SIMPLIFIED THEORY OF BOOLEAN FUNCTIONS

by

Patrick Kam Lui
B.Sc., University of Manitoba, 1981
M.Sc., University of Manitoba, 1983

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

ACCEPTED
FAC[illegible]

Requirements for the Degree of

ACCEPTED
FACULTY OF GRADUATE STUDIES
DEAN
DATE 1990-11-15

DOCTOR OF PHILOSOPHY

We accept this thesis as conforming
to the required standard

Dr. J.C. Muzio, Supervisor (Department of Computer Science)

Dr. J.A. Ellis, Departmental Member (Department of Computer Science)

Dr. F.D.K. Roberts, Departmental Member (Department of Computer Science)

Dr. G.A. Beer, Outside Member (Department of Physics)

Dr. V.K. Bhargava, Outside Member (Department of Electrical & Computer Engineering)

Dr. D.K. Pradhan, External Examiner (Dept. Elec. & Comp. Eng., U. of Massachusetts at Amherst)

© PATRICK KAM LUI, 1990

University of Victoria

All rights reserved. Thesis may not be reproduced in whole or in part, by
mimeograph or other means, without the permission of the author.

ABSTRACT

A new, intuitive approach to the study of a Boolean function using its set of *parities* of *subfunctions* called the *parity spectrum* is presented. This approach simplifies the classical theory of *Boolean difference*, and serves to unify and extend a number of previous results on the *modulo-2 logic design* and *fault detection* of digital logic networks. Fundamental properties of the parity spectrum are established. They are instrumental in developing the principal results.

New algebraic and geometric representations for *fixed polarity* and *fixed basis modulo-2 canonical expansions (FPEs and FBEs)* are obtained by identifying coefficients in these expansions to subfunction parities in the parity spectrum. These representations offer new insights into the underlying structure of modulo-2 canonical expansions as well as algorithms that manipulate them.

Boolean matrix transforms among the parity spectrum, the FPEs, and the FBEs are described in a unified manner using *Kronecker products*, and efficient recursive algorithms derived for these and other transforms are applied to two different approaches to the *minimization* of FPEs and FBEs.

By verifying subfunction parities from the parity spectrum of the function implemented by a digital logic network, the generalized *constrained parity testing* technique is developed. It is considered for detecting *multiple stuck-at faults* in *single-output combinational networks*.

Supervisor: **Prof. Jon C. Muzio**

Examiners:

Dr. J.C. Muzio, Supervisor (Department of Computer Science)

Dr. J.A. Ellis, Departmental Member (Department of Computer Science)

Dr. F.D.K. Roberts, Departmental Member (Department of Computer Science)

Dr. G.A. Beer, Outside Member (Department of Physics)

Dr. V.K. Bhargava, Outside Member (Department of Electrical & Computer Engineering)

Dr. D.K. Pradhan, External Examiner (Dept. Elec. & Comp. Eng., U. of Massachusetts at Amherst)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ACKNOWLEDGEMENTS

Dr. J.C. Muzio has been my supervisor throughout my graduate studies, and has nourished me with his technical expertise, research inspirations, and personal wisdom. His guardianship, support, and encouragement has been a tremendous driving force throughout the course of this research. His patience and kindness, especially while this dissertation is being written off-campus in Ottawa, is warmly felt and treasured. My deep appreciation and sincere gratitude for his invaluable supervision is beyond the mere expression of words.

I am thankful to the many who have assisted my research through their helpful comments and generous support. In particular, I would like to thank Dr. M. Cheng, Dr. J. Ellis, Dr. N. Horspool, Mr. J. Kowalski, Mr. J. Lee, Dr. M. Levy, Mr. V. Liu, Dr. R. Probert, Dr. F. Ruskey, Dr. M. Serra, Dr. W. Wadge, Mr. M. Whitney, and Mr. A. Wong for their help. I would also like to thank the Computer Science Department in University of Ottawa for providing the computing facilities in which this dissertation is completed. Finally, I would like to thank the members of my Examining Committee for their careful review of this dissertation.

This research has been financially supported by a University of Victoria Graduate Fellowship, and by NSERC, grant no. A5711.

CHAPTER 1

INTRODUCTION

1.1. A Simplified Approach

Modern day digital logic systems manipulate discrete elements of information encoded in *binary* form. Data is represented by physical quantities called *binary signals* and data processing is performed on these signals by means of *binary logic elements*. The mathematical counterpart of a physical quantity having precisely two stable states is a *binary variable*, and the manipulation of binary signals can be expressed in terms of *Boolean (or switching) functions* in binary variables.

In a paper[Ake59] entitled *On a Theory of Boolean Functions*, Akers applies the concept of *Boolean difference* for defining and examining various formal properties of a Boolean function. This Boolean difference is shown to correspond in many respects to the finite-difference operator and accordingly an analogy is drawn between the theories of Boolean and ordinary functions. However, there are two major problems with Boolean difference. Firstly, the expressions often involve complex algebraic notation and derivations. Secondly, it is difficult to grasp the intuitive meaning of Boolean difference. It appears that these difficulties stem from using an ordinary function approach which may be too general and notationally too cumbersome for the purpose of studying practical engineering applications of Boolean functions.

In this dissertation, we introduce a new approach to the study of Boolean functions using its *parity spectrum*, which is the set of *parities* of all of its *subfunctions*. The new approach simplifies and unifies a number of existing results in the design, analysis, and testing of digital logic networks implementing Boolean functions, and allows these results to be generalized. Using an intuitive interpretation of the values

of Boolean differences as subfunction parities in the parity spectrum, properties of Boolean functions previously derived from Boolean differences are re-expressed with simplified notation and proofs, and new properties are derived. Applications of the theory of parity spectrum to the *modulo-2 logic design* and *fault detection* of digital logic networks are investigated. This chapter introduces these applications and outlines the results obtained in our investigation.

1.2. Modulo-2 Logic Design

1.2.1. Motivation

Following the classic work of Shannon[Sha38] on the theory of *switching algebra*, the design of digital logic systems is based heavily upon the simplification of *inclusive-or sum-of-product expressions* of Boolean functions and the implementation of these expressions using *AND/OR/NOT logic gates*. However, the work of Reed[Ree54] and Muller[Mul54] on *exclusive-or (modulo-2)* sum-of-product expressions has initiated an alternative design approach using *XOR* gates. In practice, it is well-known that many useful circuits such as arithmetic units and parity checkers are heavily XOR oriented and it is more economical to implement their modulo-2 expressions. Some authors[Eve67, Muk70, Rob82] even conjecture that it is generally more economical to base logic design on modulo-2 expressions rather than conventional inclusive-or expressions. Recent empirical results in [Sas90] support this view by showing that programmable logic arrays (PLAs) implementing XOR sum of products of randomly generated functions require, on the average, fewer *product terms* than standard PLAs implementing conventional inclusive-or sum of product expressions. Furthermore, digital networks realized by XOR gates have the feature of being *easily tested*[Red72, Red73, Sal75, Pra78] because a change in an input to an XOR gate is always propagated to the gate output. In contrast, implementations using AND/OR gates require specific input patterns to establish a *sensitized path*[Mic86] for a change to be propagated.

The above advantages of modulo-2 logic design motivate the study of modulo-2 expressions and their simplifications in this research.

1.2.2. Modulo-2 Canonical Expansions

Previous research on modulo-2 logic design emphasizes on the simplification of the modulo-2 expressions of a Boolean function. Because the underlying mathematical structures of these expansions are not well understood, results have been presented in vastly different notations and terminologies. Consequently, much confusion is generated and duplication in reporting appears inevitable.

Fundamental to the study of modulo-2 expressions are the *modulo-2 canonical expansions*. These expansions are said to be *canonical* because, under pre-defined conditions, each expansion is a unique representation for a Boolean function. The best known canonical form is called the *complement-free (positive) Reed-Muller canonical expansion*, first considered by Reed[Ree54] and Muller[Mul54] for applications to logic design and to error correcting codes. The characteristic of this canonical expansion is that each binary variable appears only in its normal uncomplemented form. By allowing variables to appear complemented or uncomplemented but not both, Akers[Ake59] arrives at a more general set of 2^n canonical forms, where n is the number of variables of the considered function. These expansions are called the *fixed polarity modulo-2 canonical expansions*, or simply the *FPEs*, of a Boolean function. The set of 2^n FPEs is further generalized by Bioul et al.[Bio73] into 3^n canonical forms, called *fixed basis modulo-2 canonical expansions (FBEs)*, by allowing some variables to *always* appear in both complemented and uncomplemented form. A part of this research investigates the properties and structures of FPEs and FBEs, as well as the matrix transforms and simplification procedures for these expansions.

Various authors[Ree54, Ake59, Dav71, Bio72, Bio73, Dav78] apply a Boolean difference approach to derive coefficients in FPEs and FBEs using complex algebraic notations and proofs. This research reveals a simpler derivation by identifying these coefficients with subfunction parities in the parity spectrum of the Boolean function. Thus a new and simplified algebraic representation for FPEs and FBEs is obtained.

Furthermore, the intuitive characteristics of subfunction parities give rise to a new geometric construction called the *parity hypercube*, or *P-cube*, representation of a Boolean function. A graphical display of the subfunction parities in the parity spectrum, the P-cube geometrically represents all the FPEs and FBEs of the Boolean function, and visually highlights the relationship among coefficients within an FPE or FBE as well as the relationship among all the FPEs and FBEs. It is an exposition of many interesting properties of a Boolean function that are not previously evident, and provides visual interpretations for algorithms that manipulate Boolean functions and their modulo-2 canonical expansions.

1.2.3. Transforms and Minimization

Many authors (e.g. [Cal61, Lec63, Dav71, Swa72, Bio73, Dav78, WuC82, Bes83, Zha84]) consider transforms among FPEs and FBEs and their applications to the simplification of these expansions. In this research, these transforms are unified and generalized. As proposed by Lechner[Lec63], these transforms are described as *Kronecker products* of elementary Boolean matrices since this approach leads to efficient algorithms for carrying out the transforms. Also described are all the possible transforms between the parity spectrum and the FPEs and FBEs, again using a Kronecker matrix product approach. Recursive procedures implementing the various transforms are also included.

The *weight* of an FPE or FBE is its number of non-zero product terms. In this dissertation, the simplification of FPEs and FBEs for a Boolean function is considered by deriving the expansion(s) containing the minimal weight. Two minimization approaches are considered. The first approach explicitly generates all the FPEs or FBEs and evaluates each of their weights individually. In the second approach, the FPEs and FBEs are not explicitly generated. Rather, their weights are computed simultaneously from the parity spectrum by means of Boolean matrix transforms also described as Kronecker products. Computationally, the first approach requires less space, while the second approach is faster.

1.3. Fault Detection

The ever-increasing complexity of integrated circuits has lead to renewed interest in the search for new, cheaper testing techniques for detecting faults and locating faulty components in a digital logic system. Using the parity spectrum, this research also considers the detection of faults in *single-output irredundant combinational networks* implementing Boolean functions. The *multiple stuck-at fault model*[Eld59, Bos71] is assumed, although the proposed testing technique can be applied to any other logic fault model.

In the traditional test vector based method, input patterns are applied to the *network under test (NUT)* and the output responses are verified one by one. Any discrepancy indicates a fault. This technique has given way to the more recent *compaction testing*[McC85] techniques, which reduce the volume of the responses by compacting them into a bit vector called a *signature.* Several compact testing techniques[Car82, Ake88, Dam89] verify the parity or subfunction parities of the Boolean function implemented by the NUT. These are special cases of the *constrained parity testing* technique developed in this research. Under this generalized technique, a signature is formed from any subset of subfunction parities from the parity spectrum of the implemented function. During testing, each subparity in the the signature is collected by constraining a subset of the inputs to the NUT to 0's and 1's and applying all combinations of binary values to the remaining inputs. Since a test which constrains all inputs is equivalent to a test in the traditional test vector method, constrained parity testing is also a generalization of the test vector method. Compared with other techniques, constrained parity testing offers many practical advantages such as versatility, flexibility, low test volume, low test time, high fault coverage, and reduced fault simulation and test generation costs.

1.4. Outline of Remaining Chapters

Chapter 2 provides the necessary background, notation and terminology.

In chapter 3, the parity spectrum is introduced and its relationship to Boolean difference is examined. Fundamental properties of the parity spectrum are

established. These properties are instrumental in the development of the applications presented in Chapters 4 to 6.

The relationship of the parity spectrum and coefficients of the fixed polarity and fixed basis expansions (FPEs and FBEs) are derived in Chapter 4, and the parity hypercube (P-cube) is introduced as a new geometric representation for the parity spectrum, the FPEs and the FBEs.

Using a Kronecker matrix product approach, Chapter 5 derives all possible transforms among the parity spectrum, the FPEs and the FBEs. Together with transforms that compute the weights of FPEs and FBEs from the parity spectrum, algorithms are described to derive the FPE or FBE with the least number of terms.

The constrained parity testing technique is developed in Chapter 6. By considering *vacuous* faults, which include over 99 percent of all multiple stuck-at faults, an approach is described to derive a signature for testing all multiple stuck-at faults in a combinational logic network with small number of fanout lines. For non-vacuous stuck-at faults, a method is described for test generation using *subnetworks* of the network under test (NUT). A hybrid scheme by combining with *syndrome testing* is also considered to detect all single stuck-at faults and most multiple stuck-at faults in *internally unate networks* without the need for expensive fault simulation.

Finally, Chapter 7 concludes this dissertation with a list of the contributions by this research and a discussion of possible directions for future research.

For notational convenience, tables, figures, algorithms or equations within each chapter are numbered sequentially and referred to in others chapters using their chapter number as prefix. For example, Table 1 in Chapter 2 is referred to in other chapters as Table 2–1.

CHAPTER 2

BACKGROUND

2.1. Introduction

This chapter provides the background, notation, and terminology for the remaining chapters. Algebraic and geometric representations for *Boolean functions* are described. Properties of the *exclusive-or (XOR)* operator are presented. *Fixed polarity* and *fixed basis modulo-2 canonical expansions* are defined. Background on *Kronecker products* of Boolean matrices, and *binary* and *ternary reflected Gray codes* is also covered. Finally, an overview of the research on *fault detection techniques* for digital logic networks is presented.

2.2. Boolean Functions

In 1854, Boole[Boo54] introduced an orderly treatment of logic and developed an algebraic system now called *Boolean algebra.* In 1938, Shannon[Sha38] applied the classical propositional calculus, or equivalently, a two-valued Boolean algebra now commonly called *switching algebra,* to the systematic representation, analysis, and synthesis of bistable electrical switching circuits. Since then, switching algebra has formed a cornerstone for the study of digital logic systems.

Switching algebra is defined on a set of two elements B = {0, 1}, two binary operators AND and OR (Λ and V), and a unary operator NOT ($\bar{}$). A formal definition for the algebra and its operations is included in almost every textbook on digital logic design and is not repeated here (see for example [Man84]).

A *binary variable* is one which can assume one of the two values in {0, 1}. An n-variable *Boolean function (or switching function)* $f(x_n \cdots x_1)$ on the n binary

variables in $\{x_n \cdots x_1\}$ is a one-to-one mapping from $\{0, 1\}^n$ into $\{0, 1\}$. Thus there are 2^{2^n} Boolean functions of n variables.

A Boolean function may be represented by a truth table, which tabulates its values for all possible assignments of its variables. Let $f(u_n \cdots u_1)$, where $u_i \in \{0, 1\}$, be the value of $f(x_n \cdots x_1)$ when $x_i = u_i$, $\forall\ 1 \le i \le n$. The *truth vector* $\mathbf{T}(f)$ of f is given by $\mathbf{T}(f) = [f(0 \cdots 0), f(0 \cdots 0, 1) \cdots f(1 \cdots 1)]^t$. Note that the superscript "t" denotes vector transpose and $\mathbf{T}(f)$ is a $2^n \times 1$ column vector. The *weight* or *syndrome* of f, denoted by $w(f)$, is the number of 1's in $\mathbf{T}(f)$. The *parity* of f, denoted by $p(f)$, is equal to 0 or 1 depending on whether $w(f)$ is even or odd.

A Boolean function can also be represented by a *Boolean expression* in its variables using operators from the switching algebra. A Boolean expression can be transformed into a number of possible digital logic networks composed of AND, OR and NOT *logic gates*, which can in turn be fabricated using the various integrated circuit (IC) technologies available. Many Boolean expressions exist for a given Boolean function, and much attention has been focused on the simplification of these expressions in order to minimize the implementation cost.

A binary variable x_i in a Boolean expression may appear in its *normal (uncomplemented)* or *complemented* form (i.e. x_i or $\bar{x}_i$). A *product term* is the AND product of a subset of the variables from $\{x_n \cdots x_1\}$, each of which can appear as normal or complemented but not both. For example, $x_1 \Lambda \bar{x}_2 \Lambda x_4$ is a product term. The AND operators are usually omitted and we write $x_1\bar{x}_2x_4$ instead of $x_1 \Lambda \bar{x}_2 \Lambda x_4$. It is conventional to express a Boolean function as an OR sum of AND product terms. For example, the 4-variable *majority-vote* function can be written as $f(x_4, x_3, x_2, x_1) = x_1x_2x_3 \vee x_1x_2x_4 \vee x_1x_3x_4 \vee x_2x_3x_4$. It may be verified that $f = 1$ if and only if three or more of its variables assume the value of 1.

We develop a notation for product terms using *binary* and *ternary n-tuples*, where n is the number of variables in the considered (Boolean) function.

Let $N = 2^n - 1$, $M = 3^n - 1$, $1 \le i \le n$, $0 \le u \le N$, and $0 \le \alpha \le M$. It is convenient to interpret u as the decimal equivalent of the binary n-tuple $(u_n \cdots u_1)$ where

$u_i \in \{0, 1\}$, and we write $u \sim (u_n \cdots u_1)$ or $(u_n \cdots u_1) \sim u$ whenever $u = \sum_{i=1}^{n} u_i \cdot 2^{i-1}$. (We say u *corresponds* to $(u_n \cdots u_1)$ or vice versa). Similarly, α may be decomposed into the ternary n-tuple $(\alpha_n \cdots \alpha_1)$ where $\alpha_i \in \{0, 1, 2\}$ and and we write $\alpha \sim (\alpha_n \cdots \alpha_1)$ or $(\alpha_n \cdots \alpha_1) \sim \alpha$ whenever $\alpha = \sum_{i=1}^{n} \alpha_i \cdot 3^{i-1}$. A ternary n-tuple $(\alpha_n \cdots \alpha_1) \sim \alpha$ is said to be *reducible* to a binary n-tuple $(u_n \cdots u_1) \sim u$, written $(\alpha_n \cdots \alpha_1) \Rightarrow (u_n \cdots u_1)$ or $\alpha \Rightarrow u$, if $\alpha_i \in \{0, 1\}$ and $\alpha_i = u_i \; \forall i$, $1 \le i \le n$.

