the 2^n different GRM forms ultimately arise because for a given polarity, every input may be present in either complemented or uncomplemented form, but not both. However, in the most general sense, an ESOP form can include both complemented and uncomplemented literals in a single expression. In the work of Sasao [62], the more general expressions are classified in terms of the number of different possible representations for a given Boolean function. The complement-free ring-sum (or polarity-2^n form) is called the positive polarity RM expression (PPRME). The class of 2^n GRM (or fixed polarity) RM expressions are denoted as the fixed-polarity RM expressions (FPRME). The PPRME is a subset of the FPRME class and is unique, thus it contains a single member. These other forms are defined by observing the structure of the algebraic expression used to describe them. The ESOP form is described by Equation 65 and the other forms are described by Equation 66.

$$f(x_1, x_2, \cdots, x_n) = r_0 p_0 \oplus r_1 p_1 \oplus \cdots \oplus r_k p_k \tag{65}$$

Where each p_i refers to a product composed of any number of literals.

$$\begin{aligned} f(x_1, x_2, \cdots, x_n) = {} & r_0 \oplus r_1 \dot{x_1} \oplus \cdots \oplus r_n \dot{x_n} \oplus r_{12} \dot{x_1} \dot{x_2} \oplus \\ & r_{13} \dot{x_1} \dot{x_3} \oplus \cdots \oplus r_{n,n-1} \dot{x_n} \dot{x_{n-1}} \oplus \cdots \oplus r_{12 \cdots n} \dot{x_1} \dot{x_2} \cdots \dot{x_n} \end{aligned} \tag{66}$$

In the work in [62], Equation 66 represents 3 major classes of ESOP forms which are differentiated by the following properties.

GRME: $\dot{x_k} = x_k$ or $\overline{x_k}$

FPRME: $\dot{x_k} = x_k$ or $\overline{x_k}$ exclusively

PPRME: $\dot{x_k} = x_k$

The most general form, the ESOP, may have a particular product term, $p_1 = \overline{x}_0 x_1$, and another product term, $p_2 = x_0 \overline{x}_1$ while this would not be allowed for the more restrictive GRME class.

The diagram in Figure 5.1 shows the relationship of the various ESOP forms. While a unique form exists for a given polarity, the fact that 2^n different polarities are present allows for the consideration of the minimal polarity form for a given function. The minimization problem can be further generalized by considering all possible ESOP forms. Currently, the ESOP minimization problem is open and an active area of research. The various proposed minimization techniques for ESOP circuits provide the motivation for determining an efficient way to realize a particular ESOP form given a particular polarity number.

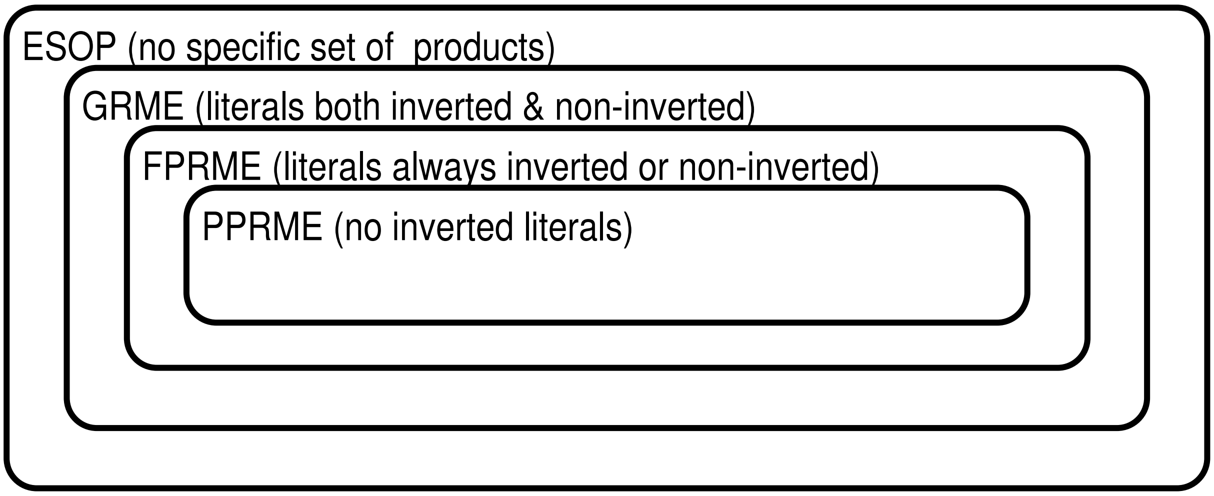


Figure 5.1. Venn diagram of various classes of ESOP functions

5.2 Development of the Real-Valued RM Transform

The development of the mathematical foundations of the computation of the RM transform using real arithmetic rely upon the concepts of algebraic rings and their morphisms.

A Boolean ring satisfies all of the properties of a general ring in addition to the idempotence property [63]. The familiar Boolean algebra may be expressed in terms of a Boolean ring, $\Re$, defined as follows:

$$\Re : \{0, 1, \oplus, \cdot\}$$

Where 0 and 1 denote the logic values of zero and one, the addition operator, $\oplus$, denotes the binary XOR operation, and the multiplicative operator, $\cdot$, denotes the binary AND function. The set, $\Re$, forms a Boolean ring since all of the properties of a general ring are satisfied [64] along with the idempotence property.

Next, consider an alternative set, $\Re'$. This set contains the following elements:

$$\Re' : \{\emptyset, I, +, \times\}$$

Where:

$$\emptyset \equiv \{0, \pm 2, \pm 4, \pm 6, \ldots, \pm 2j, \ldots\} \tag{67}$$

and,

$$I \equiv \{\pm 1, \pm 3, \pm 5, \ldots, \pm(2i+1), \ldots\} \tag{68}$$

Thus, $\emptyset$ and I are the sets of all even integers (including 0) and all odd integers (excluding 0), respectively. The two operators in the set, $\Re'$, are the additive operator, $+$, and the multiplicative operator, $\times$. These operators perform real addition and real multiplication, respectively. $\Re'$ also satisfies the properties of a Boolean ring [63].