Let $\alpha \sim (\alpha_n \cdots \alpha_1)$; $x_i^0 = x_i$, $x_i^1 = \overline{x}_i$, and $x_i^2 = 1$; $X = \{x_n \cdots x_1\}$; and $X^\alpha = \bigwedge_{i=n}^{1} x_i^{\alpha_i} = x_n^{\alpha_n} \cdots x_1^{\alpha_1}$. Then X^α is a product term for n-variable functions. Clearly, there are 3^n such product terms, including the constant term $X^M = 1$ where $M = 3^n - 1$.

When $\alpha \Rightarrow u$ (i.e. $\alpha_i \in \{0, 1\} \; \forall i$), the product term X^α involves all the n variables and is called a *minterm*. Since $0 \le u \le N$, there are 2^n minterms of n variables. For convenience, we denote a minterm by X^u instead of X^α when $\alpha \Rightarrow u$. Ambiguity is avoided by noting that u and α correspond to binary and ternary n-tuples respectively.

Let $\overline{u} = N - u$ be the 1's complement of u so that $\overline{u} \sim (\overline{u}_n \cdots \overline{u}_1)$, and recall that $T(f) = [f_0 \cdots f_N]^t$, where $f_u = f(u_n \cdots u_1)$, is the truth vector of $f(x_n \cdots x_1)$. Using *Shannon's decomposition*[Sha38]:

$$f(x_n \cdots x_1) = \overline{x}_i \, f(x_{n-1} \cdots x_{i+1}, 0, x_{i-1} \cdots x_1) \vee x_i \, f(x_{n-1} \cdots x_{i+1}, 1, x_{i-1} \cdots x_1) \qquad (1)$$

on all the n variables $\{x_n \cdots x_1\}$, we can derive the *sum of minterm expansion*

$$f(x_n \cdots x_1) = \bigvee_{u=0}^{N} f_u \, X^{\overline{u}} \qquad (2)$$

Since only the terms involving the 1 values in $\mathbf{T}(f)$ remains in (2), the above sum of minterm expansion uniquely identifies an n-variable function from among the 2^{2^n} sum of minterm expansions of all possible n-variable functions. An expansion that is unique for a Boolean function is called a *canonical expansion.* We describe a class of *modulo-2 canonical expansions* involving the *exclusive-or* (XOR) operator later in this chapter.

A third representation of an n-variable Boolean function $f(x_n \cdots x_1)$ is geometric using an *n-dimensional Boolean hypercube* of 2^n vertices corresponding to the 2^n minterms of n variables. A vertex representing a minterm $X^{\bar{u}}$ is marked white or black depending on whether its coefficient f_u in the minterm expansion (2) is 0 or 1. Since the f_u's are the truth values of f, we call this the *truth hypercube*, or *T-cube* representation of f. For example, Figure 1 is a T-cube representation for $f(x_3, x_2, x_1) = \bar{x}_3\bar{x}_2x_1 \vee \bar{x}_3x_2\bar{x}_1 \vee x_3x_2x_1$ with $\mathbf{T}(f) = [0\ 1\ 1\ 0\ 0\ 0\ 0\ 1]^t$. Note that we choose to label vertices by function values $f_{u_n \cdots u_1} = f_u$ instead of minterms $X^{\bar{u}}$, as is customary in most descriptions for the T–cube found in the literature (see for example [Hurst78]).

Two new representations of a Boolean function are introduced in Chapters 3 and 4. As described in these chapters, the *parity spectrum* in Chapter 3 and the *parity hypercube* in Chapter 4 may be interpreted as extensions from the truth vector and the truth hypercube representations, respectively.

2.3. The Exclusive-or Operator

Modern Boolean algebra employs the *inclusive-or* (i.e. OR) operator. Thus it is perhaps rather amusing that Boole originally used[Boy68] the equivalent of an *exclusive-or* (XOR) operator in his algebra. The use of AND and OR in Shannon's switching algebra is natural as these operators correspond respectively to the operations of serial and parallel connections of switches. In 1954, Reed[Ree54] and Muller[Mul54] observed that any Boolean function can be expressed as an XOR sum of AND products and paved the way for the application of the XOR operator to

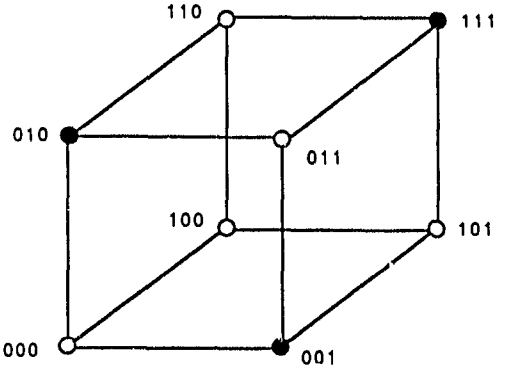


***Figure* 1:** T-cube Representation

digital logic design. Although the implementation of an XOR gate is more complex and costly than an OR gate in most known integrated circuit technologies, many practical circuits, especially those for arithmetic operations, often have more economical XOR implementations. For example, it is well-known that the XOR sum of a group of bits is equal to its *parity*, which may also be interpreted as the sum bit in the binary addition of these bits. Thus XOR is also known as *modulo-2 addition*, and XOR sum of product expressions are often called *modulo-2 (sum-of-product) expansions.*

The XOR operator is usually denoted by the symbol $\oplus$. Table 1 contains a list of properties concerning the XOR operator. Most of them are well known (see for example [Sel68b]) and we only prove X9-X13 in Appendix I.

In Table 1, A, B, and C are Boolean expressions; $f = f(x_n \cdots x_1)$ is a function

Table 1. Properties of the XOR Operator

X1. $0 \oplus 0 = 1 \oplus 1 = 0;\ 0 \oplus 1 = 1 \oplus 0 = 1$

X2. $b_1 \oplus \cdots \oplus b_k = 1$ if there is an odd number of 1 bits in the b_i's, 0 otherwise

X3. $0 \oplus A = A;\ 1 \oplus A = \overline{A};\ A \oplus A = 0;\ A \oplus \overline{A} = 1$

X4. $A \oplus B = \overline{A}B \vee A\overline{B};\ \overline{A \oplus B} = AB \vee \overline{A}\,\overline{B}$

X5. $A \oplus \overline{B} = \overline{A} \oplus B = \overline{A \oplus B};\ A \oplus B = \overline{A} \oplus \overline{B}$ (complementary)

X6. $A \oplus B = B \oplus A$ (commutative)

X7. $A \oplus B \oplus C = (A \oplus B) \oplus C = A \oplus (B \oplus C)$ (associative)

X8. $A(B_1 \oplus \cdots \oplus B_k) = AB_1 \oplus \cdots \oplus AB_k$ (distributive)

X9. $A \vee (B_1 \oplus \cdots \oplus B_{2k-1}) = (A \vee B_1) \oplus \cdots \oplus (A \vee B_{2k-1});$
$\overline{A}(B_1 \oplus \cdots \oplus B_{2k}) = (A \vee B_1) \oplus \cdots \oplus (A \vee B_{2k})$

X10. $A \oplus AB = A\overline{B};\ \overline{A} \oplus AB = 1 \oplus A\overline{B}$

X11. $AB = A \oplus B \oplus (A \vee B);\ A \vee B = A \oplus B \oplus AB$

X12. $A_1 \vee \cdots \vee A_k = A_1 \oplus \cdots \oplus A_k$ if $A_iA_j = 0$ whenever $i \neq j$

X13. $f = \overline{x}_n f^0 \oplus x_n f^1$ (Shannon's decomposition)
$= f^0 \oplus x_n(f^0 \oplus f^1)$ (positive decomposition)
$= f^1 \oplus \overline{x}_n(f^0 \oplus f^1)$ (negative decomposition)
where $f^0 = f(0, x_{n-1} \cdots x_1)$ and $f^1 = f(1, x_{n-1} \cdots x_1)$

of n-variables; $f^0 = f(0, x_{n-1} \cdots x_1)$ and $f^1 = f(1, x_{n-1} \cdots x_1)$. Property X1 defines the XOR operation. X2 computes the parities of a group of bits. X3 concerns a single Boolean expression A. Incidentally, the expression $\overline{A} = 1 \oplus A$ and its derivative

$A = 1 \oplus \overline{A}$ are used by many authors (e.g. [Zha84]) for algebraic conversions among the modulo-2 canonical expansions defined in the next section. X4 relates the inclusive- and exclusive-or operations, and is equivalent to the relationship between inclusive and exclusive unions in set theory. X5 shows the interesting behaviour of XOR under complementation. X6 and X7 shows the commutativity and associativity of the XOR operator. X8 shows that AND is distributive over XOR. X9[1] shows that OR is distributive over XOR only if the number of terms involved is odd, but a nice relationship also exists when the number is even. X10 contains two term-reduction rules used in a number of (mostly heuristic) minimization algorithms for general modulo-2 expressions(e.g. [Eve67, Jad70, Mar74, Rob82]). These rules can be extended to cover more terms. For example, $A \oplus AB \oplus AC \oplus ABC = A\overline{B}\,\overline{C}$, etc. X11 relates AND and OR using XOR. X12 demonstrates the equivalence of the XOR and OR in cases when the operands are *mutually disjoint*. Since minterms are mutually disjoint, an immediate corollary is that the *inclusive-or sum of minterm expansion* of a Boolean function given in (2) can be re-written as the *exclusive-or* (or *modulo-2*) *sum of minterm expansion*, i.e.

$$f(x_n \cdots x_1) = \bigoplus_{u=0}^{N} f_u X^{\overline{u}} \tag{3}$$

Finally, X13 represents the three different ways of decomposing a function into an XOR sum of simpler functions. The expansions can be carried out for any x_i but, for simplicity, are listed only for x_n. The first of these three equations is the XOR form of Shannon's decomposition in (1). The second yields an expansion in x_n but not $\overline{x}_n$, while the last yields the opposite. We have seen in (2) and (3) that the min-term canonical expansion of a Boolean function can be obtained by repeated use of Shannon's decomposition. In the next section, we define a class of 3^n modulo-2 canonical expansions derived by using a combination of the three decomposition

[1] These two properties of $\oplus$ do not appear to have been previously published, although their derivation is straightforward.

rules in X13.

2.4. Modulo-2 Canonical Expansions

2.4.1. FBEs and FPEs

Rearranging the order of the equations in property X13 of Table 1 and extending the consideration to any $x_i \in \{x_n \cdots x_1\}$, we have:

$$f(x_n \cdots x_1) = {}^{x_i}f^0 \oplus x_i\,({}^{x_i}f^0 \oplus {}^{x_i}f^1) \tag{4.1}$$

$$= {}^{x_i}f^1 \oplus \overline{x}_i\,({}^{x_i}f^0 \oplus {}^{x_i}f^1) \tag{4.2}$$

$$= \overline{x}_i\,{}^{x_i}f^0 \oplus x_i\,{}^{x_i}f^1 \tag{4.3}$$

where ${}^{x_i}f^b = f(x_n \cdots x_{i+1}, b, x_{i-1} \cdots x_1)$ for $b \in \{0, 1\}$. Let $B_0 = [1\ \ x]$, $B_1 = [1\ \ \overline{x}]$, and $B_2 = [\overline{x}\ \ x]$ be row vectors where x is one of $x_n \cdots x_1$. We say that equations (4.1), (4.2) and (4.3) are expansions of $f(x_n \cdots x_1)$ in x_i using *bases* B_0, B_1, and B_2 respectively. If we fix a basis B_{α_i} (where $\alpha_i \in \{0, 1, 2\}$) for every variable x_i, $1 \le i \le n$, and then expand f in all its n variables, we obtain a modulo-2 sum of products form called the *fixed basis modulo-2 canonical expansion (FBE)* with composite *basis* $\alpha \sim (\alpha_n \cdots \alpha_1)$, where $\alpha_i \in \{0, 1, 2\}$. Alternatively, we call it the α–FBE of f. Since $0 \le \alpha \le M = 3^n-1$, f has 3^n FBEs.

Example 1: Consider $f(x_3,x_2,x_1) = x_3\overline{x}_2 \vee \overline{x}_2x_1$. Using the basis $19 \sim (2,0,1)$, we apply (4.1)–(4.3) as follows:

$$f(x_3,x_2,x_1) = \overline{x}_3(\overline{x}_2x_1) \oplus x_3(\overline{x}_2)$$

$$= \overline{x}_3(x_1 \oplus x_2x_1) \oplus x_3(1 \oplus x_2)$$

$$= \overline{x}_3x_1 \oplus \overline{x}_3x_2x_1 \oplus x_3 \oplus x_3x_2$$

$$= \overline{x}_3 \oplus \overline{x}_3\overline{x}_1 \oplus \overline{x}_3x_2 \oplus \overline{x}_3x_2\overline{x}_1 \oplus x_3 \oplus x_3x_2 \tag{5}$$

to obtain the final expression (5), which is the 19-FBE of f.

Notice that for *every single* product term in the FBE expansion (5), a variable with basis B_0 either does not appear or appears as uncomplemented; a variable with basis B_1 either does not appear or appears as complemented; and a variable with basis B_2 *always* appears and can either be complemented or uncomplemented.

Using the ternary n-tuple representation for product terms, we develop a new, compact notation for FBEs as follows. Introduce the *circle-star (c-star)* operator, denoted by $\circledast: \{0, 1\}^n \times \{0, 1, 2\}^n \rightarrow \{0, 1, 2\}^n$, with the following componentwise table of operation:

$\circledast$	0	1	2
0	0	1	0
1	2	2	1

(6)

As before, let $x^0 = x$, $x^1 = \overline{x}$, and $x^2 = 1$; $X = \{x_n \cdots x_1\}$; and $X^\alpha = x_n^{\alpha_n} \cdots x_1^{\alpha_1}$. Then the α–FBE of f can be expressed as

$$f(x_n \cdots x_1) = \bigoplus_{u=0}^{N} c_u^\alpha \, x_n^{\bar{u}_n \circledast \alpha_n} \cdots x_1^{\bar{u}_1 \circledast \alpha_1} \tag{7.1}$$

$$= \bigoplus_{u=0}^{N} c_u^\alpha \, X^{\bar{u} \circledast \alpha} \tag{7.2}$$

where $N = 2^n - 1$, $c_u^\alpha \in \{0, 1\}$, $\bar{u} = N - u$, and $\bar{u} \sim (\bar{u}_n \cdots \bar{u}_1)$. Let $C^\alpha(f) = [c_0^\alpha \cdots c_N^\alpha]^t$ be the column vector of 2^n *coefficients* in the α–FBE of f. For a fixed α, it can be shown that the 2^{2^n} possible assignments of $C^\alpha(f)$ corresponds to the α–FBEs of all n-variable functions, so that $C^\alpha(f)$ uniquely represents f. Hence an α–FBE is a modulo-2 canonical expansion.

When $\alpha \Rightarrow v \sim (v_n \cdots v_1) \in \{0, 1\}^n$, only equations (4.1) and (4.2) are used to obtain the expansion and the α–FBE contains each variable i as uncomplemented (x_i) or complemented ($\overline{x}_i$) but not both. We say that x_i has a fixed *polarity* and call the α–FBE the *fixed polarity modulo-2 canonical expansion (FPE)* with composite *polarity* v, or simply, the v–FPE of f. For convenience, we allow the c-star operator to

act on two binary n-tuples so that $\overline{u} \oplus v = (\overline{u}_n \oplus v_n \cdots \overline{u}_1 \oplus v_1)$ and use an alternative to (7.2) for expressing the v–FPE as:

$$f(x_n \cdots x_1) = \bigoplus_{u=0}^{N} a_u^v \, X^{\overline{u} \oplus v} \tag{8}$$

and let $A^v(f) = [a_0^v \cdots a_N^v]^t$ be the column vector of coefficients from this v–FPE.

The set of 2^n FPEs has attracted much attention from a number of authors. Reed[Ree54] and Muller[Mul54] first show that any Boolean function can be expressed in its 0–FPE:

$$f(x_n \cdots x_1) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_1x_2 \oplus a_4x_3 \oplus \cdots \oplus a_Nx_1 \cdots x_n \tag{9}$$

where $a_u \in \{0, 1\}$, $0 \le u \le N = 2^n-1$. Equation (9) is more often called the *complement-free (or positive) Reed-Muller canonical expansion.* By symmetry consideration of x_i and $\overline{x}_i$, Akers[Ake59] shows that f can be expressed in, for $0 \le v \le N$, its v–FPE:

$$f(x_n \cdots x_1) = a_0^v \oplus a_1^v x_1^{v_1} \oplus a_2^v x_2^{v_2} \oplus a_3^v x_1^{v_1} x_2^{v_2} \oplus \cdots \oplus a_N^v x_1^{v_1} \cdots x_n^{v_n} \tag{10}$$

which is equivalent to (and somewhat more readable than) equation (8). Because of their popularity in the literature, the FPEs have been given a host of exotic names such as *Taylor expansions*[Ake59, Dav71], *consistent canonical forms*[Coh60, Coh62], *polarized polynomial forms*[Mar74], *generalized Reed-Muller expansions*[WuC82], and *Reed-Muller polynomials with fixed polarity*[Zha84].

In contrast, the more general set of 3^n FBEs are not well known in the literature. Bioul, Davio, and Deschamps[Bio73] are the first to derive these expansions using equations (4.1)–(4.3) in vector notation. These expansions are also emphasized in a book[Dav78] published later by Davio, Deschamps and Thayse, but do not appear to have been discussed elsewhere. Part of this research is devoted to the extension of known results on matrix transforms and minimization procedures for FPEs to FBEs. These extended results are reported in Chapter 5.

One *mixed polarity* FBE of special interest is the M-FBE where $M = 3^n-1 \sim (2 \cdots 2)$. Since only Shannon's expansion (4.3) is used throughout, the M-FBE is the modulo-2 sum of minterm expansion as given in (3). Putting $\alpha = M$ in (7.2) and noting that $\bar{u} \circledast M = \bar{u}$, it can be seen that (7.2) reduces to (3). Consequently, $c_u^M = f_u$ and $\mathbf{C}^M(f) = \mathbf{T}(f)$.

Apart from being of immense theoretical interest, the study of FPEs and FBEs is also of practical importance.

Firstly, the economical design of digital logic networks composed of XOR gates is dependent on effective simplification procedures for general modulo-2 expansions. Unfortunately, no effective method has yet been developed and only heuristic ones which do not guarantee optimality have been proposed. By studying less general but structurally simpler modulo-2 canonical expansions such as the FPEs and FBEs, valuable insights can be gained into the structure of general modulo-2 expansions and their minimization procedures. For example, the FBEs have been used in [Bio73] to obtain minimal modulo-2 expansions for any 4-variable function. As another illustration, it is noted that some heuristic modulo-2 minimization procedures(e.g.[Rob82]) start from an FPE with the smallest number of terms among all the FPEs, and then search for mixed polarity reductions.

Secondly, digital logic networks which directly implement FPEs have been shown to have *easily testable* properties[Red72, Sal75]. Because of the escalating cost of testing VLSI chips, an FPE network may be preferable to a cheaper modulo-2 or even inclusive-or sum of product implementation if the gain in testability is sufficient to offset the increase in implementation cost.