5.2.1 Isomorphic Relationship of the RM Transform and the Real-Valued Number System

The following lemma establishes the relationship between the two rings $\Re$ and $\Re'$:

Lemma 9 : *The function, $f(x) = x(\text{mod}2)$, forms a ring morphism from $\Re'$ to $\Re$.*

Proof: To prove this lemma, we demonstrate that the following relationships exist:

$$f(a+b) = f(a) \oplus f(b) \tag{69}$$
$$f(a \times b) = f(a) \cdot f(b) \tag{70}$$

for the following three exhaustive cases:

Case 1: $(a, b) \in \emptyset$

Case 2: $(a, b) \in I$

Case 3: $a \in \emptyset, b \in I$

Case 1:

Let $a = 2n$ and $b = 2m$:

$$\begin{aligned} f(a+b) &= f(2n+2m) \\ &= [2(n+m)](\text{mod}2) \\ &= 0 \\ f(a) \oplus f(b) &= [2n(\text{mod}2)] \oplus [2m(\text{mod}2)] \\ &= 0 \oplus 0 \\ &= 0 \end{aligned}$$

$\therefore f(a+b) = f(a) \oplus f(b)$

$$
\begin{aligned}
f(a \times b) &= f(2n \times 2m) \\
&= 4nm(\text{mod}2) \\
&= 0 \\
f(a) \cdot f(b) &= [2n(\text{mod}2)] \cdot [2m(\text{mod}2)] \\
&= 0 \cdot 0 \\
&= 0
\end{aligned}
$$

$\therefore f(a \times b) = f(a) \cdot f(b)$

Case 2:

Let $a = 2n + 1$ and $b = 2m + 1$:

$$
\begin{aligned}
f(a + b) &= f(2n + 1 + 2m + 1) \\
&= [2(n + m + 1)](\text{mod}2) \\
&= 0 \\
f(a) \oplus f(b) &= (2n + 1)(\text{mod}2) \oplus (2m + 1)(\text{mod}2) \\
&= 1 \oplus 1 \\
&= 0
\end{aligned}
$$

$\therefore f(a + b) = f(a) \oplus f(b)$

$$
\begin{aligned}
f(a \times b) &= f[(2n + 1) \times (2m + 1)] \\
&= [2(2nm + n + m) + 1](\text{mod}2) \\
&= 1 \\
f(a) \cdot f(b) &= (2n + 1)(\text{mod}2) \cdot (2m + 1)(\text{mod}2) \\
&= 1 \cdot 1 \\
&= 1
\end{aligned}
$$

$\therefore f(a \times b) = f(a) \cdot f(b)$

Case 3:

Let $a = 2n$ and $b = 2m + 1$:

$$\begin{aligned} f(a+b) &= f(2n+2m+1) \\ &= [2(n+m)+1](\mathrm{mod}2) \\ &= 1 \\ f(a) \oplus f(b) &= 2n(\mathrm{mod}2) \oplus (2m+1)(\mathrm{mod}2) \\ &= 0 \oplus 1 \\ &= 1 \end{aligned}$$

$\therefore f(a+b) = f(a) \oplus f(b)$

$$\begin{aligned} f(a \times b) &= f[2n \times (2m+1)] \\ &= [2(2nm+n)](\mathrm{mod}2) \\ &= 0 \\ f(a) \cdot f(b) &= 2n(\mathrm{mod}2) \cdot (2m+1)(\mathrm{mod}2) \\ &= 0 \cdot 1 \\ &= 0 \end{aligned}$$

$\therefore f(a \times b) = f(a) \cdot f(b)$

Since Equations 69 and 70 hold for all 3 cases, $f(x) = x(\mathrm{mod}2)$ forms a ring morphism between $\Re'$ and $\Re$. $\square$

5.2.2 Linear System Formulation of the RM Transform

In this section, an alternate definition of the RM transform in terms of the solution of a linear system of equations is provided. This formulation makes use of the isomorphic relation described in the preceeding and ultimately allows an efficient spectral computation method to be applied. The complement-free ring sum formulation of the canonical RM form for a three-variable function may be expressed as shown in Equation 71.

$$f(x) = r_0 \oplus r_1\dot{x}_1 \oplus r_2\dot{x}_2 \oplus r_3\dot{x}_3 \oplus r_4\dot{x}_1\dot{x}_2 \oplus r_5\dot{x}_1\dot{x}_3 \oplus r_6\dot{x}_2\dot{x}_3 \oplus r_7\dot{x}_1\dot{x}_2\dot{x}_3 \tag{71}$$

Where the dotted variables represent function inputs that are all uncomplemented. The r_i terms are Boolean constants that have value "1" or "0" and are in fact the RM spectral coefficients. Equation 71 contains operations and elements from the Boolean ring, $\Re$ (the omission of a binary operator between two consecutive variables in Equation 71 implies that the operation denoted by $\cdot$ occurs). Equation 71 is rewritten in the form given in Equation 72.

$$f(x) = r_0g_0 \oplus r_1g_1 \oplus r_2g_2 \oplus r_3g_3 \oplus r_4g_4 \oplus r_5g_5 \oplus r_6g_6 \oplus r_7g_7 \tag{72}$$

Where the r_i are Boolean constants described above, and each g_i is the AND (product) of a subset of the function's literals. Each particular realization of the complement-free ring sum of the RM form is given by a specified set of r_i values that will be referred to as the vector, $\underline{R}$. The set of functions, g_i, will be referred to as the "product functions" of $f(x)$. It is convenient to rewrite Equation 72 in vector matrix form as:

$$G\underline{R} = \underline{F} \tag{73}$$

Where the elements of the function vector, $\underline{F}$, and the transformation matrix, G, are known (or are easily computed), and the elements of the $\underline{R}$ vector specify the complement-free ring sum formulation of the RM canonical form. The column vectors formed from all possible outputs of the product functions, g_i, are concatenated to form the coefficient matrix, G, and the corresponding function outputs are concatenated to form the function vector, $\underline{F}$.

This formulation differs from that typically used to compute the RM spectrum of a logic function. As described in [40] [49], the RM spectrum may be computed as given in Equation 74 where all operations are performed using those defined in $\Re$.

$$\underline{R} = G\underline{F} \tag{74}$$

The reason Equation 74 holds is because $G = G^{-1}$ in the ring $\Re$. However $G \neq G^{-1}$ in the ring, $\Re'$, therefore the solution of $G\underline{R} = \underline{F}$ is required when real valued arithmetic is used.

The RM transformation matrix was derived in the previous section, however the following definition is useful since it defines the matrix in terms of the solution of the linear equation shown in 73. This definition is for the polarity-$2^n - 1$ matrix, however all of the results hold for matrices of any arbitrary polarity since they may be formed by reordering the rows of the polarity-$2^n - 1$ matrix.

Definition 2 : *The matrix, G, is a concatenation of the output column vectors of the product functions, g_i. Each row of the matrix is considered to be the output vector of a constituent function, $f_c(x)$.*

Consider the small G matrix, denoted as G_2, where the subscript indicates the number of function inputs.

$$
\begin{array}{ccc}
 & \begin{array}{cccc} g_0 & g_1 & g_2 & g_3 \end{array} & \\
G_2 \;=\; & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} & \begin{array}{c} f_{c0} \\ f_{c1} \\ f_{c2} \\ f_{c3} \end{array}
\end{array}
$$

Where the product functions are defined as:

$$
\begin{aligned}
g_0 &= 1 \\
g_1 &= x_0 \\
g_2 &= x_1 \\
g_3 &= x_0 \cdot x_1
\end{aligned}
$$

And the constituent functions are defined as:

$$
\begin{aligned}
f_{c0} &= \overline{x}_0 \cdot \overline{x}_1 \\
f_{c1} &= \overline{x}_1 \\
f_{c2} &= \overline{x}_0 \\
f_{c3} &= 1
\end{aligned}
$$

Lemma 10 : *The coefficient matrix G is triangular with all $g_{ii} = 1$.*

Proof: The trivial matrix,

$$G_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{75}$$

is triangular.

Also, from Definition 2, G_2 is also seen to be triangular. Higher ordered matrices, G_n, may be recursively defined using G_1 as a kernel since all functions, g_i, are AND functions or literal values themselves. The following definition holds:

$$G_n = \begin{bmatrix} G_{n-1} & 0 \\ G_{n-1} & G_{n-1} \end{bmatrix} \tag{76}$$

Since G_1 and G_2 are triangular and G_n may derived in terms of G_{n-1}, by induction, G_n is also triangular since it is formed with sub-matrices, G_{n-1}, along its diagonal with the all zero sub-matrix in one quadrant. □

Next, we will show that the solution vector, $\underline{R}$, always contains components that are members from the field of positive and negative integers, Z. Where Z is defined as:

$$Z = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$$

Lemma 11 : *The solution vector, $\underline{R}$, in the equation, $G\underline{R} = \underline{F}$, contains only integer components.*

Proof: As mentioned previously, the coefficient matrix, G_n, contains only 1 and 0 elements. Also, this matrix is triangular with diagonals consisting of all 1's. Hence:

$$det(G_n) = 1$$

It is desired to solve the equation:

$$G\underline{R} = \underline{F}$$

The solution to this equation is well known and may be expressed in terms of the cofactors of G as:

$$\begin{aligned} \underline{R} &= G^{-1}\underline{F} \\ &= [det(G)]^{-1}[cofactor(G)]\underline{F} \\ &= [1][cofactor(G)]\underline{F} \end{aligned}$$

Since the cofactor matrices are formed with no division operations, the matrix, $[cofactor(G)]$, contains only integers. Since the matrix, G, always has a determinant of 1, the solution vector, $\underline{R}$, is formed as a vector-matrix multiplication of the integer matrix, $[cofactor(G)]$, and the integer vector, $\underline{F}$. Hence the solution vector always contains integer components. $\square$

This lemma establishes the fact that the solution of Equation 73 is performed over the Boolean ring, $\Re'$. This result along with the previous definitions and lemmas allow the following theorem to be stated and proved.

Theorem 3 : *Given any Boolean function, f, and a binary vector, $\underline{B} = [b_1, b_2, b_3, \ldots, b_n]^T$ such that $f = b_1 g_1 \oplus b_2 g_2 \oplus \ldots \oplus b_n g_n$, then there exists a vector, $\underline{R}$ with each $r_i \in Z$, such that $G\underline{R} = \underline{F}$ and $\underline{R}(\mathrm{mod}2) = \underline{B}$. Where $\underline{F}$ is the binary vector formed from the output column of the truth table of the function, f.*

Proof: From Lemma 11, $\underline{R}$ has the property that each $r_i \in Z$. From Lemma 9, the function, $f(x) = x(\mathrm{mod}2)$ forms a ring morphism between $\Re'$ and $\Re$. Hence, the application of $f(x)$ to each component in the vector, $\underline{R}$, isomorphically maps $\underline{R}$ to $\underline{B}$.

□

Corollary 2 : *Every Boolean function has a unique complement-free ring sum form of the RM expansion.*

Proof: From Lemma 10 it was proven that all G_n matrices are triangular. From Definition 2 it was noted that all $g_{ii} = 1$. Hence the determinant of the coefficient matrices is 1, guaranteeing a unique matrix inverse exists. Since $\underline{R}$ and $\underline{B}$ from Theorem 3 are isomorphic and $\underline{R}$ is unique due to the existence of a unique inverse of G then $\underline{B}$ also is unique. □

This section has presented the mathematical justification computing the RM coefficients using real arithmetic. This result allows the efficient spectral calculation method formulated in Chapter 2 to be extended to the RM case.

5.2.3 Example of the Computation of the RM Spectrum

Using Real Arithmetic

Consider the Boolean function specified by the truth table in Figure 5.2.

x_2	x_1	x_0	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Figure 5.2. Truth table of example function to be synthesized

The resulting matrix equation is given as:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \underline{R} \quad = \quad \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Solving this system in the real field yields the following solution vector:

$$\underline{R}^T \quad = \quad [\ 1 \quad 0 \quad -1 \quad 0 \quad 1 \quad 1 \quad -1 \quad 0\ ]$$

Mapping this vector, from $\Re'$ to $\Re$ using $f(x)$, we obtain:

$$\underline{\tilde{R}^T} = [\ 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0\ ]$$

Each element of the solution vector corresponds to a specific r_i resulting in a realization of the RM form as given in Equation 71. The resulting function is:

$$f(x) = 1 \oplus x_1 \oplus x_2x_1 \oplus x_2x_0 \oplus x_1x_0 \tag{77}$$

5.3 Efficient Computation of the RM Spectral Coefficients

In the development given in the preceding subsection, it was shown that a RM spectral coefficient may be obtained by first computing a value in $\Re'$ (denoted as r') and then mapping it to $\Re$. Unfortunately the formation and solution of the matrix equation to compute the values in $\Re'$ has a complexity similar to computing the coefficient directly using $GF(2)$ arithmetic. However, the r' value can be computed without resorting to solving a matrix equation by exploiting its relation to the output probability of a circuit. Before this result is derived, the following definition is helpful.