Lastly, as demonstrated in [Ree54, Mul54], the study of FPEs and FBEs may have applications in error detecting and correcting codes.

In Chapter 4, the parity spectrum is used to develop new algebraic and geometric representations for FPEs and FBEs.

2.4.2. Modulo-2 Minimization

The *weight* of an FPE or FBE is its number of non-zero terms. Let $\omega^{\alpha}(f)$ denote the weight of the α–FBE of $f(x_n \cdots x_1)$, then

$$\omega^{\alpha}(f) = \sum_{u=0}^{N} c_u^{\alpha} \tag{11.1}$$

where $N = 2^n-1$ and $C^{\alpha} = [c_0^{\alpha} \cdots c_N^{\alpha}]^t$ is the vector of coefficients from the α–FBE of f. For brevity, we write ω^{α} instead of $\omega_{\alpha}(f)$ if it is clear which function f is being considered. The vector of all 3^n FBE weights is denoted by

$$\Omega(f) = [\omega^0 \cdots \omega^M]^t \tag{11.2}$$

where $M = 3^n-1$. Similarly, the weight of the v–FPE of $f(x_n \cdots x_1)$ is denoted by

$$w^v(f) = \sum_{u=0}^{N} a_u^v(f) \tag{12.1}$$

where $A^v = [a_0^v \cdots a_N^v]^t$ is the vector of coefficients from the v–FPE. The vector of all 2^n FPE weights is

$$W(f) = [w^0 \cdots w^N]^t \tag{12.2}$$

where $w^v = w^v(f)$. Clearly, $\omega^{\alpha} = w^v$ when $\alpha \Rightarrow v$, and $W(f)$ is a subset of $\Omega(f)$.

A minimization algorithm for FBEs selects from among the 3^n FBEs one that has the smallest number of non-zero terms, i.e. an FBE with the minimal weight. Since several FBEs can have the same weight, a minimal solution may not be unique. Analogously, an algorithm for FPEs selects from among the 2^n FPEs one that has the minimal weight. The assumption is that by minimizing the number of terms that have to be implemented, the implementation cost is reduced. Also, a minimal FPE or FBE is a good starting point for applying heuristic minimization procedures to derive a near-minimal modulo-2 expression(e.g.[Rob82]).

Many authors consider the minimization of FPEs(e.g.[Dav71, Swa72, Sal79, WuC82, Bes83, Zha84, Gre87]) using vastly different sets of notations and terminologies. Consequently, much confusion is generated and duplicated reports of the

same results have appeared. In Chapter 5, these results are presented in a unified manner, and extended to cover FBEs. Using Boolean matrix transforms, two approaches to the minimization of FPEs and FBEs are described, analyzed and compared.

2.5. Boolean Matrix Transforms

Many authors[Lec63, Dav71, Swa72, Dav78, Sal79, WuC82, Bes83, Zha84, Gre87] consider fast transforms among FPEs, and to a smaller extend, among FBEs of a Boolean function. These transforms are best described by Boolean matrices. Calingaert[Cal61] first introduces the use of square non-singular Boolean matrix transforms, and Lechner[Lec63] proposed to describe them as *Kronecker products* of elementary matrices. The Kronecker product approach allows efficient recursive algorithms to be devised for these transforms.

Let $A = \left[a_{ij}\right]_{p\times q}$ and $B = \left[b_{ij}\right]_{r\times s}$ be Boolean matrices. The Kronecker product $A \otimes B$ is a Boolean matrix of dimension $pr \times qs$ given by

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1q}B \\ a_{21}B & a_{22}B & \cdots & a_{2q}B \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ a_{p2}B & a_{p2}B & \cdots & a_{pq}B \end{bmatrix} \tag{13}$$

Let C and D also be Boolean matrices. The properties in Lemma 1 below are well known for real matrices under real multiplication (see for example [Gra81], Section 2.3). They can easily be proved for Boolean matrices under modulo-2 matrix multiplication.

Lemma 1:

(a) $A \otimes (B \otimes C) = (A \otimes B) \otimes C$

(b) $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$, if the dimensions of the matrices are such that AC and BD exist

(c) $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$, if the inverses A^{-1} and B^{-1} exist □

In Chapter 5, existing matrix transforms among the FPEs and FBEs are generalized and new transforms between the parity spectrum and the FPEs and FBEs are developed. These transforms, which are described using Kronecker products, are applied to the minimization of FPEs and FBEs. Two minimization approaches are described. The first approach explicitly generates all FPEs (or FBEs) in a *Gray code* sequence of their polarities (bases). In the second approach, weights of the FPEs or FBEs are computed from the parity spectrum using matrix transforms, which are also described using Kronecker products.

2.6. Gray Codes

The *distance* between two binary n-tuples $(u_n \cdots u_1)$ and $(v_n \cdots v_1)$, denoted by d(u,v), is the number of positions where $u_i \neq v_i$. The distance $d(\alpha,\beta)$ for two ternary n-tuples is similarly defined. If d(u,v) = 1, u and v are said to be adjacent. Similarly, α and β are adjacent if $d(\alpha,\beta) = 1$.

If we list all 2^n possible binary n-tuples in such a way that each n-tuple is adjacent to its immediate predecessor, we form a sequence known as a *binary Gray code.* Similarly, a *ternary Gray code* is a list of 3^n ternary n-tuples such that each n-tuple is adjacent to the preceding one.

To simplify the notation we represent an n-tuple in a (binary or ternary) code sequence by a string of n *bits (binary digits)* or *trits (ternary digits).* The n-bit *binary reflected Gray code* B_n is defined recursively as(see [Rei77]):

$$B_1 = \begin{matrix} 0 \\ 1 \end{matrix}; \qquad B_n = \begin{matrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 1 \end{matrix} \begin{matrix} \boxed{B_{n-1}} \\ \\ \boxed{B_{n-1}^R} \end{matrix} \tag{14.1}$$

where B_n^R is the list B_n reversed. Thus B_1 = 0, 1; B_2 = 00, 01, 11, 10; and B_3 = 000, 001, 011, 010, 110, 111, 101, 100. This can be extended to the *ternary reflected Gray code*[Flo56] T_n, which is defined recursively as:

$$T_1 = \begin{matrix} 0 \\ 1 \\ 2 \end{matrix}; \qquad T_n = \begin{matrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 1 \\ 2 \\ \vdots \\ 2 \end{matrix} \begin{matrix} \boxed{T_{n-1}} \\ \\ \boxed{T_{n-1}^R} \\ \\ \boxed{T_{n-1}} \end{matrix} \tag{14.2}$$

where T_n^R is the reverse of the list T_n. For example, T_1 = 0, 1, 2; and T_2 = 00, 01, 02, 12, 11, 10, 20, 21, 22. Algorithm 1 (in pseudo-PASCAL) extends Algorithm 5.7(a) from [Rei77] to generate either the binary or ternary reflected Gray code. It runs in O(n) time, and is used in the Gray code approach to the minimization of FPEs and FBEs in Chapter 5.

Algorithm 1: Generating Binary or Ternary Reflected Gray Code

```
const
  k  = 2 or 3;                          (* binary or ternary *)
  k1 = 1 or 2;
var
  g: array[1 · · · n] of 0 · · · k1;    (* the code *)
  d: array[1 · · · n] of (−1,+1);       (* directions *)
  S: stack;

procedure Init;
begin
      for i := n downto 1 do
            g[i] := 0; d[i] := +1; Push(S, i);
end Init;

procedure Next;
begin
      if IsEmpty(S) then stop;
      i := StackTop(S);
      g[i] := g[i] + d[i];
      if (g[i] = 0) or (g[i] = k1) then
            d[i] := − d[i]; Pop(S);
      for j:=i−1 downto 1 do
            Push(S, j);
end Next;
```

2.7. Fault Detection

Digital logic systems can be faulty for a number of reasons such as design errors, manufacturing defects, abusive misuse, or wear and tear. The ability to detect and locate faults can substantially affect the total cost of design, production, testing

and field maintenance of a digital system. Thus the problem of digital logic network testing has received much attention in recent years.

A digital logic system is an assembly of digital logic networks which accept input binary logic signals, process these signals, and produce binary signals as outputs. Digital logic networks may be studied at the circuit level (switch level) or logic level (gate level). Of interest in the circuit level are electrical components such as transistors, diodes and resistors; and physical quantities such as voltages, currents and wave shapes. At the logic level, a digital logic network is considered to be an interconnection of memory elements called flip-flops and processing elements called logic gates. A digital logic network without memory elements and feedback connection loops is classified as *combinational* (or *memoryless*), otherwise it is said to be *sequential*. The characteristic of a combinational network is that its steady state outputs depends only on its steady state inputs. In other words, it directly implements Boolean functions with its inputs representing binary variables and outputs corresponding to function values.

This research is only concerned with the testing of combinational logic networks. Although sequential networks are not considered, they can be tested in the same manner as combinational networks provided that they are designed using scan-path techniques such as Level Sensitive Scan Design (LSSD) (see [Wil84]) which separates the implementation of the combinational control logic from that of the memory elements. Design for testability techniques such as LSSD greatly simplifies the testing problem, and represent a growing trend in the digital electronic industry to incorporate testability considerations in the design in order to manufacture products that are more easily tested. The recent proposed Joint Test Action Group (JTAG) standard on boundary scan testing[van90] is an extension of this approach to the printed-circuit-board (PCB) level.

To guarantee reliable products, two kinds of testing must be performed. Parametric testing (or AC testing) is concerned with the time-related behaviour of the logic network and involves the measurement of actual voltage and current levels. In

logic testing (or DC testing), the objective is to detect any *logic fault* which causes the combinational *network under test (NUT)* to cease to implement the intended Boolean function(s). This research deals only with logic testing. For simplicity, only single-output combinational networks are considered, although many of the results can be extended to logic networks with multiple outputs.

Logic testing consists of applying a sequence of input stimuli to the NUT and comparing its responses to the expected sequence of values. Any discrepancy indicates the presence of a fault. In the extreme, an exhaustive test where all possible input stimuli are applied to the NUT and all output responses are verified individually is the only way to detect arbitrary logic faults. Since an n-input single-output network requires the application of 2^n input patterns and the storage for 2^n correct responses, the test time and data volume for exhaustive testing increases exponentially with the number of inputs n. Assuming that input patterns can be applied to the NUT at a rate of 1 μs per input pattern, an LSI chip having 25 input pins and one output pin requires 32 seconds of test time and 32M (mega) bits of storage. For a chip with 40 inputs, the test time increases to over 12 days and the test volume to 1024G (giga) bits! Therefore, exhaustive testing is feasible only for chips with a small number of inputs.

Another approach to logic testing is to exercise the functionality of the NUT by applying stimuli that are considered to be its typical inputs. If all the responses produced are correct, it is tempting to conclude that the NUT is free of defects. Unfortunately, a fault may not be detected by the selected stimuli and this approach easily leads to faulty products being shipped to the customers.

In 1959, Eldred[Eld59] advocated the technique of testing for hardware defects rather than functional incorrectness. The most commonly occurring faults caused by known real physical defects are modelled by a *fault model.* Input stimuli can then be chosen according to this model such that any fault within the model would cause the NUT to produce an incorrect response to one of the selected stimuli. Because of its effectiveness in practice, Elred's approach has since become a standard in digital

logic testing.

The most popular logic fault model used throughout the electronic industry is the *single stuck-at fault* model[Eld59]. The model assumes that in a logic gate level representation of the NUT, one of the gate inputs or outputs is fixed at either logic zero or logic one. With the increasing density in LSI and VLSI chips, it becomes more common to have a single physical defect that manifest itself into several logic faults at the same time. This leads to the more general *multiple stuck-at fault* model[Bos71, She72], which assumes that several lines in a gate level representation of the NUT are simultaneously stuck. As shown by the statistical analysis in [Gol77], multiple stuck-at faults are not uncommon in LSI chips and the single fault assumption is inadequate.

The main advantage of the stuck-at fault models, which have since survived many transitions in IC technology, is their effectiveness to model actual physical faults that commonly occur. However, it is known that certain common defects, in particular shorts[Mei74] and CMOS opens[2][Wad78], are not modelled by stuck-at faults. Many test generation programs still generate tests only for the stuck-at faults in view of the fact that such tests are practically sufficient to identify most of the faulty chips. Also, some encouraging experimental results reported in [Wil84] show that 99.5% of the randomly selected shorts in sample circuits are detected by test patterns designed for stuck-at faults. On the other hand, CMOS open faults can cause a faulty combinational network to exhibit sequential behaviour[Wad78] and this is probably the most potentially damaging evidence against the stuck-at fault model[3][Wil84].

Using a presumed logic fault model, the traditional *test vector testing* method applies a selected subset (called the *test set*) of all input patterns to the NUT and

---

[2] Some authors argue that CMOS open faults occur very rarely in practice.

[3] Strictly speaking, a CMOS open fault is not a logic fault. However, a logic fault test can sometimes detect such a fault if the test inputs are applied in a specific order[Red83].

verifies the observed responses bit by bit. Problems with this approach include costly test generation and verification,[4] excessive storage requirement for test inputs and expected outputs, expensive automatic test equipment, and its inability to adapt to *build-in self test (BIST)*[5]. These shortcomings are addressed by newer techniques that incorporate the concept of *compaction testing*[McC85], which passes the responses from the NUT to a compacter circuit that reduces them to a bit vector called the *signature*. The observed signature is then compared to its expected value, and any discrepancy indicates a fault. Most noticeable compaction techniques include signature analysis[Fro77], transition counting[Hay76], syndrome and spectral testing[Tzi78, Sav80, Sus83, Muz83, Lui83, Lui86], and several parity-based testing techniques[Tzi78, Car82, Ake88, Dam89].

The parity-based techniques are of special interest in this research, which develops a generalized approach to encompass all these techniques. Tzidon et al.[Tzi78] and Carter[Car82] first consider *parity testing* by employing a one-bit signature, which is the parity on the number of logic ones of all responses from the NUT. Akers[Ake88] extends the technique to *parity signature testing* using an (n+1)-bit signature that also includes n subparities obtained by constraining one of the n inputs of the NUT at logic zero. More recently, Damarla and Karpovsky[Dam89] introduce *Reed-Muller coefficient testing* by deriving a signature from coefficients in the complement-free Reed-Muller canonical expansion (i.e. the 0-FPE) of the implemented function. By associating coefficients in FPEs and FBEs to the parity spectrum in Chapter 4, this research shows that the use of Reed-Muller coefficients for testing in [Dam89] is equivalent to employing subparities from constraining *any number* of the network inputs at logic zero.

---

[4] Goel[Goe81] observes that the computer run time to perform test generation and fault simulation is approximately proportional to the number of logic gates to the power 3.

[5] BIST is a scheme where some or all of the external tester functions are moved onto the chip being tested. The aim is to reduce testing effort and the need for expensive external testers[McC85].

Chapter 6 presents the generalized scheme, called *constrained parity testing*, where any number of the inputs can be constrained at *either logic zero or logic one*. This scheme is also a generalization of the traditional test vector method, since a constrained test that simultaneously constraints all the network inputs is equivalent to a test vector test.

2.8. Summary

In this chapter, the background, notation, and terminology for this dissertation are presented.

A brief history of Boolean algebra is reviewed. Representations of Boolean functions are described. A notation for product terms using binary and ternary n-tuples is developed. Fundamental properties of the XOR operator are listed and proved in Appendix I. Fixed polarity and fixed basis modulo-2 canonical expansions (FPEs and FBEs) are defined, and a compact notation for these expansions is introduced using the new ⊛ (c-star) operator on binary and ternary n-tuples. The minimization problem for these expansions is also introduced. Known properties of Kronecker product of real matrices under real multiplication are re-stated for modulo-2 matrices under modulo-2 multiplication. A stack-based algorithm for generating binary reflected Gray code is generalized to the ternary reflected Gray code. Finally, background on fault detection techniques for digital logic circuits is provided.

The ternary notation developed for product terms (Section 2.2) and FBEs (Equation (7.2)) can also be used in compact expressions of general inclusive-or or modulo-2 sums of products by assigning a ternary n-tuple to each of the 3^n coefficients in such an expression.[6] Although this ternary notation for Boolean expressions is more elegant than other proposed ones, it does not appear to be widely used in the literature.

---

[6]As in Muller[Mul54] for an implementation of a modulo-2 minimization algorithm on the ILLIAC computer at University of Illinois in 1954.

In the following chapters, we introduce the concept of a parity spectrum and describe its applications to modulo-2 logic design and fault detection.

CHAPTER 3

THE PARITY SPECTRUM

3.1. Introduction

Akers[Ake59] proposes the concept of *Boolean difference* for defining and examining various formal properties of Boolean functions. His work allows an analogy to be drawn between the theories of Boolean and ordinary functions, and is useful for obtaining series expansions of a Boolean function using modulo-2 additions and multiplications (i.e. XOR sum of AND products). Since then, Boolean difference has been applied to modulo-2 logic design[Dav71, Bio72, Bio73, Dav78] and fault detection[Sel68a, Sel68b, KuM75].

There are two major problems with Boolean difference. Firstly, the expressions often involve complex algebraic notation. Secondly, it is difficult to grasp the intuitive meaning of Boolean difference. It appears that these difficulties stem from using an ordinary function theory approach which may not be appropriate for studying practical engineering applications of Boolean functions.

This chapter presents a new way of studying a Boolean function using its *parity spectrum*, which is a vector of its subfunction parities. The relationship between parity spectrum and Boolean difference is described. It is shown that the former may replace the latter in many applications with considerably simplified notation and more intuitive analysis, and that the former may be used to unify and generalize a number of important results in several areas of switching theory. Various fundamental properties of the parity spectrum are established. In particular, properties involving joint functions, subfunctions, dependent variables, variable complementation and permutation, and simple functions are examined.

Section 3.2 defines the parity spectrum. Section 3.3 describes its relationship with Boolean difference. Section 3.4 demonstrates its usefulness in unifying and generalizing existing results in series expansion, logic design, and fault detection. Section 3.5 establishes its fundamental properties which are instrumental for further development of the applications in later chapters. Proofs for these properties appear in Appendix II.

3.2. The Parity Spectrum

Let $f(x_n \cdots x_1)$ be an n-variable Boolean function. A *subfunction* of f of k variables, $0 \le k \le n$, is a Boolean function formed from restricting each of any $n - k$ variables of $x_n \cdots x_1$ in f at 0 or 1. Thus f has $\binom{n}{k} \cdot 2^{n-k}$ subfunctions of k variables for a total of 3^n subfunctions including itself. Associate each subfunction with a ternary n-tuple $(\alpha_n \cdots \alpha_1) \sim \alpha$ such that the subfunction, denoted by f_α, is formed from $f(x_n \cdots x_1)$ by replacing x_i with α_i if and only if $\alpha_i \in \{0, 1\}$.