Definition 3 *A Boolean function, $f \cdot f_c$, which is composed as the AND of a function to be transformed, f, and a constituent function, f_c, is termed the composition function and is denoted by f_{comp}.*

Definition 3 allows Lemma 12 to be easily stated and proven.

Lemma 12 *The real value, r', is directly proportional to the output probability of the Boolean function, f_{comp}.*

Proof: The value, r', can be viewed as the inner product of the output vector of the function to be transformed, f, and some constituent function, f_c. Since each element in these two output vectors is either 1 or 0, each partial product of the inner product is also 1 or 0. In fact, the partial product value of 1 only occurs when both f and f_c produce an output of 1 for a common given set of input values. Thus, r' is the total number of times the composition function, $f_{comp} = f \cdot f_c$, produces an output of 1. If the output probability of f_{comp} is known (denoted by $\wp\{f_{comp}\}$) then r' is easily computed as shown in Equation 78.

$$r' = 2^n \times \wp\{f_{comp}\} \tag{78}$$

Therefore r' is directly proportional to the output probability of the composition function with a constant of proportionality of 2^n where n is the number of primary inputs. □

It is now clear that the value of r' can be obtained without computing an inner product if the output probability is known. This leads to the result that allows the efficient spectral computation technique to be applied to the case of the RM spectrum.

Theorem 4 *A RM spectral coefficient, r, may be directly computed using the value of the output probability of f_{comp}.*

Proof:

From Lemma 9 it was shown that a particular RM coefficient can be computed from a real value as:

$$r = r'(\text{mod}2) \tag{79}$$

From Lemma 12, the value, r' is related to the output probability, $\wp\{f_{comp}\}$. Substituting Equation 78 into Equation 79 yields the final result as given in Equation

80.

$$r = (2^n \wp\{f \cdot g_c\})(\mathrm{mod} 2) \tag{80}$$

$\square$

Theorem 4 provides the necessary theoretical relation for the implementation of a method for computing a RM spectral coefficient without requiring storage resources proportional to the size of a functions' truth table.

5.3.1 Example Computation

Before the results of the implementation of this method are given, a small example is provided so that the method may be clearly understood. As an example of the computation, consider the Boolean function described by Equation 81.

$$f(x) = \overline{x}_1\overline{x}_3 + x_1\overline{x}_2x_3 + \overline{x}_1x_2 + x_2\overline{x}_3 \tag{81}$$

The constituent function corresponding to the 7^{th} RM spectral coefficient for a polarity-$2^n - 1$ transform is given by Equation 82.

$$f_{c6}(x) = \overline{x}_3 \tag{82}$$

The output vector of Equation 82 is identical to the 7^{th} row of the polarity-$2^n - 1$ RM transformation matrix for a 3 variable function. The corresponding OBDD representations for $f(x)$ and $f(x) \cdot f_{c6}(x)$ are given in Figure 5.3.

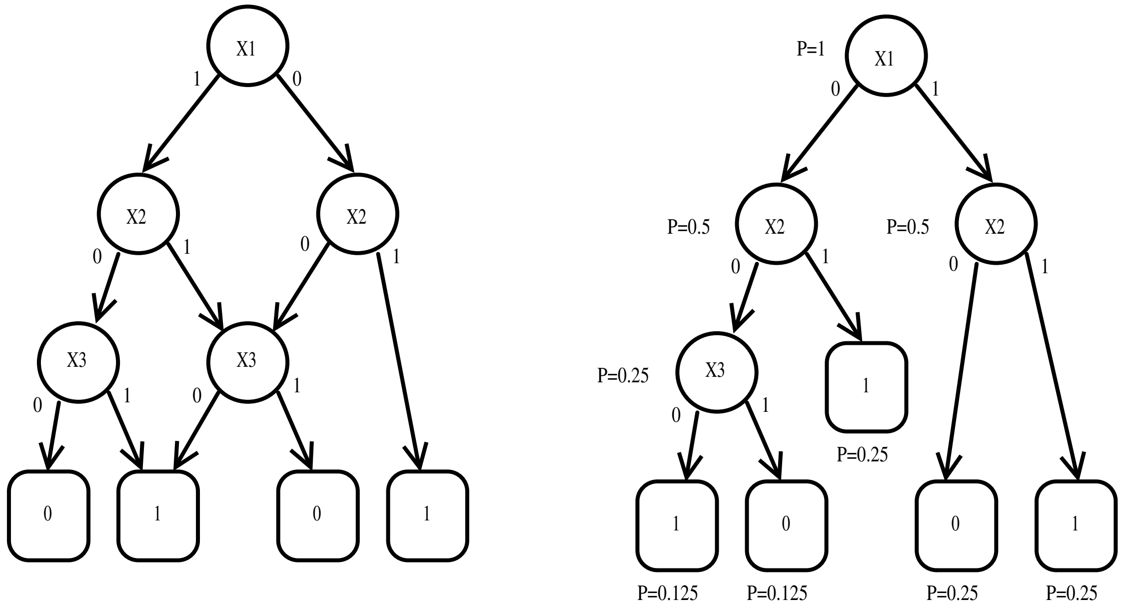


Figure 5.3. Binary decision diagrams of example function and composition function

The PAA algorithm is applied to the composition function, $f(x) \cdot f_{c6}(x)$, and the probabilities assigned to each node are shown in Figure 5.3. The required quantity is the sum of all probabilities at each terminal logic "1" node and is given as $\wp\{f(x) \cdot f_{c6}(x)\} = 0.625$. Substituting the probability value into Equation 80 yields the RM spectral coefficient as shown in Equation 83.

$$r_6 = [2^3 \times 0.625](mod2) = 1 \tag{83}$$

5.4 Implementation of the Synthesis System

A program for the efficient computation of the RM spectral coefficients was implemented using the C programming language. The BDD package developed by David Long was used for all BDD processing. The program receives the BDD description of the circuit to be transformed as input. Based upon the desired polarity number, a set of constituent functions corresponding to the rows of the RM transformation matrix are specified. For each computation of a spectral value, an OBDD representation of each composition function is formed. After the formation of the OBDD representing f_c, the $APPLY$ algorithm [31] is invoked thus forming the OBDD representing the composition function, f_{comp}. Next, the PAA is invoked resulting in the output probability which is converted to the RM spectral coefficient according to Equation 80.

Many logic functions are represented in a very compact form when they are expressed as BDDs [30] in contrast to the exponentially large size (in terms of number of inputs) required by a truth table, and hence an output vector. This fact allows the methodology implemented here to compute a RM coefficient with complexity $O(\|E_{comp}\|)$ where $\|E_{comp}\|$ is the number of edges in the BDD representing f_{comp}. However, there are some functions that have a number of BDD nodes that are comparable to the number of truth table values [45] [46]. For these cases, this methodology degenerates to having a complexity equivalent to performing the matrix calculations. Fortunately, many functions of practical importance are described with far fewer BDD nodes than truth table entries.

The overall time complexity of the approach is $O(\|E_f\| \times \|E_{g_c}\|)$ for each RM coefficient since each application of the $APPLY$ algorithm visits each node in the BDD representing f a number of times equal to the size of the OBDD representing g_c. Since the constituent functions for the RM transform are extremely simple, the OBDD representing f_{comp} is generally comparatively small with respect to the OBDD for f. Most of the time $\|E_{comp}\| < \|E_f\| \times \|E_{g_c}\|$ so the PAA algorithm requires relatively little computation time. The storage requirement also has a complexity of $O(\|E_{comp}\|$ since only the storage of the composition function BDD is required.

Instead of computing and storing a probability value (which lies in the interval $[0,1]$), the numerator of the probability value is stored (which lies in the interval, $[0,2^n]$). This modification alleviates possible underflow errors, however the numerator value can overflow for large values of n. The potential overflow problem was addressed by storing a mantissa and exponent value at each node. The exponent value is a power of 2 and is always as large as possible. Therefore, a node at the k^{th} level in the OBDD always has an exponent value that is at least $n-k$. This observation leads to an interesting fact as given in Lemma 13.

Lemma 13 *A necessary condition for a RM spectral coefficient, r_i, to have a value of 1 is that the OBDD representing the composition function, $f \cdot f_{ci}$, must contain a path consisting of n non-terminal nodes and a logic-1 valued terminal node.*

Proof: Since the initial node is labeled with an exponent value of n and the probability value of a node is halved to obtain the value of subsequent nodes, each node at the k^{th} level of the OBDD will have an exponent that is at least $n-k$ in value. Since the coefficient, r_i is computed as the modulo-2 value of the product of the probability value and 2^n as given in Equation 80, the exponent of the terminal "1" node must be equal to zero in order for $r_i = 1$. An exponent value can never reach zero unless n nodes in a single path have been traversed, allowing the initial nodes' exponent value, n, to be decremented n times. $\square$

5.4.1 Experimental Results

This implementation was tested using some of the $ISCAS85$ benchmark circuits as input. The following tables contain some results of the computations. For ease of description, the subset of RM coefficients corresponding to constituent functions containing m literals is referred to as the m^{th}-order subset of RM coefficients. The particular polarity of each literal is determined by the polarity number of the corresponding RM transformation matrix which defines the constituent functions.

Table 5.1 contains the results of the 0^{th}-order ($f_c = 1$) RM coefficients for several different $ISCAS85$ circuits. The label assigned to the particular output in the netlist description is given followed by the number of inputs, the size of the OBDD in terms of vertices, and, the CPU time the computations required on a DECstation 5000. The CPU time also includes the time required to convert the $ISCAS$ netlist into an OBDD representation. Since the OBDD for f_{comp} is exactly the same as the OBDD for f in the case of the 0^{th}-order RM coefficient, only $\|f\|$ is given in Table 5.1.

Table 5.1.–O^{th} order RM coefficients for various netlists

Netlist	Output	# Inputs	$\|f\|$	System Time (sec)
$c432$	$223gat$	18	2791	0.2
$c880$	$850gat$	29	196	< 0.1
$c2670$	188	14	14	0.1
$c3540$	399	26	185	0.1
$c6288$	$4946gat$	24	17057	7.6
$c7552$	376	28	52	0.3

Table 5.2 contains some of the 1^{st}-order RM coefficients for the circuit, $c432$, output $329gat$ which has 27 inputs and 2792 vertices in its OBDD representation. The entire amount of CPU time required for the computation of Table 5.2 was 1.0 seconds on a DECstation 5000 including the time required to parse the input netlist and construct the OBDD.

Table 5.2.–1^{st} order RM coefficients
for output $329gat$ of $c432$

Constituent Function	$\|f \cdot f_c\|$	Spectral Coefficient
$\overline{1gat}$	1387	0
$\overline{4gat}$	2156	1
$\overline{11gat}$	1389	0
$\overline{17gat}$	2160	1
$\overline{24gat}$	1393	0
$\overline{30gat}$	2168	1
$\overline{37gat}$	1401	0
$\overline{43gat}$	2184	1
$\overline{50gat}$	1417	0
$\overline{60gat}$	1488	1
$\overline{73gat}$	1558	1
$\overline{86gat}$	1712	1
$\overline{99gat}$	2028	1
$\overline{112gat}$	2027	1

This code was incorporated into a prototype synthesis tool that uses OBDD descriptions of Boolean functions as input and produces $Verilog$ netlists in generalized RM form as output. The tool was implemented in two main modules. The first module receives the BDD description and the polarity number of the desired GRM form as input. Upon receiving these inputs, a BDD representation of each composition function is formed sequentially. After the formation of the f_c BDD, the $APPLY$ algorithm [31] is invoked forming the BDD representing the composition function, f_{comp}. Next, the PAA is invoked resulting in the output probability which is converted to the RM spectral coefficient according to Equation 80. Each time a coefficient is computed, it is passed to the second module which generates a portion of the output netlist.

The second portion of this tool served as the netlist generator. Although the

Verilog hardware description language (HDL) was chosen as the output medium, any netlist format would have been possible. An overall block diagram of the tool is given in Figure 5.4.