Example 1: A 4-variable function $f(x_4 \cdots x_1)$ has $3^4 = 81$ subfunctions denoted by $f_0 \cdots f_{80}$. So $f_{65} = f(x_4,1,0,x_1)$ since $65_{10} = 2102_3$. □

Let $f_u = f(u_n \cdots u_1)$ where $u_i \in \{0, 1\}\ \forall\ i,\ n \ge i \ge 1$. The *truth vector* $T(f)$ of $f(x_n \cdots x_1)$ is given by $T(f) = [f_0 \cdots f_N]^t$, where $N = 2^n - 1$ and "t" denotes vector transpose. The *parity* of f, denoted by $p(f)$, is equal to 1 or 0 depending on whether the number of 1's in $T(f)$ is odd or even. Mathematically, the parity of f is

$$p(f) = \bigoplus_{j=0}^{N} f_j \tag{1}$$

with the summation modulo-2.

The *parities of subfunctions*, or *subparities*, of f are similarly defined. Thus if f_α is a subfunction of k variables (k is the number of 2's in α), then $p(f_\alpha)$ is the parity in $T(f_\alpha)$. Note that $T(f_\alpha)$ has 2^k entries since f_α is a k-variable function. Let $p_\alpha(f) \equiv p(f_\alpha)$ and if it is clear which function f is being considered, we simply write p_α instead of $p_\alpha(f)$. Define the *parity spectrum* of f to be the vector of subparities

$\mathbf{P}(f) = [p_0 \cdots p_M]^t$ where $M = 3^n - 1$. Note that $p_M = p(f)$. Note also that if $\alpha \Rightarrow u$, then f_α is equal to f_u in $\mathbf{T}(f)$, so that $p_\alpha = p(f_\alpha) = f_u$. Hence $\mathbf{T}(f)$ is embedded in $\mathbf{P}(f)$.

We use f_u to denote f_α (i.e. $f_\alpha = f_u = f_u$) when $\alpha \Rightarrow u$. Ambiguity is avoided by noting that u corresponds to a binary tuple while α corresponds to a ternary tuple. Also, when convenient we replace decimal number subscripts by their corresponding n-tuples and write $f_{u_n \cdots u_1}$, $p_{\alpha_n \cdots \alpha_1}$ etc. instead of f_u and p_α.

We now give an alternative definition of $p_\alpha(f)$ in terms of $\mathbf{T}(f)$. A binary n-tuple $(u_n \cdots u_1) \sim u$ *subsumes*[Die71] a ternary n-tuple $(\alpha_n \cdots \alpha_1) \sim \alpha$ if $\alpha_i = u_i$ or $\alpha_i = 2$, $\forall i$. We denote the subsumption relationship by $u \sqsubseteq \alpha$. Let the *binary domain* of a ternary n-tuple α be the set of all binary n-tuples that subsume α. Formally, we write $D(\alpha) = \{u : u \sqsubseteq \alpha\}$. It is easy to show that $D(\alpha)$ contains all entries in $\mathbf{T}(f_\alpha)$, and the alternative definition is therefore

$$p_\alpha(f) = p(f_\alpha) = \bigoplus_{u \in D(\alpha)} f_u \tag{2}$$

Example 2: Let $f(x_2, x_1) = \overline{x}_2 \vee x_1$ with $\mathbf{T}(f) = [f_{00} f_{01} f_{10} f_{11}]^t = [1\ 1\ 0\ 1]^t$. Then $p(f_{20}) = p(\overline{x}_2) = 1$. Also, $D(20_3) = \{00_2, 10_2\}$, so $p_{20}(f) = f_{00} \oplus f_{10} = 0 \oplus 1 = 1$. □

3.3. Boolean Difference

Given a Boolean function $f(x_n \cdots x_1)$, the first order Boolean difference with respect to a single variable x_i, denoted by f_{x_i}, is defined as[Ake59]

$$f_{x_i} = {}^{x_i}f^0 \oplus {}^{x_i}f^1 \tag{3}$$

where ${}^{x_i}f^b = f(x_n \cdots x_{i+1}, b, x_{i-1} \cdots x_1)$ for $b \in \{0, 1\}$. The second order difference with respect to x_i and x_j, denoted by $f_{x_i x_j}$, is defined as $(f_{x_i})_{x_j}$. From property (4) in [Ake59], $f_{x_i x_j} = f_{x_j x_i}$. Higher order (> 2) differences can be defined analogously. It may be seen that a Boolean difference of order k is a function of $n - k$ variables.

Let $h(X; Y) = h(x_n \cdots x_1; y_m \cdots y_1)$ be a Boolean function in (n+m) variables $X = \{x_n \cdots x_1\}$ and $Y = \{y_m \cdots y_1\}$, and $h_X(Y)$ be the Boolean difference of $h(X; Y)$ with respect to X. Then by definition,

$$h_X(Y) = h(0 \cdots 00; Y) \oplus h(0 \cdots 01; Y) \oplus \cdots \oplus h(1 \cdots 11; Y). \quad (4)$$

Let the binary values $v_m \cdots v_1$ be an assignment to the variables $y_m \cdots y_1$. Clearly, the Boolean difference value $h_X(v_m \cdots v_1)$ is equivalent to the parity of the subfunction $h(X; v_m \cdots v_1)$. In other words, an alternative (admittedly obscure) definition of a subfunction parity could be given in terms of *a value of a particular Boolean difference of the given function evaluated at a particular assignment of the variables*. Indeed, this definition is implicitly used by Bioul & Davio[Bio72] to obtain the *extended truth vector*, which is identical to the parity spectrum.

Example 3: Let $f(x_3, x_2, x_1)$ be a 3-variable Boolean function with $\mathbf{P}(f) = [p_{000} \cdots p_{222}]^t$. Then $p_{021}(f) = f_{x_2}(0,1)$. Conversely, $\mathbf{T}(f_{x_2}(x_3, x_1)) = [p_{020}\ p_{021}\ p_{120}\ p_{121}]^t$. □

Consequently, a formal relationship exists between the parity spectrum and the Boolean differences of a Boolean function. The expressions for this relationship are, nevertheless, rather tedious and do not seem to have any practical use. Informally, a Boolean difference is an algebraic expression defining a subset of the binary values in the parity spectrum, while the parity spectrum is a vector of values of all possible (including zero order) Boolean differences[1] evaluated at all possible assignments of the variables.

3.4. A Simplified Theory

The concept of Boolean difference has been applied to modulo-2 expansions[Ake59, Dav71, Dav78], modulo-2 minimization[Dav71, Bio73, Dav78],

[1] These are the Boolean differences as defined in [Ake59], and do not include later extensions such as the *double difference* in [Sel68a,b].

and fault detection[Sel68a,b, KuM75]. Using the proposed concept of parity spectrum in these areas has several advantages. Firstly, the parity spectrum simplifies the derivations of and offers new insights into many existing results. Secondly, the parity spectrum is a tool for unifying other previous results that are not derived from Boolean difference. Lastly, the parity spectrum allows many of the existing results to be generalized.

3.4.1. Modulo-2 Expansions

Boolean difference has been employed to obtain modulo-2 expansions of a Boolean function(see [Ake59, Dav71, Dav78]). Consider an example function $h(x_2,x_1;Y)$, From [Ake59], Theorem 2,

$$h(x_2,x_1;Y) = h(0, 0;Y) \oplus x_1 h_{x_1}(0;Y) \oplus x_2 h_{x_2}(0;Y) \oplus x_1 x_2 h_{x_1 x_2}(Y) \tag{5}$$

By putting $Y = \varnothing$ as in [Dav71], the *0-FPE* of $f(x_2,x_1)$ is

$$f(x_2,x_1) = f(0, 0) \oplus x_1 f_{x_1}(0) \oplus x_2 f_{x_2}(0) \oplus x_1 x_2 f_{x_1 x_2} \tag{6}$$

Since the *coefficients* ($f(0, 0)$, $f_{x_1}(0)$, etc.) in (6) are values of Boolean differences evaluated at $x_2 = 0$ and/or $x_1 = 0$, they can be replaced by their equivalent parities of subfunctions to form

$$f(x_2, x_1) = p_{00} \oplus p_{02}x_1 \oplus p_{20}x_2 \oplus p_{22}x_1x_2 \tag{7}$$

Obviously, (7) is notationally simpler than (6). It also seems to be clearer because, intuitively, it is easier to interpret subfunction parities than to interpret values of Boolean differences.

Expressions like (6) have been derived for any of the 2^n *Fixed Polarity Expansions (FPEs)* of an n-variable function (see [Ake59, Dav71]). Consequently, all FPEs can be expressed in terms of subfunction parities. Moreover, the results can be generalized to express *Fixed Basis Expansions (FBEs)* in terms of subfunction parities as well. These results also lead to a new geometric interpretation of FPEs and FBEs using a *parity hypercube.* These results are presented in Chapter 4.

In summary, the parity spectrum leads to many new and interesting results for deriving and studying modulo-2 canonical expansions.

3.4.2. Modulo-2 Minimization

Following the work in [Sha38], design methods for logic networks are based primarily on the logic primitives NOT, AND and OR. It is observed in [Ree54, Mul54] that a Boolean function may be expressed as a canonical form in terms of modulo-2 additions and multiplications (i.e. XOR and AND). Since then, attention has also been given to design using the XOR primitive. As noted in Chapter 2, although an XOR gate is more complex and costly than an OR gate in most known digital logic technologies, many practical networks, especially those for arithmetic operations, often have more economical XOR implementations.

Optimal XOR implementations ultimately rely on effective minimization procedures for general modulo-2 sum of product expansions of Boolean functions. Unfortunately, no effective solution has yet been discovered although many heuristic algorithms for finding near-minimal expansions have been proposed.

An important subproblem concerns algorithms for finding a *Fixed Polarity Expansion (FPE)* of a Boolean function with the smallest number of terms. As pointed out by some authors[Red72, Sal75], networks based on FPEs have many *easily testable* properties. Furthermore, some heuristic algorithms for the general modulo-2 minimization problem starts from a minimal FPE, and then search for mixed polarity reductions (e.g. [Rob82]). Since FBEs generalize FPEs (see Section 2.4.1), minimization algorithms for the latter often extend to cover the former (see Chapter 5).

Now it can be shown that the parity spectrum contains all the coefficients in all the FPEs and FBEs of the considered function (see Chapter 4). This allows the development of fast FPE and FBE minimization algorithms that makes use of the parity spectrum. These algorithms, to be presented in Chapter 5, resemble the ones that appear earlier in [Bio73], which use the *extended truth vector* derived from

Boolean difference. Using the parity spectrum, the derivations and proofs are considerably simplified.

Other concepts developed for the minimization of FPEs also have simpler derivations and descriptions in terms of the parity spectrum. The *polarity functions* in [Muk70] have been shown in [Dav71] to be equivalent to Boolean differences, and thus can be re-interpreted using the parity spectrum. The *polarity-coefficient matrix* in [Fis74] can be more compactly represented by the parity spectrum, and the *k-subnumbers* in the same paper are similar to elements in the binary domain of a ternary n-tuple, as used in (2) in the definition of a subfunction parity from the parity spectrum.

In summary, the parity spectrum is useful in the design of economical XOR logic networks, as is described in Chapter 5.

3.4.3. Fault Detection

A combinational logic network[2] may be tested for faults by applying a sequence of inputs to the network and comparing the outputs with the expected values. Any discrepancy indicates a fault.

Under the traditional *test vector testing* approach, a selected subset of the inputs are applied and the responses are verified bit by bit. Boolean difference has been employed to generate test vectors for detecting *single* and *multiple stuck-at faults* in a network[Sel68a,b, KuM75].

Compaction testing[McC85] is a newer approach that is used to reduce the volume of the test output data by compacting the responses into a bit vector call the *signature*, which can then be compared to its expected bit values. Several techniques based on this approach have been described in the literature. In *parity testing*[Car82], the parity of the function implemented by the network under test is

[2] As mentioned in Chapter 2, the testing of sequential networks can be reduced to the race-free testing of combinational networks using design for testability techniques such as Level Sensitive Scan Design (LSSD)[Wil84].

used as a signature. This technique is later generalized to *parity bit signature testing*[Ake88], where the parities of the function and n of its subfunctions constitute an (n+1)-bit signature. A third technique called *Reed-Muller coefficient testing*[Dam89] forms a signature by selecting coefficients from a single FPE of the function.

Using the parity spectrum, a new compact testing technique can be devised. This technique, called *constrained parity testing*, selects as the signature a subset of the subfunction parities in the parity spectrum. Constrained parity testing is a generalization of the four mentioned techniques: test vector testing, parity testing, parity bit testing, and Reed-Muller coefficient testing. This is because the truth vector is embedded in the parity spectrum, and the parity spectrum is a vector of subfunction parities that can be identified with coefficients in the FPEs.

To conclude, the parity spectrum leads to the new constrained parity testing technique which generalizes the traditional test vector testing technique and several parity-based compact testing techniques. This generalized technique is presented in Chapter 6.

3.5. Properties of the Parity Spectrum

The preceding discussion demonstrates the significance of the parity spectrum and motivates the establishment of its most fundamental properties in this section. These properties are listed in Table 1 and proved in Appendix II at the end of this presentation after the references.

In Table 1, $f(x_n \cdots x_1)$, $g(x_n \cdots x_1)$ and $h(y_m \cdots y_1)$ are Boolean functions; $\{x_n \cdots x_1\}$ and $\{y_m \cdots y_1\}$ are disjoint sets of variables; $f^0 = f(0, x_{n-1} \cdots x_1)$; and $f^1 = f(1, x_{n-1} \cdots x_1)$. Also, $\alpha \sim (\alpha_n \cdots \alpha_1)$; $D(\alpha)$ is the binary domain of α; $\alpha' \sim (\alpha_{n-1} \cdots \alpha_1)$; $\alpha'' \sim (\alpha_{n-2} \cdots \alpha_1)$; $0\alpha' \sim (0, \alpha_{n-1} \cdots \alpha_1)$ etc.; $\beta \sim (\beta_m \cdots \beta_1)$; $\alpha\beta \sim (\alpha_n \cdots \alpha_1\ \beta_m \cdots \beta_1)$; and the three *omega functions* of a ternary n-tuple $\alpha \sim (\alpha_n \cdots \alpha_1)$, denoted by $\omega_t(\alpha)$ for $t \in \{0, 1, 2\}$, are defined to be the numbers of the t's in $(\alpha_n \cdots \alpha_1)$. Thus $\omega_0(\alpha) + \omega_1(\alpha) + \omega_2(\alpha) = n$.

Table 1 : Properties of the Parity Spectrum

P1. $p_\alpha(f) = \bigoplus_{u \in D(\alpha)} f_u$

P2. $p_\alpha(f \oplus g) = p_\alpha(f) \oplus p_\alpha(g)$ where $f = f(x_n \cdots x_1)$ and $g = g(x_n \cdots x_1)$

P3. $p_\alpha(f \wedge g) = p_\alpha(f) \oplus p_\alpha(g) \oplus p_\alpha(f \vee g)$

P4. $p_\alpha(f \vee g) = p_\alpha(f) \oplus p_\alpha(g) \oplus p_\alpha(f \wedge g)$

P5. $p_\alpha(\overline{f}) = \begin{cases} \overline{p_\alpha(f)} & \text{if } \omega_2(\alpha) = 0 \\ p_\alpha(f) & \text{otherwise} \end{cases}$

P6. (a) $p_{0\alpha'}(f) = p_{\alpha'}(f^0)$

(b) $p_{1\alpha'}(f) = p_{\alpha'}(f^1)$

(c) $p_{2\alpha'}(f) = p_{\alpha'}(f^0) \oplus p_{\alpha'}(f^1)$ where $\alpha' \sim (\alpha_{n-1} \cdots \alpha_1)$

P7. (a) $p_{0\alpha'}(f) = p_{1\alpha'}(f) \oplus p_{2\alpha'}(f)$

(b) $p_{1\alpha'}(f) = p_{0\alpha'}(f) \oplus p_{2\alpha'}(f)$

(c) $p_{2\alpha'}(f) = p_{0\alpha'}(f) \oplus p_{1\alpha'}(f)$

P8. Let $\mathbf{P}(f) = \begin{bmatrix} P^0 \\ P^1 \\ P^2 \end{bmatrix}$ where each P^i is a vector of length 3^{n-1}, then

(a) $P^0 = \mathbf{P}(f^0)$

(b) $P^1 = \mathbf{P}(f^1)$

(c) $P^2 = \mathbf{P}(f^0 \oplus f^1) = \mathbf{P}(f^0) \oplus \mathbf{P}(f^1)$

P9. f is independent of x_n if and only if $p_{2\alpha'} = 0 \ \forall \alpha'$

P10. $p_\alpha(f(x_n \cdots x_1) = p_{\overline{\alpha}_n\alpha'}(f(\overline{x}_n \cdots x_1))$ where $\overline{0} = 1$, $\overline{1} = 0$ and $\overline{2} = 2$

P11. $p_\alpha(f(x_n \cdots x_1)) = p_{\alpha_{n-1}\alpha_n\alpha''}(f(x_{n-1}x_n \cdots x_1))$ where $\alpha'' \sim (\alpha_{n-2} \cdots \alpha_1)$

$$\text{P12. } p_{\alpha\beta}(f \oplus h) = \begin{cases} p_\alpha(f) \oplus p_\beta(h) & \text{if } \omega_2(\alpha) = \omega_2(\beta) = 0 \\ p_\alpha(f) & \text{if } \omega_2(\alpha) > 0 \text{ and } \omega_2(\beta) = 0 \\ p_\beta(h) & \text{if } \omega_2(\alpha) = 0 \text{ and } \omega_2(\beta) > 0 \\ 0 & \text{if } \omega_2(\alpha) > 0 \text{ and } \omega_2(\beta) > 0 \end{cases}$$

where $f = f(x_n \cdots x_1)$ and $h = h(y_m \cdots y_1)$ are variable disjoint.

$$\text{P13. } p_{\alpha\beta}(f \wedge h) = p_\alpha(f) \wedge p_\beta(h)$$

$$\text{P14. } p_{\alpha\beta}(f \vee h) = \begin{cases} p_\alpha(f) \vee p_\beta(h) & \text{if } \omega_2(\alpha) = \omega_2(\beta) = 0 \\ p_\alpha(f) \wedge \overline{p_\beta(h)} & \text{if } \omega_2(\alpha) > 0 \text{ and } \omega_2(\beta) = 0 \\ \overline{p_\alpha(f)} \wedge p_\beta(h) & \text{if } \omega_2(\alpha) = 0 \text{ and } \omega_2(\beta) > 0 \\ p_\alpha(f) \wedge p_\beta(h) & \text{if } \omega_2(\alpha) > 0 \text{ and } \omega_2(\beta) > 0 \end{cases}$$

$$\text{P15. } p_\alpha(x_n \oplus \cdots \oplus x_1) = \begin{cases} 1 & \text{if } \omega_2(\alpha) = 0 \text{ and } \omega_1(\alpha) \text{ is odd, or } \omega_2(\alpha) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{P16. } p_\alpha(x_n \wedge \cdots \wedge x_1) = \begin{cases} 1 & \text{if } \omega_0(\alpha) = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{P17. } p_\alpha(x_n \vee \cdots \vee x_1) = \begin{cases} 1 & \text{if } \omega_2(\alpha) = 0 \text{ and } \alpha \neq 0, \text{ or } \omega_2(\alpha) > 0 \text{ and } \omega_1(\alpha) = 0 \\ 0 & \text{otherwise} \end{cases}$$

Property P1 defines a coefficient in the parity spectrum in terms of a subset of the function values in $T(f)$. Properties P2-5 consider joining functions using operators XOR, AND, OR and NOT respectively. The combined spectra are expressed in terms of those of the individual functions. These four properties can be used simultaneously for more complex combinations. P6 and P8 express the parity spectrum of a function in terms of those of its subfunctions, while P7 relates coefficients within the same spectrum. P9 gives the condition for a function to be independent of a variable. P10-11 show how entries in the parity spectrum are re-arranged under variable permutation and complementation. P12-14 involve combining functions of

disjoint sets of variables. These three properties are used to derive P15-P17, each of which gives the parity spectrum of a simple function involving a single operator. The parity spectra of more complex functions, such as minterms, maxterms, product terms etc. can also be derived using P12-14.

Properties of Boolean differences[Ake59], polarity functions[Muk70], FPEs (e.g.[Fis74]), parity and parity bit signature[Ake88], and *Reed-Muller spectra*[Dam87] are all included in or derivable from Table 1, which provides a solid base for the study of Boolean functions using the parity spectrum.

3.6. Summary

A new concept for defining and examining various properties of a Boolean function using its parity spectrum is introduced. The close relationship between parity spectrum and Boolean difference is investigated. Applications of the parity spectrum to modulo-2 expansion, modulo-2 minimization, and fault detection are discussed. It is seen that the parity spectrum is a simplified tool for unifying and extending many previous results in these areas. Fundamental properties for the parity spectrum are established and proved in Appendix II. They are instrumental in further development of the applications in the following chapters (Chapters 4 to 6).

The parity spectrum is a new approach to the study of Boolean functions. Further research is required to investigate the possibility of applying it to other areas in switching theory not pursued by this research.

CHAPTER 4

THE STRUCTURE OF MODULO-2 CANONICAL EXPANSIONS

4.1. Introduction

In this chapter, new algebraic and geometric representations for *fixed polarity* and *fixed basis modulo-2 canonical expansions (FPEs* and *FBEs)* are developed. The algebraic representation is based on a new, simple, and intuitive result that identifies coefficients in FPEs and FBEs to subfunction parities in the *parity spectrum* described in the preceding chapter. The geometric representation makes use of a new structure called the *parity hypercube (P-cube)*, which graphically displays the parity spectrum, the FPEs, and the FBEs. These new representations provide interesting insights into the structure of and relationships among modulo-2 canonical expansions, and is useful for deriving and interpreting minimization algorithms for these expansions.

Section 4.2 reviews the notation for FPEs and FBEs detailed in Section 2.4.1 of Chapter 2. Section 4.3 derives algebraic representations of FPEs and FBEs from the parity spectrum. Section 4.4 describes the geometric representation using the parity hypercube. Section 4.5 makes use of the parity hypercube to geometrically interpret two approaches to the minimization of FPEs and FBEs.

4.2. FPEs and FBEs

Let $0 \le u, v \le N = 2^n-1$; $u \sim (u_n \cdots u_1)$ and $v \sim (v_n \cdots v_1)$; $\overline{u} = N - u$ is the 1's complement of u so that $\overline{u} \sim (\overline{u}_n \cdots \overline{u}_1)$; $0 \le \alpha \le M = 3^n-1$; and $\alpha \sim (\alpha_n \cdots \alpha_1)$. Let $f(x_n \cdots x_1)$ be a Boolean function; $x_i^0 = x_i$, $x_i^1 = \overline{x}_i$, and $x_i^2 = 1$; and $X^\alpha = x_n^{\alpha_n} \cdots x_1^{\alpha_1}$. From Section 2.3.1, f can be expressed in the *fixed*

polarity modulo-2 canonical expansion of *polarity* v (the v–*FPE*) as:

$$f(x_n \cdots x_1) = \bigoplus_{u=0}^{N} a_u^v \, X^{\bar{u} \oplus v} \tag{1.1}$$

or the *fixed basis modulo-2 canonical expansion* of *basis* α (the α–*FBE*) as:

$$f(x_n \cdots x_1) = \bigoplus_{u=0}^{N} c_u^\alpha \, X^{\bar{u} \oplus \alpha} \tag{1.2}$$

where the *c-star* operator is defined to be $\circledast$: $\{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, 2\}^n$ for FPEs or $\circledast$: $\{0, 1\}^n \times \{0, 1, 2\}^n \rightarrow \{0, 1, 2\}^n$ for FBEs, with the componentwise table of operation:

$\circledast$	0	1	2
0	0	1	0
1	2	2	1

(2)

In vector notation, the v–FPE and α–FBE can be represented by $A^v = [a_0^v \cdots a_N^v]^t$ and $C^\alpha = [c_0^\alpha \cdots c_N^\alpha]^t$ respectively. Since $0 \le v \le N$ and $0 \le \alpha \le M$, f has 2^n FPEs and 3^n FBEs. Note that the FBEs include the FPEs, and the M-FBE is the XOR sum of minterms of f.

4.3. Algebraic Representation

Theorem 1 describes an algebraic representation of the FPEs using the parity spectrum. Its proof requires the following result on the *parity* of a Boolean function:

***Lemma* 1:** Let $f(x_n \cdots x_1)$ be expressed as an XOR sum of AND products, then $p(f) = 1$ if and only if an odd number of minterms appears in the expression.

***Proof*:** Let

$$f(X) = \bigoplus_{j=0}^{k} P_j$$

where the P_j's are n-variable product terms from $\{x_n \cdots x_1\}$. From Property P2 of Table 3–1,

$$p(f) = p(P_1 \oplus \cdots \oplus P_k) = p(P_1) \oplus \cdots \oplus p(P_k)$$

Note that the P_j's are considered as n-variable functions in the variables $\{x_n \cdots x_1\}$. Therefore, from Property P9, $p(P_j) = 0$ if P_j contains fewer than n variables. Otherwise $p(P_j) = 1$ because P_j is a minterm and $T(P_j)$ has a single 1 entry (i.e. the *weight* $w(P_j)$ is equal to 1). Hence the parity of f is same as that of the number of minterms in any XOR sum of product expression. □

***Theorem* 1:** The u-th coefficient a_u^v in the v-FPE of $f(x_n \cdots x_1)$ is equal to the parity $p_{u \oplus v}$ of the subfunction $f_{u \oplus v}$, i.e.,

$$f(x_n \cdots x_1) = \bigoplus_{u=0}^{N} p_{u \oplus v} X^{\overline{u \oplus v}} \tag{3}$$

***Proof*:** Without loss of generality consider $p_{u \oplus v}$ where $u_n = \cdots = u_{k+1} = 0$; and $u_k = \cdots = u_1 = 1$ (i.e. $u = 2^k - 1$). Since $0 \oplus v_i = v_i$ and $1 \oplus v_i = 2$, we have $u \oplus v \sim (v_n \cdots v_{k+1}\ 2 \cdots 2)$. From (1.1),

$$f(x_n \cdots x_1) = a_0^v \oplus a_1^v x_1^{v_1} \oplus a_2^v x_2^{v_2} \oplus a_3^v x_1^{v_1} x_2^{v_2} \oplus \cdots \oplus a_N^v x_1^{v_1} \cdots x_n^{v_n} \tag{4}$$

Therefore,

$$f_{u \oplus v} = f(v_n \cdots v_{k+1}, x_k \cdots x_1)$$

$$= a_0^v \oplus a_1^v x_1^{v_1} \oplus \cdots \oplus a_u^v x_1^{v_1} \cdots x_k^{v_k} \oplus$$

$$a_{u+1} v_{k+1}^{v_{k+1}} \oplus \cdots \oplus a_N^v x_1^{v_1} \cdots x_k^{v_k} v_{k+1}^{v_{k+1}} \cdots v_n^{v_n}$$

Since $v_i^{v_i} = 0^0 = 1^1 = 0$, all terms after $a_u^v x_1^{v_1} \cdots x_k^{v_k}$ vanish. Therefore,

$$f_{u \oplus v} = a_0^v \oplus a_1^v x_1^{v_1} \oplus \cdots \oplus a_u^v x_1^{v_1} \cdots x_k^{v_k}$$

Now the subfunction $f_{u \oplus v}$ is of the k variables $x_k \cdots x_1$, thus it follows from Lemma 1 that

$$p_{u \circledast v} = p(f_{u \circledast v}) = p(a_u^v x_1^{v_1} \cdots x_k^{v_k}) = a_u^v \tag{5}$$

□

Two coefficients a_j^u and a_k^v are said to be *equivalent*, written $a_j^u \equiv a_k^v$, if and only if they are derived from the parity of the same subfunction, otherwise they are *distinct*.

***Corollary* 1:** Let $j \sim (j_n \cdots j_1)$ and $k \sim (k_n \cdots k_1)$. Two coefficients a_j^u and a_k^v from the u–FPE and v–FPE (respectively) of f are equivalent if and only if

(a) $j = k$, and

(b) $u_i \neq v_i$ implies $j_i = k_i = 1$.

***Proof*:** a_j^u and a_k^v are equivalent if and only if $j \circledast u = k \circledast v$. The results (a) and (b) follow from $0 \circledast b \neq 1 \circledast b$ for any $b \in \{0, 1\}$, and $0 \circledast 0 \neq 0 \circledast 1$, respectively. □

Corollary 1 shows that coefficients in the same position from different FPEs of f can be equivalent, but different coefficients from the same FPE must be distinct.

***Example* 1:** $a_3^0 \equiv a_3^2$; $a_N^0 \equiv a_N^1 \equiv \cdots \equiv a_N^N \equiv p_M \equiv p(f)$. □

There are 2^n FPEs each with 2^n coefficients for a total of 4^n coefficients. Since some coefficients are equivalent, there are less than 4^n distinct coefficients.

***Corollary* 2:** There are exactly 3^n distinct coefficients in the 2^n FPEs of $f(x_n \cdots x_1)$. These coefficients form the parity spectrum $\mathbf{P}(f)$.

***Proof*:** Every coefficient a_u^v is identical to a subparity $p_{u \circledast v}$. Since f only has 3^n subfunctions, there are at most 3^n distinct coefficients. Now $\circledast : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1, 2\}^n$ is a many-to-one and onto mapping so that for every $0 \leq \alpha \leq M$ there exists u, v where $0 \leq u, v \leq N$ such that $\alpha = u \circledast v$. Thus every subparity in $\mathbf{P}(f)$ can be identified with one or more FPE coefficients. Therefore, there are exactly 3^n coefficients, corresponding to all subfunction parities in $\mathbf{P}(f)$. □

In [Muk70], coefficients in FPEs are derived algebraically using *polarity functions*. In [Dav71], the polarity functions are shown to be equivalent to Boolean differences and the FPE coefficients are derived from values of these differences. Both of the above formulations involve complex expressions. The relationship between Boolean difference values and the parity spectrum is described in the preceding chapter (Chapter 3), and Theorem 1 is comparable to the equivalent Boolean difference formulation in [Dav71], equation (1). However, the analysis of Theorem 1 is new using simplified notations and proofs to allow coefficients in FPEs to be intuitively interpreted as parities of subfunctions.

Other algebraic representations for FPEs have also been described. Defining the coefficients in terms of *k-subnumbers* in [Fis74] is equivalent to the definition of subfunction parities using the binary domain of a ternary n-tuple (see P1, Table 3–1), with the latter definition being more intuitive and less cumbersome. In [Dam89], the use of *Reed-Muller spectra* creates an extra level of abstraction for interpreting the FPE coefficients.

Besides being more intuitive than previous derivations, the analysis for FPEs above extends to FBEs which algebraic representation have not been described elsewhere.

Theorem 2: The u-th coefficient c_u^{α} in the α-FBE of $f(x_n \cdots x_1)$ is equal to the parity $p_{u \oplus \alpha}$ of the subfunction $f_{u \oplus \alpha}$, i.e.

$$f(x_n \cdots x_1) = \bigoplus_{u=0}^{N} p_{u \oplus \alpha} X^{\overline{u \cap \alpha}} \tag{6}$$

Proof: Without loss of generality let

$$(u_n \cdots u_{r+1}\ u_r \cdots u_{s+1}\ u_s \cdots u_1) = (0 \cdots 0\ 1 \cdots 1\ 1 \cdots 1)$$

and

$$(\alpha_n \cdots \alpha_{r+1}\ \alpha_r \cdots \alpha_{s+1}\ \alpha_s \cdots \alpha_1) = (\alpha_n \cdots \alpha_{r+1}\ 2 \cdots 2\ v_s \cdots v_1)$$

where $0 \le s \le r \le n$. It follows from (2) that

$$u \circledast \alpha \sim (v_n \cdots v_{r+1}\ 1 \cdots 1\ 2 \cdots 2)$$

where $v_i = 0 \circledast \alpha_i \in \{0, 1\}\ \forall\, i,\ n \geq i > r$.

From (1.2)

$$f(x_n \cdots x_1) = \bigoplus_{k=0}^{N} c_k^{\alpha}\, x_n^{\bar{k}_n \oplus \alpha_n} \cdots x_1^{\bar{k}_1 \oplus \alpha_1} \tag{7}$$

Therefore

$$f_{u \oplus \alpha} = f(v_n \cdots v_{r+1},\ 1 \cdots 1,\ x_s \cdots x_1)$$

$$= \bigoplus_{k=0}^{N} c_k^{\alpha}\, v_n^{\bar{k}_n \oplus \alpha_n} \cdots v_{r+1}^{\bar{k}_{r+1} \oplus \alpha_{r+1}} 1^{\bar{k}_r \oplus 2} \cdots 1^{\bar{k}_{s+1} \oplus 2} x_s^{\bar{k}_s \oplus v_s} \cdots x_1^{\bar{k}_1 \oplus v_1}$$

is a function of s variables $x_s \cdots x_1$.

For $n \geq i > r$, since $v_i = 0 \circledast \alpha_i$ and $v_i^{v_i} = 0^0 = 1^1 = 0$, all terms with any $k_i = 1$ vanish and only terms where $0 \leq k < 2^r$ remains. When $k_i = 0$, $v_i^{\bar{k}_i \oplus \alpha_i} = (0 \circledast \alpha_i)^{1 \oplus \alpha_i} = 1$ because $0^2 = 1^2 = 0^1 = 1$. Therefore,

$$f_{u \oplus \alpha} = \bigoplus_{k=0}^{2^r - 1} c_k^{\alpha}\, 1^{\bar{k}_r \oplus 2} \cdots 1^{\bar{k}_{s+1} \oplus 2} x_s^{\bar{k}_s \oplus v_s} \cdots x_1^{\bar{k}_1 \oplus v_1}$$

For $r \geq i > s$, since $\bar{k}_i \circledast 2 = \bar{k}_i$ and $1^1 = 0$, all terms with any $k_i = 0$ vanish. Also, $1^0 = 1$. Therefore,

$$f_{u \oplus \alpha} = \bigoplus_{k=2^r - 2^s}^{2^r - 1} c_k^{\alpha}\, x_s^{\bar{k}_s \oplus v_s} \cdots x_1^{\bar{k}_1 \oplus v_1}$$

For $s \geq i \geq 1$, since $1 \circledast v_i = 2$ (where $v_i \in \{0, 1\}$) and $x_i^2 = 1$, a product term $x_s^{\bar{k}_s \oplus \alpha_s} \cdots x_1^{\bar{k}_1 \oplus \alpha_1}$ involves all the s variables if and only if $k_i = 1\ \forall i$. This requires $k = 2^r - 1 = u$. Since $f_{u \oplus \alpha}$ is a function of s variables, it follows from Lemma 1 that

$$p_{u \oplus \alpha} = p(f_{u \oplus \alpha}) = p(c_u^{\alpha} x_s^{v_s} \cdots x_1^{v_1}) = c_u^{\alpha} \tag{8}$$

□

As for FPEs, two FBE coefficients are equivalent if they are derived from the same subfunction parity, otherwise they are distinct. We can generalize Corollary 1 to show that coefficients from different FBEs (not necessarily from the same position) can be equivalent, but coefficients in the same FBE must be distinct.

Corollary 3: Let $\beta \sim (\beta_n \cdots \beta_1)$. Two coefficients c_j^{α} and c_k^{β} from the α–FBE and the β–FBE respectively are equivalent if and only if

(a) $\alpha = \beta$ and $j = k$, or

(b) $\alpha \neq \beta$ and for each $\alpha_i \neq \beta_i$:

(1) if $\alpha_i = 0$, $\beta_i = 1$, then $j_i = k_i = 1$,

(2) if $\alpha_i = 0$, $\beta_i = 2$, then $j_i = k_i = 0$,

(3) if $\alpha_i = 1$, $\beta_i = 2$, then $j_i = k_i = 0$.

Proof: c_j^{α} and c_k^{β} are equivalent if and only if $j \circledast \alpha = k \circledast \beta$. Result (a) follows because $0 \circledast \alpha_i \neq 1 \circledast \alpha_i$. Similarly, result (b) follows from the table of operations of $\circledast$ in (2). Note that all possible values of α_i and β_i are covered by this corollary, due to symmetry of the $\equiv$ operation. □

Example 2: $c_4^5 \equiv c_6^{15} = p_{210} = p(f(x_3,1,0))$ since $100_2 \circledast 012_3 = 110_2 \circledast 120_3 = 210_3$. □

Because of equivalence, there are less than 6^n distinct coefficients in the 3^n FBEs.

Corollary 4: There are exactly 3^n distinct coefficients in the 3^n FBEs of $f(x_n \cdots x_1)$. These coefficients form the parity spectrum $P(f)$.

Proof: Similar to corollary 2. □

FBEs are more general than FPEs, but have received much less attention. They are used in [Bio73] to obtain a minimal modulo-2 expansion for any 4-variable function.

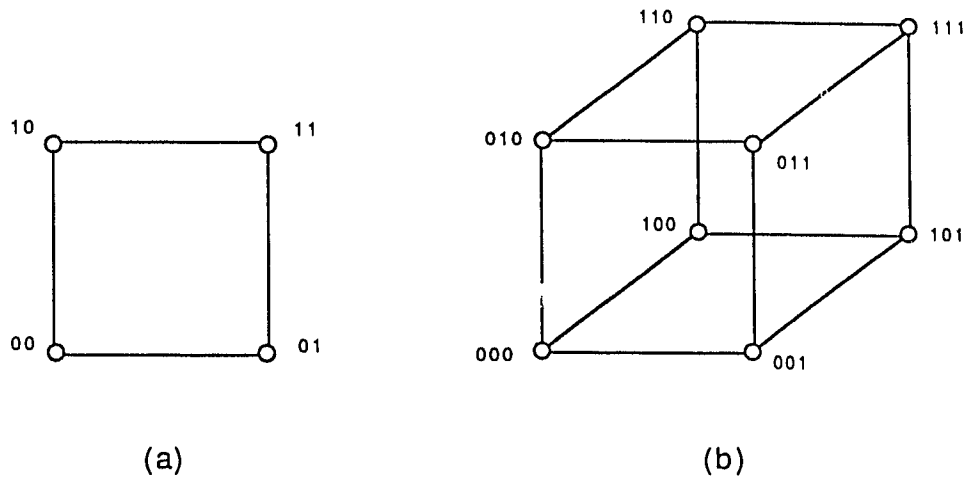


Figure 1: Boolean Hypercubes

4.4. Geometric Representation

Boolean functions and algorithms that manipulate them are usually described algebraically. However, it is often found that geometric representations of Boolean functions are easier to comprehend than algebraic ones. While algebraic formulations are suitable for mathematical analysis and computer automation, geometric representations offer insights into the underlying structure which may be difficult or impossible to be described in algebraic terms. These insights are useful in the interpretation of existing algorithms and in the derivation of new ones.