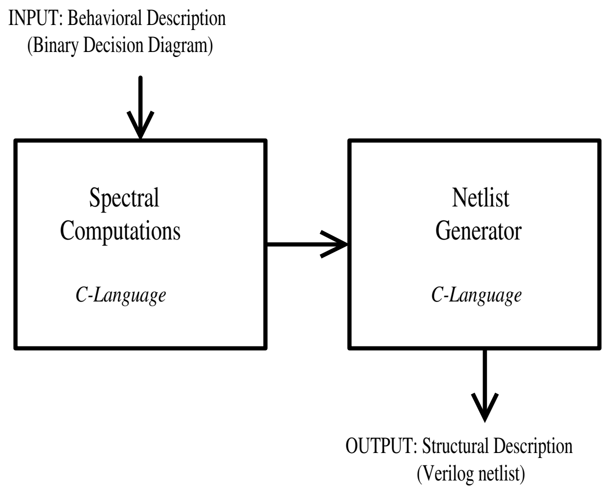


Figure 5.4. Block diagram of the generalized RM translation tool

The synthesis tool has successfully synthesized several different functions of varying complexity. The first circuit is a small 5-input function described by the following *Verilog* module:

Example 1:

```
module exm1beh (f, inp1 , inp2 , inp3 , inp4, inp5 );

 output f;
 input inp1,inp2,inp3,inp4,inp5;

 assign{f} = !(inp2 || (inp1 && inp3)) || (!inp4) && inp1 || inp5 && (!inp2);

endmodule
```

This behavioral description was used to generate the OBDD for the synthesis computations. The synthesis computations and netlist generator modules were invoked next. The resulting output netlist file follows:

```
module rmcircuit (f, inp1 , inp2 , inp3 , inp4 , inp5 );

 output f;
 input inp1,inp2,inp3,inp4,inp5;

 and   g_0 ( wir0 , inp1, inp2);
 and   g_1 ( wir1 , inp1, inp3, inp4);
 and   g_2 ( wir2 , inp1, inp2, inp4);
 and   g_3 ( wir3 , inp1, inp2, inp3, inp4);
 and   g_4 ( wir4 , inp1, inp3, inp4, inp5);
 and   g_5 ( wir5 , inp1, inp2, inp3, inp4, inp5);
 xor   g_6 ( f, 1'b1,  inp2, wir0, wir1, wir2, wir3, wir4, wir5);
endmodule
```

□

The next example is a slightly larger circuit. The behavioral description for this circuit follows:

Example 2:

```
module exm2beh (f, inp1 , inp2 , inp3 , inp4, inp5, inp6, inp7, inp8 );

 output f;
 input inp1,inp2,inp3,inp4,inp5,inp6,inp7,inp8;

 assign{f} = !(inp2 || (inp1 && inp3)) || (!inp4) && inp1 || inp5 && (!inp2)
             || inp3&&(!inp8 || inp6&&inp7);

endmodule
```

The resulting output structural description is:

```
module rmcircuit (f, inp1 , inp2 , inp3 , inp4 , inp5 , inp6 , inp7 , inp8 );

 output f;
 input inp1,inp2,inp3,inp4,inp5,inp6,inp7,inp8;

 and   g_0 ( wir0 , inp1, inp2);
 and   g_1 ( wir1 , inp2, inp3);
```

```
 and   g_2 ( wir2 , inp1, inp2, inp3);
 and   g_3 ( wir3 , inp1, inp2, inp4);
 and   g_4 ( wir4 , inp2, inp3, inp8);
 and   g_5 ( wir5 , inp1, inp2, inp3, inp4);
 and   g_6 ( wir6 , inp1, inp3, inp4, inp8);
 and   g_7 ( wir7 , inp1, inp2, inp3, inp8);
 and   g_8 ( wir8 , inp2, inp3, inp6, inp7, inp8);
 and   g_9 ( wir9 , inp1, inp3, inp4, inp5, inp8);
 and   g_10 ( wir10 , inp1, inp3, inp4, inp6, inp7, inp8);
 and   g_11 ( wir11 , inp1, inp2, inp3, inp6, inp7, inp8);
 and   g_12 ( wir12 , inp1, inp2, inp3, inp4, inp5, inp8);
 and   g_13 ( wir13 , inp1, inp3, inp4, inp5, inp6, inp7, inp8);
 and   g_14 ( wir14 , inp1, inp2, inp3, inp4, inp5, inp6, inp7, inp8);
 xor   g_15 ( f, 1'b1,  inp2, wir0, wir1, wir2, wir3, wir4, wir5,
wir6, wir7, wir8, wir9, wir10, wir11, wir12, wir13, wir14);
endmodule
```

□

These results show the output of the synthesis tool. The output form in these two examples is an ESOP form with all inputs uncomplemented. This is the polarity-$2^n - 1$ GRM form. Other unique ESOP forms can also be computed by changing the polarity number which is used to specify the ordering of the constituent functions. Although all RM transformation methods suffer from the requirement of necessarily computing an exponential number of coefficients, this method has provided a very efficient way of computing each coefficient. In addition, it has been shown in Lemma 13 that OBDDs representing f_{comp} functions that do not have a maximal path will always yield a coefficient value of 0. This fact may be exploited by allowing the spectral coefficients to be computed only when the f_{comp} contains a maximal path.

CHAPTER 6

CONCLUSIONS AND AREAS OF FUTURE RESEARCH

The research presented in this dissertation has demonstrated the viability of spectral methods for digital logic synthesis. Although spectral based methods have been developed in the past, their use was limited to very small circuits deeming the approach impractical. By devising a technique to efficiently compute a spectral coefficient, the computational hurdles of past methods have been overcome.

6.1. Conclusions

Spectral based synthesis methodologies typically impose demanding computational requirements. By using extremely compact representations of Boolean functions, each spectral coefficient can be computed very efficiently. Further, by developing a synthesis system that utilizes a very small number of coefficients, excessive storage requirements are avoided in addition to limiting the number of required coefficient computations.

The efficient spectral calculation technique is also very general requiring no special transformation matrix properties. This generality allowed a synthesis methodology to be developed that combines the optimization and technology mapping phases into a single processing step. By building the transformation matrix as a collection of constituent functions that correspond to the particular logic cells in a library, the circuit resulting from the maximum correlation synthesis method is already described in terms of actual layout cells.

The extension of the efficient spectrum computation technique allowed it to be applied to the Reed-Muller forms of spectra. The desirable properties of the Reed-Muller family of circuits such as testability and compactness has increasingly lead to their inclusion in modern designs. Unfortunately, the current state of CAD tools for Reed-Muller design has lagged far behind those that target other circuit forms. Therefore, computational advances in Reed-Muller circuit synthesis are especially relevant in the area of automated CAD research.

6.2. Contributions to Synthesis Methodologies

Most common logic synthesis tools in use today are based upon an area minimization criteria. The trend in VLSI technology is that the required substrate area for a given design tends to decrease by 1/2 every 4 to 5 years. While physical limitations will eventually halt this trend, the short term result has caused designers to worry more about timing optimization and less about area conservation. By incorporating a strict critical path minimization criteria into the synthesis method described in Chapter 3, this problem has been addressed without an extreme sacrifice in required area.