In this section, a new geometric representation for a Boolean function and its parity spectrum is introduced. It is also a representation for all the FPEs and FBEs,

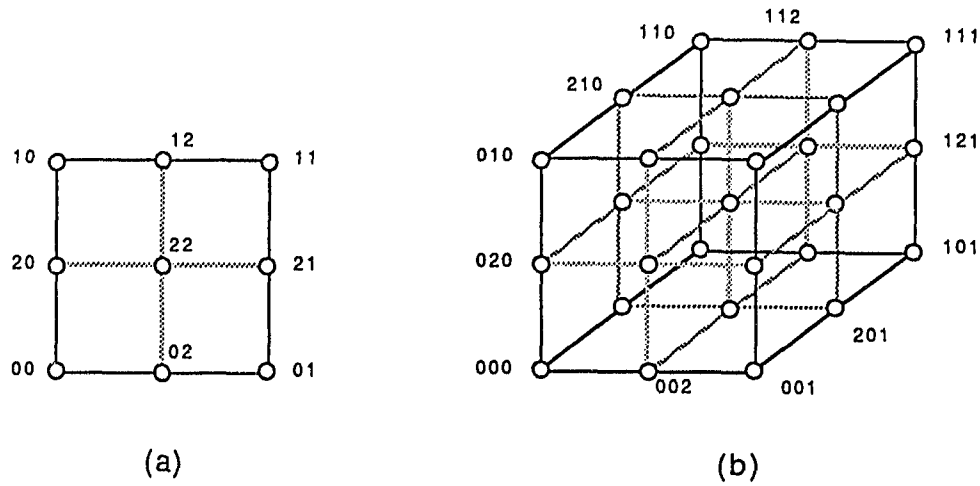


***Figure* 2:** Ternary Hypercubes

and allows the structures of and relationships among these expansions to be visualized.

An n-dimensional *Boolean hypercube* is defined on a co-ordinate system with 2^n co-ordinates $(u_n \cdots u_1)$ where $u_i \in \{0, 1\}$ (Figure 1). Analogously, an n-dimensional *ternary hypercube* is defined with 3^n co-ordinates $(\alpha_n \cdots \alpha_1)$ where $\alpha_i \in \{0, 1, 2\}$ (Figure 2). As illustrated in Figure 2, we abandon the natural order in the domain of α_i and use 0, 2, 1 instead of 0, 1, 2 in ordering the co-ordinates. This allows a ternary hypercube (Figure 2) to be viewed as an expansion of a Boolean hypercube (Figure 1) with the latter forming an enclosure around the former.

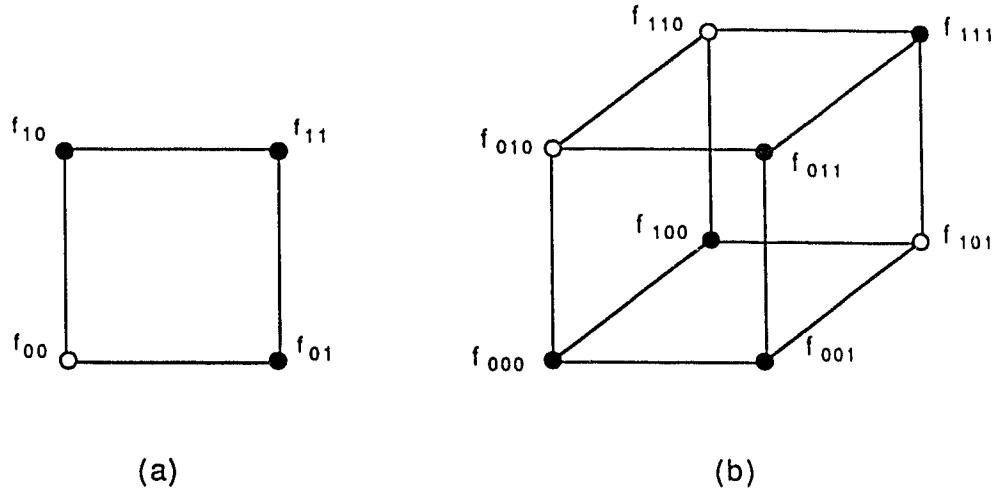


***Figure* 3:** T-cube Representations

A well-known geometric representation for an n-variable function $f(x_n \cdots x_1)$ uses an n-dimensional Boolean hypercube described in Section 2.2 as the *truth hypercube*, or *T-cube*. The 2^n function values in the truth vector $T(f) = [f_0 \cdots f_N]^t$, where $N = 2^n-1$, are mapped to the 2^n vertices in the T-cube such that a vertex with coordinate $(u_n \cdots u_1) \sim u$ is marked black or white depending on whether the function value f_u in $T(f)$ is 1 or 0. The T-cubes for two example functions $f(x_2, x_1) = x_1 \vee x_2$ and $g(x_3, x_2, x_1) = \overline{x}_1\overline{x}_2 \vee x_1x_2 \vee x_1\overline{x}_3$ of 2- and 3-variables respectively are given in Figure 3.

The T-cube can be interpreted as a Boolean hypercube representation of all 0-variable subfunction parities of f. Expand it as described earlier into a ternary

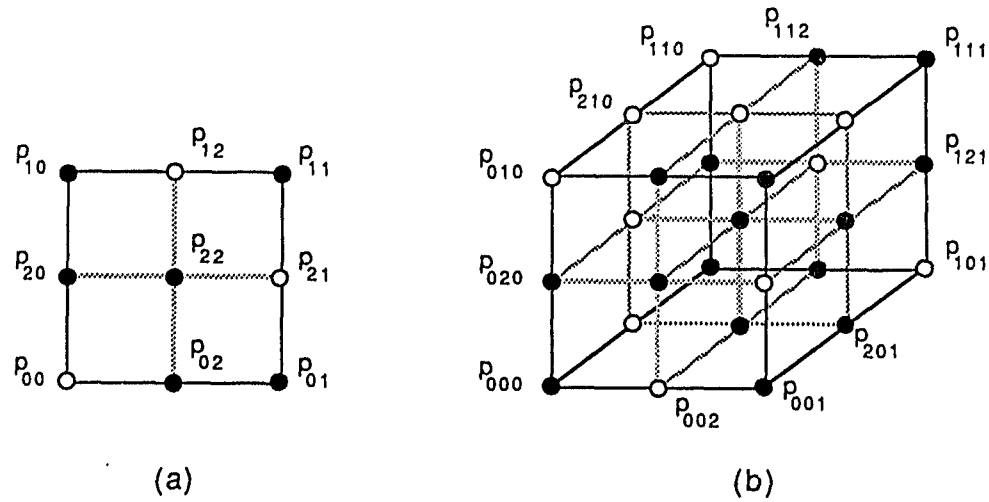


***Figure* 4:** P-cube Representations

hypercube of 3^n vertices to represent all the subfunction parities in $P(f)$. In this what we call the *parity hypercube*, or *P-cube*, representation of f, each vertex with co-ordinate $(\alpha_n \cdots \alpha_1) \sim \alpha$ in the P-cube is mapped to subfunction parity $p_\alpha = p(f_\alpha)$, and is marked black or white depending on whether $p_\alpha = 1$ or $p_\alpha = 0$. For example, T-cubes in Figure 3 can be expanded to form the P-cubes in Figure 4.

From Property P7(c) of Table 3–1, $p_{\alpha_n \cdots \alpha_{i+1} 2 \alpha_{i-1} \cdots \alpha_1} = p_{\alpha_n \cdots \alpha_{i+1} 0 \alpha_{i-1} \cdots \alpha_1} \oplus p_{\alpha_n \cdots \alpha_{i+1} 1 \alpha_{i-1} \cdots \alpha_1}$. As is evident from Figure 4, values of vertices (i.e. black or white) in a P-cube are computed as XORs of two neighboring vertex values, starting from those in the enclosing T-cube. In Figure 4(a) for example,

$p_{00} = f_{00}$; $p_{01} = f_{01}$; $p_{10} = f_{10}$; $p_{11} = f_{11}$;

$p_{02} = p_{00} \oplus p_{01}$; $p_{20} = p_{00} \oplus p_{10}$;

$p_{12} = p_{10} \oplus p_{11}$; $p_{21} = p_{01} \oplus p_{11}$;

$p_{22} = p_{02} \oplus p_{12} = p_{20} \oplus p_{21}$

The usefulness of a P-cube lies in its ability to represent FPEs and FBEs. Consider first the $2^2 = 4$ FPEs of a two-variable function $f(x_2, x_1)$. From Theorem 1, the coefficients are:

$A^0 = [p_{00}\ p_{02}\ p_{20}\ p_{22}]^t$

$A^1 = [p_{01}\ p_{02}\ p_{21}\ p_{22}]^t$

$A^2 = [p_{10}\ p_{12}\ p_{20}\ p_{22}]^t$

$A^3 = [p_{11}\ p_{12}\ p_{21}\ p_{22}]^t$

As illustrated in Figure 5 (for the P-cube in Figure 4(a)), any 2-dimensional P-cube can be divided into 4 2-dimensional hypercubes each of which contains vertices that correspond to coefficients in an FPE. In other words, a P-cube simultaneously represents all the FPEs of a Boolean function.

The extension to FBEs in a 2-dimensional P-cube is straightforward. From Theorem 2, the $3^2 = 9$ FBEs of $f(x_2, x_1)$ are:

$C^0 = A^0$; $C^1 = A^1$; $C^3 = A^2$; $C^4 = A^3$

$C^2 = [p_{00}\ p_{01}\ p_{20}\ p_{21}]^t$

$C^5 = [p_{10}\ p_{11}\ p_{20}\ p_{21}]^t$

$C^6 = [p_{00}\ p_{02}\ p_{10}\ p_{12}]^t$

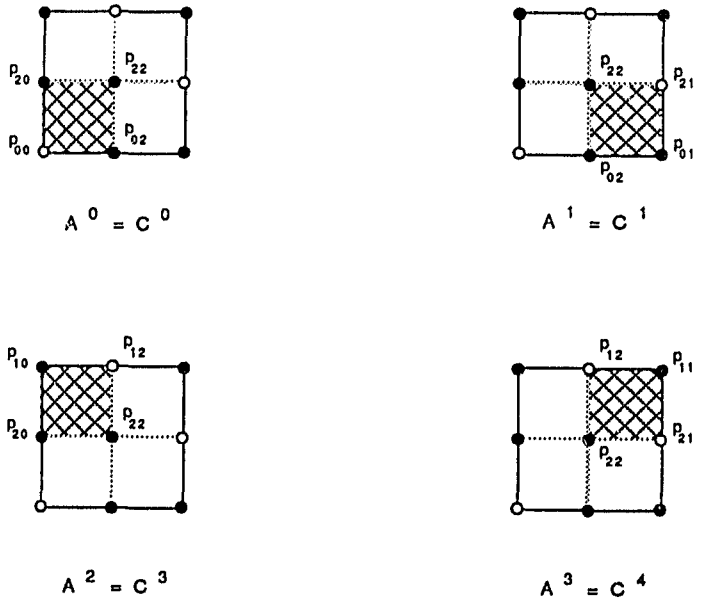


Figure 5: FPEs in a P-cube

$C^7 = [p_{01}\ p_{02}\ p_{11}\ p_{12}]^t$

$C^8 = [p_{00}\ p_{01}\ p_{10}\ p_{11}]^t = T(f)$

In Figures 5 and 6, these FBEs correspond to all the 9 possible Boolean hypercubes embedded within a 2-dimensional P-cube. Thus a P-cube is a representation for all the FBEs of the considered function.

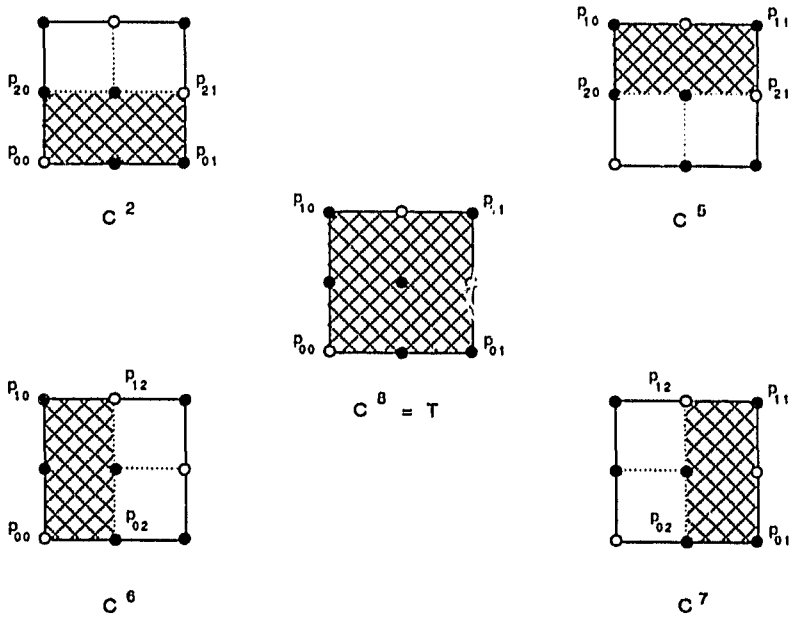


Figure 6: Other FBEs in a P-cube

In general, an n-dimensional P-cube of an n-variable function $f(x_n \cdots x_1)$ embeds 3^n Boolean hypercubes each of which represents an FBE of f. The Boolean hypercube representing the α-FBE (for some $\alpha \sim (\alpha_n \cdots \alpha_1)$, $0 \leq \alpha \leq M$) is identified (in the P-cube) as follows. Consider a P-cube as a collection of segments of two edges connecting 3 vertices such that all segments along dimension i connects vertex triplets of the form $v_0 = p_{\alpha_n \cdots \alpha_{i+1} 0 \alpha_{i-1} \cdots \alpha_1}$, $v_2 = p_{\alpha_n \cdots \alpha_{i+1} 2 \alpha_{i-1} \cdots \alpha_1}$, and $v_1 = p_{\alpha_n \cdots \alpha_{i+1} 1 \alpha_{i-1} \cdots \alpha_1}$ (see Figure 7). From Theorem 2, the Boolean hypercube for

the α–FBE is composed of the vertex pair (v_0 , v_2) when $\alpha_i = 0$; (v_1 , v_2) when $\alpha_i = 1$; or (v_0 , v_1) when $\alpha_i = 2$.

A P-cube representation ceases to be a useful visual aid when $n > 3$, but the insights gained in interpreting 2- and 3-dimensional P-cubes can be carried over to higher dimensions without difficulty. In particular, the orderly arrangement of FPEs and FBEs within the parity spectrum and the relationship among FPEs of different polarities and FBEs with different bases are graphically illustrated by a P-cube representation. As such it is useful for deriving and interpreting minimization algorithms for selecting optimal FPEs and FBEs, to be discussed in the next section.

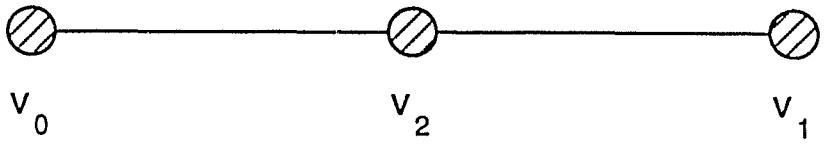
4.5. Modulo-2 Minimization

The P-cube representation provides interesting insights into two minimization algorithms for FBEs. These algorithms selects from among the 3^n FBEs of an n-variable function the one(s) with the least number of terms in order to obtain an economical AND-XOR implementation. They can be modified to select an optimal FPE from among the 2^n FPEs. In this section, we describe only the geometric interpretation of these algorithms as related to the P-cube representation, and leave their mathematical and algorithmic details to the next chapter (Chapters 5), using Boolean matrix transforms.

4.5.1. Gray Code Approach

The *distance* $d(\alpha, \beta)$ between two ternary n-tuples $(\alpha_n \cdots \alpha_1) \sim \alpha$ and $(\beta_n \cdots \beta_1) \sim \beta$ is the number of places where $\alpha_i \neq \beta_i$. If $d(\alpha, \beta) = 1$, α and β are said to be *adjacent*.

In the P-cubes, two FBEs with adjacent bases are represented by intersecting Boolean hypercubes sharing half of their vertices. For example, in Figure 5, since (0, 0) is adjacent to (0, 1), (0, 2), (1, 0), and (2, 0), so C^0 shares half of its coefficients with C^1, C^2, C^3, or C^6. This observation can also be extended to 3-dimensional P-cubes, and generalized to higher dimensions. Furthermore, any two (distinct) Boolean hypercubes embedded within a P-cube can share at most half of



$$v_0 = p\alpha_n \cdots \alpha_{i+1} 0 \alpha_{i-1} \cdots \alpha_1$$
$$v_1 = p\alpha_n \cdots \alpha_{i+1} 1 \alpha_{i-1} \cdots \alpha_1$$
$$v_2 = p\alpha_n \cdots \alpha_{i+1} 2 \alpha_{i-1} \cdots \alpha_1$$

Figure 7: A segment in the i-th dimension of a P-cube

their vertices, when the hypercubes represent FBEs with adjacent bases.

The above observations lead to a new, efficient FBE minimization algorithm which explicitly computes the 3^n FBEs of an n-variable function one at a time, and selects from among them the one(s) with the least number of terms. The FBEs are generated in a *ternary Gray code* [Flo56] sequence of the bases. (For example, a ternary Gray code for 2-tuples is {(0, 0), (0, 1), (0, 2), (1, 2), (1, 1), (1, 0), (2, 0), (0, 1), (2, 2)}.) This sequence allows each new FBE to be computed from another with an adjacent basis. Since the two FBEs share half of their coefficients,, the amount of computational work is minimized.

This FBE minimization algorithm can be modified to minimize FPEs only, using a *binary Gray code* sequence of the 2^n polarities. The binary Gray code method has been reported in various forms, including one based on folding coefficient maps[WuC82] that are simply planar projections of the Boolean hypercubes

embedded in the P-cube. A survey of the development of the binary Gray code algorithm for FPEs and the generalization obtained in this research to the ternary Gray code algorithm for FBEs are described in a unified manner in the next chapter, using matrix transforms.

4.5.2. Parity Spectrum Approach

Define the *weight* of an FBE to be its number of non-zero terms. Thus the weight of the α–FBE of f is the arithmetic sum of coefficients in $C^{\alpha}(f)$. In the Gray code algorithm, all the FBEs are explicitly generated, and the weight of each FBE is computed separately. Since the FBEs (of an n-variable function) share coefficients among themselves, and the parity spectrum implicitly contains all the FBEs, it may be more efficient to first compute the parity spectrum, and then compute the 3^n weights simultaneously from the parity spectrum.

As is often the case, suppose a function f is defined by its truth vector $T(f)$. The minimization algorithm requires two transforms. The first transform computes $P(f)$ from $T(f)$ and can be visualized as the expansion of a T-cube into a P-cube using XORs (modulo-2 additions) as described earlier. The second transform computes the weight of each FBE from the P-cube by arithmetic addition of vertex values. Because FBEs share coefficients, parts of the weight of one FBE contribute to weights of other FBEs. Thus the number of arithmetic additions required can be reduced by exploiting such overlaps in the weights.

Although the P-cube representation provides an intuitive framework, fast computations of the two transforms are best described by recursive matrices. These transforms has first been deduced [Bio73] using a Boolean difference approach. A simplified derivation in terms of the parity spectrum is described in the next chapter. As with the Gray code approach, the parity spectrum method can be modified to minimize FPEs only.

4.6. Summary

In this chapter, coefficients in FPEs and FBEs are identified with subfunction parities in the parity spectrum. The algebraic formulation thus obtained can also be represented geometrically using a ternary hypercube called the parity hypercube (P-cube). The P-cube representation of the parity spectrum, the FPEs, and the FBEs allows the geometric interpretations of two approaches to the minimization of FPEs or FBEs. Details on these minimization algorithms are described in the next chapter (Chapter 5), using Boolean matrix transforms.

Cohn[Coh60, Coh62] and Davio et al.[Dav78] mention that there exist other modulo-2 canonical forms besides the FPEs and FBEs. It is not clear if our algebraic and geometric representations based on the parity spectrum can be extended from FPEs and FBEs to these other canonical forms.

CHAPTER 5

MATRIX TRANSFORMS FOR MODULO-2 MINIMIZATION

5.1. Introduction

This chapter describes the Boolean matrix transforms among the *fixed polarity* and *fixed basis modulo-2 canonical expansions (FPEs* and *FBEs)* of a Boolean function. It also describes transforms between the *parity spectrum* and the FPEs and FBEs, as well as transforms from the parity spectrum to the *weights* of these expansions. These transforms, described as *Kronecker products*[Gra81] of elementary matrices, are applied in two different approaches to the minimization of FPEs and FBEs.

Calingeart[Cal61] first considers Boolean matrix transforms among canonical expansions of a Boolean function. Lechner[Lec63] proposes to describe these transforms in terms of Kronecker products of elementary matrices in order to derive fast algorithms for computing these transforms. Most results on matrix transforms among FPEs and their applications to FPE minimization are first developed by Davio[Dav71] and Bioul et al.[Bio73] (see also the book by Davio et al.[Dav78], who summarize and enhance these results), but less general results are published by later authors[Swa72, Sal79, WuC82, Bes83, Zha84, Gre87] using different sets of notations and terminologies.

In this chapter, the transforms among FPEs as well as FBEs are described in a unified manner, and recursive algorithms for the fast computation of these transforms are presented. Using these transforms, the *Gray code* approach for the minimization of FPEs is reviewed and generalized to FBEs.

Matrix transforms between the *parity spectrum* and the FPEs and FBEs and from the parity spectrum to the *weights* of all the FPEs or FBEs are also described. These transforms lead to another approach to the minimization of FPEs and FBEs using the parity spectrum. This approach is faster than the Gray code approach but requires more storage.

Section 5.2 derives square transforms among FPEs and FBEs, while Section 5.3 derives rectangular transforms between the parity spectrum and the FPEs and FBEs. The minimization of FPEs and FBEs are described in Section 5.4. For the purpose of comparison, a brute-force approach is considered before the Gray code and parity spectrum approaches.

5.2. Transforms among FPEs and FBEs

5.2.1. The Transforms

This section describes square Boolean matrix transforms among FPEs and FBEs using Kronecker products of nine elementary order-2 matrices. Denote these nine matrices by e_{00}, e_{01}, e_{02}, e_{10}, $\cdots$, e_{22} where $e_{00} = e_{11} = e_{22} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$; $e_{01} = e_{10} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$; $e_{02} = e_{20} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$; $e_{12} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$; and $e_{21} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$. They have two important properties under modulo-2 multiplication:

Lemma 1: Let $\alpha, \beta, \gamma \in \{0, 1, 2\}$, then

(a) $e_{\gamma\alpha} e_{\beta\gamma} = e_{\beta\alpha}$

(b) $e_{\alpha\beta}^{-1} = e_{\beta\alpha}$

where matrix multiplications are performed in modulo-2.

Proof:

(a) By exhaustively checking the 27 possible combinations of α, β, and γ, one can show that $e_{00}e_{00} = e_{10}e_{01} = e_{20}e_{02} = e_{00}$; $e_{00}e_{10} = e_{10}e_{11} = e_{20}e_{12} = e_{10}$; $\cdots$; and $e_{02}e_{20} = e_{12}e_{21} = e_{22}e_{22} = e_{22}$.

(b) From (a), $e_{\alpha\beta}e_{\beta\alpha} = e_{\beta\beta} = e_{\beta\alpha}e_{\alpha\beta} = e_{\alpha\alpha} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Therefore, $e_{\alpha\beta}^{-1} = e_{\beta\alpha}$. □

Let the vectors of coefficients in the α– and β–FBEs of $f(x_n \cdots x_1)$ be $C^{\alpha}(f)$ and $C^{\beta}(f)$ respectively, where $C^{\alpha}(f) = [c_0^{\alpha} \cdots c_N^{\alpha}]$, $C^{\beta}(f) = [c_0^{\beta} \cdots c_N^{\beta}]$, and $N = 2^n-1$. The transform from $C^{\beta}(f)$ to $C^{\alpha}(f)$ can be expressed as the Kronecker products of n elementary matrices from $e_{00} \cdots e_{22}$ as follows:

Theorem 1:

$$C^{\alpha}(f) = \left[\bigotimes_{i=n}^{1} e_{\beta_i\alpha_i} \right] C^{\beta}(f) \tag{1}$$

where $\left[\bigotimes_{i=n}^{1} e_{\beta_i\alpha_i} \right] = e_{\beta_n\alpha_n} \otimes \cdots \otimes e_{\beta_1\alpha_1}$ is of order $2^n \times 2^n$ and matrix multiplications are performed in modulo-2.

Proof: From Theorem 1 of Bioul et al.[Bio73], we have

$$C^{\beta}(f) = \left[\bigotimes_{i=n}^{1} e_{2\beta_i} \right] T(f) \tag{2}$$

where $T(f) = C^{M}(f)$ is the truth vector of f. Using Lemma 2–1(c) in Section 2.5 of Chapter 2 and Lemma 1(b) above,

$$T(f) = \left[\bigotimes_{i=n}^{1} e_{2\beta_i} \right]^{-1} C^{\beta}(f) = \left[\bigotimes_{i=n}^{1} e_{2\beta_i}^{-1} \right] C^{\beta}(f) = \left[\bigotimes_{i=n}^{1} e_{\beta_i 2} \right] C^{\beta}(f)$$

From (2) and the associativity of modulo-2 matrix multiplication, we have

$$C^{\alpha}(f) = \left[\bigotimes_{i=n}^{1} e_{2\alpha_i} \right] T(f) = \left[\bigotimes_{i=n}^{1} e_{2\alpha_i} \right] \left\{ \left[\bigotimes_{i=n}^{1} e_{\beta_i 2} \right] C^{\beta}(f) \right\}$$

$$= \left\{ \left[\bigotimes_{i=n}^{1} e_{2\alpha_i} \right] \left[\bigotimes_{i=n}^{1} e_{\beta_i 2} \right] \right\} C^{\beta}(f)$$

Finally, it follows from Lemma 2–1(b) and and Lemma 1(a) that

$$C^{\alpha}(f) = \left[\bigotimes_{i=n}^{1} e_{2\alpha_i e_{\beta_i 2}} \right] C^{\beta}(f) = \left[\bigotimes_{i=n}^{1} e_{\beta_i \alpha_i} \right] C^{\beta}(f)$$

□

Corollary 1:

$$\left[\bigotimes_{i=n}^{1} e_{\beta_i \alpha_i} \right] = \left[\bigotimes_{i=n}^{1} e_{\alpha_i \beta_i} \right]^{-1} \tag{3}$$

Proof: Let I_n be the identity matrix of order 2^n. From Lemma 2–1(b) of Chapter 2 and Lemma 1(a),

$$\left[\bigotimes_{i=n}^{1} e_{\beta_i \alpha_i} \right] \left[\bigotimes_{i=n}^{1} e_{\alpha_i \beta_i} \right] = \left[\bigotimes_{i=n}^{1} e_{\beta_i \alpha_i} e_{\alpha_i \beta_i} \right] = \bigotimes_{i=n}^{1} I_1 = I_n$$

Similarly,

$$\left[\bigotimes_{i=n}^{1} e_{\alpha_i \beta_i} \right] \left[\bigotimes_{i=n}^{1} e_{\beta_i \alpha_i} \right] = I_n$$

□

Alternatively, Corollary 1 can be proved from Theorem 1 and the associativity of modulo-2 matrix multiplication, since $C^{\alpha}(f) = \left[\bigotimes_{i=n}^{1} e_{\beta_i \alpha_i} \right] \left[\bigotimes_{i=n}^{1} e_{\alpha_i \beta_i} \right] C^{\alpha}(f)$.

Corollary 2: [Dav71, Bio73]

(a) Let $s_0 = e_{20} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ and $s_1 = e_{21} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$, then

$$A^{v}(f) = \left[\bigotimes_{i=n}^{1} s_{v_i} \right] T(f) \tag{4.1}$$

where $A^{v}(f)$ is the vector of coefficients from the v-FPE of $f(x_n \cdots x_1)$.

(b) Let $s_2 = e_{22} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, then

$$C^{\alpha}(f) = \left[\bigotimes_{i=n}^{1} s_{\alpha_i} \right] T(f) \tag{4.2}$$

where $C^{\alpha}(f)$ is the vector of coefficients from the α–FBE of $f(x_n \cdots x_1)$. □

Davio[Dav71] establishes Corollary 2(a), which is later generalized to Corollary 2(b) by Bioul et al.[Bio73]. Davio et al.([Dav78], Theorem 4.13) also describe transforms between any two FPEs. However, these authors employ Kronecker products of symbolic matrices in order to obtain algebraic expressions after matrix multiplications. The formulae thus derived are cumbersome and difficult to use. Indeed, these results seem to have been lost sight by later authors[Swa72, Sal79, WuC82, Bes83, Zha84, Gre87] who implicitly or explicitly repeat the description of some of these transforms using different sets of notations.

Theorem 1 is a generalized result since it covers all possible transforms among FPEs and FBEs. The use of a Kronecker product structure in terms of the elementary matrices allows all the transforms to be compactly and coherently described in a single theorem. It also allows a simple algorithmic description of the fast transforms as shown below.

5.2.2. The Fast Transforms

Many fast algorithms for computing part of the transforms covered by Theorem 1 have been described in the literature[Lec63, Dav71, Bio73, Dav78, WuC82, Bes83, Zha84, Gre87]. These fast transforms are much more efficient than full matrix multiplications. Unfortunately, very different sets of notations and terminologies have been used and there is an overlap of results between the various authors. Furthermore, the discussions are often informal and an algorithmic description suitable for direct computer implementation has yet to appear.

As pointed out by Lechner[Lec63], the Kronecker product structure of the transform matrices can be used to formulate fast transforms. In Theorem 1, there are

$3^n \cdot 3^n = 3^{2n}$ possible combinations of α and β and hence 3^{2n} possible transform matrices, but their similar structure calls for a homogeneous treatment of their fast transforms. Algorithm 1 describes, in pseudo-PASCAL notation, a fast recursive procedure for computing efficiently any of the 3^{2n} transforms.

Algorithm 1 is self-explanatory and only the recursive steps need to be described further. First we establish a simple result on the Kronecker product involving an order-2 matrix A. Let $A = \left[a_{ij}\right]_{2\times 2}$; $B = \left[b_{ij}\right]_{2^{n-1}\times 2^{n-1}}$; $C = [c_0 \cdots c_N]^t$; $C^0 = [c_0 \cdots c_{2^{n-1}-1}]^t$; and $C^1 = [c_{2^{n-1}} \cdots c_N]^t$, then it can be shown that

$$(A \otimes B)C = \begin{bmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{bmatrix} \begin{bmatrix} C^0 \\ C^1 \end{bmatrix} = \begin{bmatrix} a_{11}BC^0 \oplus a_{12}BC^1 \\ a_{21}BC^0 \oplus a_{22}BC^1 \end{bmatrix} \tag{5}$$

where $\oplus$ in (5) above denotes elementwise modulo-2 addition between two $2^{n-1}\times 1$ column vectors. Using Theorem 1, Lemma 1(a) and the above result, the recursive calls in Algorithm 1 can easily be constructed.

Previous fast transforms consider only those from $T(f)$ to the 0-FPE[Lec63, WuC82, Bes83, Zha84, Gre87], to any FPE[Dav71, Dav78], or to any FBE[Bio73, Dav78]; from the 0-FPE to any other FPE[Gre87]; or between FPEs with adjacent polarities[Dav71, Dav78, WuC82, Bes83, Zha84, Gre87]. Algorithm 1 is a unified description since it encompasses all the previous fast transforms. It is a generalized result since Theorem 1 is generalized. Its compact and algorithmic description makes it amenable to computer implementation and complexity analysis.

It is important to perform a complexity analysis on the fast transforms since they are used in minimization algorithms for FPEs and FBEs. A full matrix multiplication for a transform in Theorem 1 would require 4^n modulo-2 additions (XORs). We show in the next theorem that a fast transform using Algorithm 1 takes at most $n \cdot 2^{n-1}$ XORs. In other words, the fast transform is at least $\frac{2^{n+1}}{n}$ times faster than a full matrix multiplication.

Algorithm 1: Fast Transforms for Computing $C^{\alpha}(f)$ from $C^{\beta}(f)$

```
type
  BoolVec : array[0 · · · 2^n−1] of boolean;
  ntrits : array[1 · · · n] of 0 · · · 2;
procedure Trans(n:integer; var C^α:BoolVec; C^β:BoolVec; α, β:ntrits);
  procedure Trans-Aux(n,lba,lbb:integer);
  begin
    if n = 1 then
      a := C^β[lbb]; b := C^β[lbb + 1];
      case (α_n β_n) of
        00,11,22:C^α[lba] := a; C^α[lba+1] := b;
        01,10: C^α[lba] := a ⊕ b; C^α[lba+1] := b;
        02,20: C^α[lba] := a; C^α[lba+1] := a ⊕ b;
        12:    C^α[lba] := a ⊕ b; C^α[lba+1] := a;
        21:    C^α[lba] := b; C^α[lba+1] := a ⊕ b;
      end
    else
      case (α_n β_n) of
        00,11,22:Trans-Aux(n-1,lba,lbb); Trans-Aux(n-1,lba + 2^(n−1),lbb + 2^n−1);
        01,10: Trans-Aux(n-1,lba,lbb); Trans-Aux(n-1,lba + 2^(n−1),lbb + 2^n−1);
               for k := 0 to 2^(n−1)−1 do
                 C^α[lba + k] := C^α[lba + k] ⊕ C^α[lba + 2^(n−1) + k];
        02,20: Trans-Aux(n-1,lba,lbb); Trans-Aux(n-1,lba + 2^(n−1),lbb + 2^(n−1));
               for k := 0 to 2^(n−1)−1 do
                 C^α[lba + 2^(n−1) + k] := C^α[lba + k] ⊕ C^α[lba + 2^(n−1) + k];
        12:    Trans-Aux(n-1,lba,lbb + 2^(n−1)); Trans-Aux(n-1,lba + 2^(n−1),lbb);
               for k := 0 to 2^(n−1)−1 do
                 C^α[lba + k] := C^α[lba + k] ⊕ C^α[lba + 2^(n−1) + k];
        21:    Trans-Aux(n-1,lba,lbb + 2^(n−1)); Trans-Aux(n-1,lba + 2^(n−1),lbb);
               for k := 0 to 2^(n−1)−1 do
                 C^α[lba + 2^(n−1) + k] := C^α[lba + k] ⊕ C^α[lba + 2^(n−1) + k];
      end
  end Trans-Aux;
begin
  Trans-Aux(n, 0, 0);
end Trans;
```

Theorem 2: The fast transform from C^{β} to C^{α} takes $m \cdot 2^{n-1}$ modulo-2 additions, where $m = d(\alpha,\beta)$ is the *distance* between the bases α and β; and n is the number of variables in the considered function.

Proof: From Algorithm 1, the recurrence relation for T_n, the number of modulo-2 additions for the fast transform of $C^{\beta} \rightarrow C^{\alpha}$, satisfies

$$T_n = \begin{cases} 2 \cdot T_{n-1} & \text{if } \alpha_n = \beta_n \\ 2 \cdot T_{n-1} + 2^{n-1} & \text{otherwise} \end{cases} \tag{6.1}$$

and

$$T_1 = \begin{cases} 0 & \text{if } \alpha_1 = \beta_1 \\ 1 & \text{otherwise.} \end{cases} \tag{6.2}$$

Using (6.1) and (6.2), we can prove Theorem 2 by induction on n as follows:

(a) First consider n = 1 where the bases in the transform are $\alpha \sim (\alpha_1)$ and $\beta \sim (\beta_1)$, then

$$T_1 = \begin{cases} 0 & \text{if } d(\alpha_1, \beta_1) = 0 \\ 1 & \text{if } d(\alpha_1, \beta_1) = 1 \end{cases}$$

Therefore, $T_n = m \cdot 2^{n-1}$ for n = 1. Thus the theorem is true for n = 1.

(b) As the induction hypothesis, suppose $T_{n-1} = m' \cdot 2^{n-2}$ for the fast transform $C^{\beta'} \rightarrow C^{\alpha'}$ involving any two (n-1)-variable bases $\alpha' \sim (\alpha_{n-1} \cdots \alpha_1)$ and $\beta' \sim (\beta_{n-1} \cdots \beta_1)$ whose distance is $m' = d(\alpha', \beta')$. Now consider T_n for the transform involving bases $\alpha \sim (\alpha_n \cdots \alpha_1)$ and $\beta \sim (\beta_n \cdots \beta_1)$.

(1) If $\alpha_n = \beta_n$, then $d(\alpha', \beta') = d(\alpha,\beta) = m' = m$, By the induction hypothesis, $T_{n-1} = m \cdot 2^{n-2}$. Therefore, $T_n = 2 \cdot T_{n-1} = m \cdot 2^{n-1}$.