Many of todays' CAD tools are used to aid designers in supplying semi-custom, standard cell solutions to customer requirements. Typically, these tools perform the optimizations in a technologically independent fashion (using generic logic gates) and are then faced with the daunting task of mapping the design in terms of the standard cell library. By exploiting the fact that any generalized transform may be computed using the efficient technique, a synthesis methodology was described in Chapter 4 that unifies the optimization and technology mapping phases. This formulation allows the designers' choice of the standard cell library to inherently define the spectral transformation.

Although heuristic-based systems for Reed-Muller circuit design do exist, they are primarily the products of academic research and lack the maturity needed for inclusion in commercial products. The designer is generally forced to use an exhaustive method for synthesizing a Reed-Muller circuit structure. Unlike the more common canonical forms, no efficient Karnaugh mapping or Quine-McCluskey tabulation methods exist for the Reed-Muller case. Thus, the designer is faced with performing cumbersome Boolean algebraic manipulations, or, computing the Reed-Muller spectrum. The exponential complexity of spectral calculations generally precludes the use of the latter approach. However, by extending the efficient spectral computation method to handle the Reed-Muller coefficients, it is now practical to consider the spectral approach to Reed-Muller circuit design.

6.3. Future Research Directions

Future research areas include the extension and improvement of the methods presented here as well as the application of spectral techniques to other areas of digital systems engineering. The use of BDDs to construct probability expressions is a fundamental result that allows the methods of probability and statistics to be applied to many areas that involve discrete, and in particular, Boolean systems. The following subsections outline a few specific areas of research that will be investigated in the near future.

6.3.1. Extension to BDD Forms Other Than OBDD

Several extensions to the efficient spectral computation technique are worthy of investigation. Although most functions are represented in OBDD form much more compactly than when their algebraic or truth table descriptions are used, there are certain classes that require an exponential number of BDD vertices [44] [45]. This problem has recently been addressed through the development of indexed BDDs (IBDDs) [65]. The extension of the spectrum calculation method presented here to use IBDDs as input would further increase its value for CAD applications. However, in the IBDD form, nodes labeled with the same primary input can be encountered more than once in a single path. This means that the BDD traversal algorithm in the PAA will have to be modified to store additional intermediate data. A careful analysis of the amount of extra data required must be performed to analyze the feasibility of this modification.

The nature of a BDD is to represent a single output Boolean function in a compact manner. Many of todays logic synthesis tools have the desirable feature of handling multi-output circuits that depend on a common set of primary inputs. Recently, researchers have proposed a new form of BDD that represents multi-output functions called shared BDDs (SBDDs) [66]. The SBDDs could provide the means for extending the logic synthesis methodologies presented here to handle the multi-output circuit. In order to use SBDDs, efficient counterparts to the $APPLY$ and $RESTRICT$ operations defined for OBDDs will need to be developed.

6.3.2. Application to Low Power Design

As portable computing and communications devices increase in popularity, more emphasis is placed on designs that operate with reduced power consumption. Low power design techniques are currently a research topic of great interest. To date, the most successful approaches have utilized architectural modifications such as reduced operating voltages and frequencies, or, device level changes such as the development and use of low power standard cell libraries. The lack of effective methods at the logic design level have resulted in an area that is ripe for new research endeavors.

One possible approach is to relate the switching activity at internal circuit nodes in a CMOS circuit to power dissipation levels. In the research presented here, output probability computations have been described in detail. By computing switching probabilities at each node of a logic circuit as it is synthesized, it may be possible to formulate constraints that attempt to keep the probabilities very high or very low. If the switching probabilities at each node are not close to the value, 1/2, the overall switching activity, and hence, power dissipation levels are minimized.

6.3.3. Design Verification

Another area closely related to logic synthesis is that of design verification. As CAD tools mature, the circuit to be realized is represented at various levels of abstraction. Each level of abstraction subsequently becomes closer to the ultimate design description as the designer adds further specifications. The design verification system is tasked with ensuring that all representations of the circuit are functionally equivalent.

It has been proven that the design verification problem is NP-Complete [67]. For this reason, interest has been generated in the use and development of statistical verification systems that do not necessarily guarantee 100% functional equivalence, but do guarantee a high percentage of equivalence in a very short time. Since a set of spectral coefficients forms a unique signature of a logic function regardless of its form of representation, the efficient spectral computation method may be used to formulate a statistical verification technique. One aspect of this research area will be to determine which subset of spectral coefficients should be used to maximize the percentage of verification while keeping the total number of coefficients needed as low as possible.

It has been shown in past work that the Chow parameters can be used to partition all possible Boolean functions of n variables into a collection of disjoint subsets using the principles of NPN equivalence [16]. Two functions are said to be NPN equivalent if the output of one is identical to the output of the other when appropriate inputs are negated (N), permuted (P), and the output may or may not be negated (N). The number of of NPN classes is generally very large compared to all possible functions of n inputs. Thus, determining if two functions are NPN equivalent is analogous to verifying they are the same with some degree of error based upon the number of functions within the given NPN class.

6.3.4. Finite State Machine Synthesis

Before the logic synthesis task is invoked, most CAD systems execute a module responsible for finite state machine (FSM) partitioning and state assignment. Since Boolean expressions may be readily converted to output probability expressions, all transition relations in a FSM may be represented as probabilities. Thus, the FSM is transformed to a representation of a Markov chain.

The vast amount of results available from discrete stochastic systems theory can be used for tasks such as FSM partitioning and state assignment once the FSM has been transformed to a Markov chain model. For example, a logical partitioning paradigm may prove to be the restriction of high probability transitions within a single FSM and to partition by "cutting" the low probability transitions. This would especially be beneficial when very large FSMs are implemented using multiple FPGAs or PLDs since interchip state transitions can occur much more quickly than intrachip transitions.

References

[1] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* Mc-Graw-Hill, New York, New York, 1994.

[2] R. Rudell and A. L. Sangiovanni-Vincintelli. Espresso-mv: Algorithms for multiple-valued logic minimization. *Proceedings IEEE Cust. Int. Circ. Conf. (CICC-85)*, pages 230–234, May 1985.

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

[4] W. V. Quine. The Problem of Simplifying Truth Functions. *Am. Math. Monthly*, vol. 59, No. 8:521–531, October 1952.

[5] E. J. McCluskey. Minimization of Boolean Functions. *Bell System Tech. J.*, vol. 35:1417–1444, November 1956.

[6] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincintelli, and A. R. Wang. Mis: A multiple-level logic optimization system. *IEEE Trans. on CAD*, vol. CAD-6, no. 6:1062–1081, November 1987.

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

[8] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney. The Transduction Method - Design of Logic Networks Based on Permissible Functions. *IEEE Trans. on Comp.*, vol. 38, no. 10:1404–1424, October 1989.

[9] K. C. Chen and S. Muroga. SYLON-DREAM: A Multi-Level Network Synthesizer. *Proceedings ICCAD*, pages 552–555, 1989.

[10] K. C. Chen, Y. Matsunaga, S. Muroga, and Fujita M. A Resynthesis Approach for Network Optimization. *Proceedings DAC*, pages 458–463, 1991.

[11] H. Sato, Y. Yasue, Y. Matsunaga, and M. Fujita. Boolean Resubstitution with Permissible Functions and Ordered Binary Decision Diagrams. *Proceedings DAC*, 1990.

[12] S. Jeong, F. Somenzi, and T. Sasao (editor). *A New Algorithm For 0-1 Programming Based on Binary Decision Diagrams (Chap. 6) in Logic Synthesis and Optimization.* Kluwer Academic Publishers, Boston, Massachusetts, 1993.

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

[14] R. Lechner. Harmonic Analysis of Switching Functions. *in Recent developments in switching theory*, pages 121–228, 1971.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] R. L. Ashenhurst. The Decomposition of Switching Functions. *Proceedings of an International Symposium on the Theory of Switching*, pages 74–116, April 1957.

[21] V. M. Tokmen. Disjoint Decomposability of Multiple Valued Functions by Spectral Means. *Proceedings IEEE 10th Int. Symp. Mult. Valued Logic*, pages 88–93, 1980.

[22] 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.

[23] T. Damarla. Generalized Transforms for Multiple Valued Circuits and their Fault Detection. *IEEE Trans. Comp.*, vol. C-41, no. 9:1101–1109, September 1992.

[24] 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.

[25] 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.

[26] A. K. Susskind. Testing by Verifying Walsh Coefficients. *IEEE Trans. Comp.*, vol. C-32, no. 2:198–201, February 1983.

[27] 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.

[28] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Computation*, vol. 19:297–301, 1965.

[29] J. L. Shanks. Computation of the Fast Walsh-Fourier Transform. *IEEE Trans. Comp.*, vol. C-18:457–459, May 1969.

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

[31] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comp.*, vol. C-35, no. 8:677–691, August 1986.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] K. P. Parker and E. J. McCluskey. Probabilistic Treatment of General Combinational Networks. *IEEE Trans. Comp.*, vol. c-24:668–670, June 1975.

[37] 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.

[38] 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.

[39] 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.

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

[41] 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.

[42] C. Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell System Technical Journal*, vol. 38:985–999, July 1959.

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

[44] R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, vol. 24, no. 3:293–318, September 1992.

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

[46] R. Bryant. On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Applications to Integer Multiplication. *IEEE Trans. Comp.*, vol. 40, no. 2:205–213, February 1991.

[47] F. J. Hill and G. R. Peterson. *Computer Aided Logical Design With Emphasis on VLSI.* John Wiley & Sons, Inc., New York, New York, 1993.

[48] D. Bryan. The ISCAS85 Benchmark Circuits and Netlist Format. *ISCAS85 Benchmark Documentation*, 1985.

[49] M. Davio, J.-P. Deschamps, and A. Thayse. *Discrete and Switching Functions.* Mc-Graw-Hill, New York, New York, 1978.

[50] P. Wayner. Silicon in Reverse. *BYTE Magazine*, vol. 19(8), 1994.

[51] C. E. Shannon. Symbolic Analysis of Relay and Switching Circuits. *Trans. AIEE*, vol. 57:713–723, 1938.

[52] D. K. Pradham (editor). *Fault-Tolerant Computing Theory and Techniques Volume I.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[53] 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.

[54] S. Milak, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment. *Proceedings of ICCAD*, pages 6–9, 1988.

[55] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams. *Proceedings of ICCAD*, pages 2–5, 1988.

[56] T. Sasao and P. Besslich. On the complexity of mod-2 sum pla's. *IEEE Trans. Comp.*, vol. C-39, no. 2:262–266, February 1990.

[57] U. R. Rollwage. The complexity of mod-2 sum pla's for symmetric functions. *Proceedings IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 6–12, Spetember 1993.

[58] S. M. Reddy. Easily testable realizations for logic functions. *IEEE Trans. Comp.*, vol. C-21, no. 11:1183–1188, November 1972.

[59] A. Sarabi and M. Perkowski. Fast Exact Quasi-Minimal Minimization of Highly Testable Fixed-Polarity ANDXOR Canonical Networks. *Proceedings of the IEEE Design Automation Conference*, pages 30–35, 1992.

[60] D.E. Muller. Application of Boolean Algebra to Switching Circuit Design and to Error Detection. *IRE Trans. Elec. Comp.*, pages 6–12, September 1954.

[61] A. Mukhopadhyay and G. Schmitz. Minimization of EXCLUSIVE OR and LOGICAL EQUIVALENCE Switching Circuits. *IEEE Trans. Comp.*, C-19, no. 2:132–140, 1970.

[62] T. (editor) Sasao. *Logic Synthesis and Optimization.* Kluwer Academic Publishers, Boston, Massachusetts, 1993.

[63] T.C. Bartee, I.L. Lebow, and I.S. Reed. *Theory and Design of Digital Machines*. Mc-Graw-Hill, New York, New York, 1962.

[64] V.H. Larney. *Abstract Algebra A First Course*. Prindle, Weber, and Schmidt, Boston, Massachusetts, 1975.

[65] J. Jain, M. Abadir, J. Bitner, D. S. Fussell, and J. A. Abraham. IBDDs: An Efficient Functional Representation or Digital Cir cuits. *Proceedings of the European Design Automation Conference*, March 1992.

[66] S. Minato, N. Ishiura, and S. Yajima. Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation. *Proceedings of the ACM/IEEE Design Automation Conference*, pages 52–57, 1990.

[67] R. Wei and A. L. Sangiovanni-Vincentelli. PROTEUS: A Logic Verification System for Combinational Circuits. *Proceedings of the International Test Conference*, pages 350–359, 1986.