(2) If $\alpha_n \neq \beta_n$, then $m' = m - 1$. By the induction hypothesis, $T_{n-1} = (m-1) \cdot 2^{n-2}$. Therefore, $T_n = 2 \cdot T_{n-1} + 2^{n-1} = (m-1) \cdot 2^{n-1} + 2^{n-1} = m \cdot 2^{n-1}$.

From (1) and (2), the theorem is true for n variables if it is true for (n-1) variables.

From (a) and (b), the theorem is true for all $n \geq 1$. □

Corollary 3:

(a) The fast transform of $\mathbf{T}(f) \rightarrow \mathbf{A}^u(f)$ takes $n \cdot 2^{n-1}$ modulo-2 additions.

(b) The fast transform of $\mathbf{A}^v(f) \rightarrow \mathbf{A}^u(f)$ takes $m \cdot 2^{n-1}$ modulo-2 additions, where $m = d(u,v)$. □

The analysis as in Corollary 3(a) and the case m = 1 in Corollary 3(b) is first carried out by Davio et al.[Dav78] and later repeated by others[Bes83, Zha84, Gre87]. Green[Gre87] also analyses the fast transform from $\mathbf{A}^0(f)$ to $\mathbf{A}^u(f)$. Thus Theorem 2 extends all the previous analysis to cover all possible transforms among FPEs and FBEs.

5.3. Transforms for the Parity Spectrum

5.3.1. The Forward Transforms

The transform from the α–FBE $\mathbf{C}^\alpha(f)$ to the parity spectrum $\mathbf{P}(f)$, denoted by the Boolean matrix $\mathbf{L}_\alpha$, can also be described as Kronecker products of elementary Boolean matrices. Unlike the transforms among FBEs, these matrices are rectangular rather than square.

For brevity, let

$$e_0 = e_{02} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix};\ e_1 = e_{12} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix};\ e_2 = e_{22} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and define new 3×2 elementary matrices

$$l_0 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix};\ l_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix};\ l_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Let $\bigotimes_{i=n}^{1} e_{\alpha_i} = e_{\alpha_n} \otimes \cdots \otimes e_{\alpha_1}$, and $\bigotimes_{i=n}^{1} l_0 = l_0 \otimes \cdots \otimes l_0$ (n times), etc.

Lemma 2: $l_i = l_2\, e_i$, for any $i \in \{0, 1, 2\}$.

Proof: By ordinary modulo-2 matrix multiplication. □

Lemma 3:

$$\mathbf{T}(f) = \left[\bigotimes_{i=n}^{1} e_{\alpha_i} \right] \mathbf{C}^{\alpha}(f) \tag{7}$$

Proof: From Theorem 1. □

Lemma 4:

$$\mathbf{P}(f) = \left[\bigotimes_{i=n}^{1} l_2 \right] \mathbf{T}(f) \tag{8}$$

Proof: By induction. Let $\mathbf{L}_n = \bigotimes_{i=n}^{1} l_2$.

(a) Consider n = 1, then $\mathbf{L}_1\mathbf{T}(f) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_0 \oplus f_1 \end{bmatrix} = \mathbf{P}(f)$.

(b) As the induction hypothesis, assume the result is true for k = n−1. Consider k = n where

$$\mathbf{T}(f(x_n \cdots x_1)) = [f_0 \cdots f_N]^t = \begin{bmatrix} \mathbf{T}(f(0, x_{n-1} \cdots x_1)) \\ \mathbf{T}(f(1, x_{n-1} \cdots x_1)) \end{bmatrix} = \begin{bmatrix} \mathbf{T}(f^0) \\ \mathbf{T}(f^1) \end{bmatrix}$$

From the induction hypothesis, $\mathbf{P}(f^0) = \mathbf{L}_{n-1}\, \mathbf{T}(f^0)$ and $\mathbf{P}(f^1) = \mathbf{L}_{n-1}\, \mathbf{T}(f^1)$. Therefore,

$$\mathbf{L}_n\mathbf{T}(f) = (l_2 \otimes \mathbf{L}_{n-1})\, \mathbf{T}(f)$$

$$= \begin{bmatrix} \mathbf{L}_{n-1} & 0 \\ 0 & \mathbf{L}_{n-1} \\ \mathbf{L}_{n-1} & 0 \end{bmatrix} \mathbf{T}(f) = \begin{bmatrix} \mathbf{L}_{n-1}\, \mathbf{T}(f^0) \\ \mathbf{L}_{n-1}\, \mathbf{T}(f^1) \\ \mathbf{L}_{n-1}\, \mathbf{T}(f^0) \oplus \mathbf{L}_{n-1}\, \mathbf{T}(f^1) \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{P}(f^0) \\ \mathbf{P}(f^1) \\ \mathbf{P}(f^0) \oplus \mathbf{P}(f^1) \end{bmatrix} = \mathbf{P}(f)$$

by Property P8 of Table 3–1. □

Thus the transform from $\mathbf{C}^{\alpha}(f)$ to $\mathbf{P}(f)$ is as follows:

Theorem 3: Let $\mathbf{L}_{\alpha} = \left[\bigotimes_{i=n}^{1} l_{\alpha_i} \right]$, then

$$\mathbf{P}(f) = \mathbf{L}_{\alpha}\, \mathbf{C}^{\alpha}(f) \tag{9}$$

Proof: From Lemmas 2 to 4 and the associativity of modulo-2 matrix multiplications,

$$\mathbf{P}(f) = \left[\bigotimes_{i=n}^{1} l_2 \right] \mathbf{T}(f)$$

$$= \left[\bigotimes_{i=n}^{1} l_2 \right] \left[\bigotimes_{i=n}^{1} e_{\alpha_i} \right] \mathbf{C}^{\alpha}(f) = \left[\bigotimes_{i=n}^{1} \left(l_2\, e_{\alpha_i} \right) \right] \mathbf{C}^{\alpha}(f)$$

$$= \left[\bigotimes_{i=n}^{1} l_{\alpha_i} \right] \mathbf{C}^{\alpha}(f)$$

□

Using the Kronecker product structure, a fast recursive implementation of the above transform is formulated in Algorithm 2.

Theorem 4: The fast $\mathbf{L}_{\alpha}$ transform takes $3^n - 2^n$ modulo-2 additions.

Proof: The number of XOR operations required in Algorithm 2 satisfies the recurrence relation

$$T_n = 3^{n-1} + 2 \cdot T_{n-1};\ T_1 = 1 \tag{10.1}$$

Let $T_n = 2^n \cdot Y_n$, then

Algorithm 2: Fast Transforms from $C^{\alpha}(f)$ to $P(f)$

```
type
  BoolVec : array[0 · · · 2^n−1] of boolean;
  ntrits : array[1 · · · n] of 0 · · · 2;
  PSpec : array[0 · · · 3^n−1] of boolean;
procedure Ltrans(n:integer; α:ntrits; C^α:BoolVec; var P:PSpec);
  procedure Ltrans-aux(n,lbc,lbp:integer);
  begin
    if n = 1 then
      a := C^α[lbc]; b := C^α[lbc + 1];
      case α_1 of
        0:    P[lbp] := a; P[lbp + 1] := a ⊕ b; P[lbp + 2] := b;
        1:    P[lbp] := a ⊕ b; P[lbp + 1] := a; P[lbp + 2] := b;
        2:    P[lbp] := a; P[lbp + 1] := b; P[lbp + 2] := a ⊕ b;
      end
    else
      case α_n of
        0:    Ltrans-aux(n−1, lbc, lbp);
              Ltrans-aux(n−1, lbc + 2^(n−1), lbp + 2·3^(n−1));
              for k := 0 to 3^(n−1)−1 do
                P[lbp + 3^(n−1) + k] := P[lbp + k] ⊕ P[lbp + 2·3^(n−1) + k];
        1:    Ltrans-aux(n−1, lbc, lbp + 3^(n−1));
              Ltrans-aux(n−1, lbc + 2^(n−1), lbp + 2·3^(n−1));
              for k := 0 to 3^(n−1)−1 do
                P[lbp + k] := P[lbp + 3 sup n-1 + k] ⊕ P[lbp + 2·3^(n−1) + k];
        2:    Ltrans-aux(n−1, lbc, lbp + 3^(n−1));
              Ltrans-aux(n−1, lbc + 2^(n−1), lbp + 3^(n−1));
              for k := 0 to 3^(n−1)−1 do
                P[lbp + 2·3^(n−1) + k] := P[lbp + k] ⊕ P[lbp + 3^(n−1) + k];
      end
  end Ltrans-aux;
begin
  Ltrans-aux(n, 0, 0);
end Ltrans;
```

$$Y_n = Y_{n-1} + \frac{3^{n-1}}{2^n};\ Y_1 = \frac{1}{2} \tag{10.2}$$

Evaluating Y_n as the sum of a geometric series and substituting back for T_n, we get

$$T_n = 3^n - 2^n \tag{10.3}$$

□

Since every FBE coefficient is a subfunction parity, $P(f)$ embeds $C^{\alpha}(f)$. Thus there are $3^n - 2^n$ new and distinct entries in $P(f)$ to be computed in the transform. Because Algorithm 2 requires an average of only one XOR operation per new entry, it is optimal. In contrast, an ordinary matrix multiplication for the transform requires $6^n - 3^n$ XORs. Even if an algorithm which considers only non-zero entries of the transform matrix could be developed, a total of $4^n - 3^n$ XORs are required.

5.3.2. The Inverse Transforms

The inverse problem of computing $C^{\alpha}(f)$ from $P(f)$ is also of interest. Mathematically, we seek the inverse transform matrix K_{α} such that $C^{\alpha} = K_{\alpha}P(f)$. From Theorem 3, $K_{\alpha}P(f) = K_{\alpha}(L_{\alpha}C^{\alpha}(f)) = (K_{\alpha}L_{\alpha})C^{\alpha}(f)$. Thus $K_{\alpha}P(f) = C^{\alpha}(f)$ if and only if $K_{\alpha}L_{\alpha} = I_n$, the $2^n \times 2^n$ identity matrix.

Since L_{α} is non-square, its left inverse is not unique. Now $P(f)$ embeds $C^{\alpha}(f)$ and the transform needs only to be an extraction process. In other words, K_{α} has a single 1 in every row.

Define the following 2×3 elementary matrices:

$$k_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix};\ k_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix};\ k_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Lemma 5: $k_i l_i = I_1\ \forall\ i \in \{0, 1, 2\}$.

Proof: By ordinary modulo-2 matrix multiplication. □

Theorem 5:

$$C^{\alpha}(f) = \left[\bigotimes_{i=n}^{1} k_{\alpha_i} \right] P(f) \tag{11}$$

Proof: From Theorem 3, Lemma 2–1, and Lemma 5:

$$\left[\bigotimes_{i=n}^{1} k_{\alpha_i} \right] P(f) = \left[\bigotimes_{i=n}^{1} k_{\alpha_i} \right] \left[\bigotimes_{i=n}^{1} l_{\alpha_i} \right] C^{\alpha}(f) = \left[\bigotimes_{i=n}^{1} k_{\alpha_i} l_{\alpha_i} \right] C^{\alpha}(f)$$

$$= I_n C^{\alpha}(f) = C^{\alpha}(f)$$

□

The recursive extraction of $C^{\alpha}(f)$ from $P(f)$ is given in Algorithm 3. A measure of its time complexity is its number of recursive procedure calls, which satisfies $T_n = 2 \cdot T_{n-1}$; $T_1 = 1$. Thus $T_n = 2^{n-1}$.

An alternative for constructing the inverse transform as described is to use another set of elementary matrices:

$$\hat{k}_0 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} ; \hat{k}_1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} ; \hat{k}_2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

These alternative transform matrices have 2^n ones per row and modulo-2 additions are required to carry out the transforms. Consequently, this approach is less efficient.

A third method of computing the transform is algebraic using the ⊛ *(c-star)* operator (see Theorem 4–2 in Chapter 4). This approach is also less efficient in general, but is the preferable method for extracting random FBE coefficients individually from the parity spectrum.

Algorithm 3: Fast Transforms from $P(f)$ to $C^{\alpha}(f)$

```
procedure Ktrans(n:integer; c:matrits; P:PSpec; var C^α:BoolVec);
  procedure Ktrans-aux(n,lbp,lbc:integer);
  begin
    if n = 1 then
      a := P[lbp]; b := P[lbp+1]; c := P[lbp+2];
      case α_1 of
        0:    C^α[lbc] := a; C^α[lbc+1] := c;
        1:    C^α[lbc] := b; C^α[lbc+1] := c;
        2:    C^α[lbc] := a; C^α[lbc+1] := b;
      end
    else
      case α_n of
        0:    Ktrans-aux(n−1, lbc, lbp);
              Ktrans-aux(n−1, lbc + 2^(n−1), lbp + 2·3^(n−1));
        1:    Ktrans-aux(n−1, lbc, lbp + 3^(n−1));
              Ktrans-aux(n−1, lbc + 2^(n−1), lbp + 2·3^(n−1));
        2:    Ktrans-aux(n−1, lbc, lbp);
              Ktrans-aux(n−1, lbc + 2^(n−1), lbp + 3^(n−1));
      end
  end Ktrans-aux;
begin
  Ktrans-aux(n, 0, 0);
end Ktrans;
```

5.4. Minimization of FPEs and FBEs

5.4.1. The Minimization Approaches

Recall from Section 2.4.2 that the *weight* of an FPE or FBE is its number of non-zero terms, and a minimization algorithm for FPEs or FBEs select from among the 2^n FPEs or 3^n FBEs an expansion with the minimal weight. Let $w^v = w^v(f)$ and

$\omega^\alpha = \omega^\alpha(f)$ be the weights of the v–FPE and α–FBE of $f(x_n \cdots x_1)$. The vector of the 2^n FPE weights of f is denoted by

$$\mathbf{W}(f) = [w^0 \cdots w^N]^t \tag{12.1}$$

and the vector of the 3^n FBE weights of f is denoted by

$$\Omega(f) = [\omega^0 \cdots \omega^M]^t \tag{12.2}$$

where $N = 2^n-1$ and $M = 3^n-1$. Note that $\omega^\alpha = w^v$ when $\alpha \Rightarrow v$, and $\mathbf{W}(f)$ is a subset of $\Omega(f)$.

Let $f(x_n \cdots x_1)$ be represented by its truth vector $\mathbf{T}(f)$. Using the transforms developed in the previous sections, two approaches to the minimization of FPEs or FBEs are possible. In the first approach, the FPEs or FBEs are generated explicitly and the weights are computed directly from the expansions. To minimize the required amount of computation, the expansions are generated in a Gray code sequence of the polarities for FPEs or of the bases for FBEs. In the second approach, the expansions are not explicitly generated. Instead, Boolean matrix transforms also described in terms of Kronecker products are used to obtain the weight vectors from the parity spectrum by ordinary matrix multiplications (as opposed to modulo-2 matrix multiplications). These minimization approaches are described in this section. For the purpose of comparison, we first describe below a brute-force approach that explicitly generates each of the FPEs or FBEs from the truth vector.

5.4.2. Brute-Force Approach

Consider the problem of generating all 2^n FPEs of $f(x_n \cdots x_1)$ from $\mathbf{T}(f)$. A brute-force approach is to apply (4.1) in Corollary 2(a) and compute the v–FPE as

$$\mathbf{A}^v(f) = \left[\bigotimes_{i=n}^{1} e_{2v_i} \right] \mathbf{T}(f) \tag{13}$$

Let α and M be decimal equivalents of ternary n-tuples such that $\alpha \Rightarrow v$ and $M \sim (2 \cdots 2)$. Since $\mathbf{A}^v(f) = \mathbf{C}^\alpha(f)$ and $\mathbf{T}(f) = \mathbf{C}^M(f)$, it follows that $d(\alpha, M) = n$.

From Theorem 2, an application of (13) in computing an FPE requires $n \cdot 2^{n-1}$ modulo-2 additions. Therefore, the number of modulo-2 additions required for generating all the 2^n FPEs from $\mathbf{T}(f)$ is

$$2^n \cdot (n \cdot 2^{n-1}) = \frac{n \cdot 4^n}{2} \tag{14}$$

Similarly, the 3^n FBEs can be generated by applying (4.2) in Corollary 2(b) to compute the α–FBE as

$$\mathbf{C}^{\alpha}(f) = \left[\bigotimes_{i=n}^{1} e_{2^{\alpha_i}} \right] \mathbf{T}(f) \tag{15}$$

Let $m = d(\alpha, M)$, then $0 \le m \le n$ and the number of modulo-2 additions required in computing an FBE using (15) is, from Theorem 2, equal to $m \cdot 2^{n-1}$. Now for all the 3^n FBEs, the number of bases α such that $d(\alpha, M) = m$ is

$$m \cdot \binom{n}{m} \cdot 2^m = n \cdot \binom{n-1}{m-1} \cdot 2^m \tag{16}$$

Summing over all possible m's we obtain the number of modulo-2 additions required for generating all the 3^n FBEs from $\mathbf{T}(f)$ as

$$\left[\sum_{m=1}^{n} n \cdot \binom{n-1}{m-1} \cdot 2^m \right] \cdot 2^{n-1} = 2 \cdot n \cdot \left[\sum_{m=0}^{n-1} \binom{n-1}{m} \cdot 2^m \right] \cdot 2^{n-1} = 2 \cdot n \cdot 3^{n-1} \cdot 2^{n-1} = \frac{n \cdot 6^n}{3} \tag{17}$$

The brute-force approach to the generation of FPEs is first mentioned briefly, also for the purpose of comparison, by Davio[Dav71]. Some later authors[Swa72, Sal79] erroneously suggest that instead of carrying out 2^n different transforms for the 2^n FPEs, a single transform matrix should be used. The different FPEs must then be obtained by applying this single transform to vectors obtained by permuting entries in $\mathbf{T}(f)$. According to Corollary 3(a), each of the 2^n fast transforms requires the same number of operations, and the single-transform approach does not speed up the exhaustive generation of the FPEs. In fact, it is less efficient because extra computation is required to permute the entries in $\mathbf{T}(f)$.

By extending the brute-force approach for generating FPEs to FBEs and calculating the time complexities of both schemes, a basis is provided for comparison with the Gray code approach, which may be seen as a simple refinement of the brute-force approach.

5.4.3. Gray Code Approach

Recall from Chapter 4, Section 4.5 that two FPEs with adjacent polarities or two FBEs with adjacent bases share half of their coefficients in the P-cube. This observation is further expanded by Theorem 2, which shows that the number of modulo-2 additions required in computing one FPE/FBE from another is directly proportional to the distance between the two polarities/bases.

In the brute force approach for generating FPEs of $f(x_n \cdots x_1)$, each FPE $\mathbf{A}^{\mathbf{v}}(f)$ is generated from the truth vector $\mathbf{T}(f) = \mathbf{C}^{\mathbf{M}}(f)$ where the distance d(M, v) = n is at the farthest. Since it is most efficient to compute $\mathbf{A}^{\mathbf{v}}(f)$ from another FPE with an adjacent polarity, an improved algorithm is obtained by computing all the FPEs in an n-bit *binary Gray code* sequence of their polarities. The first FPE, say the 0–FPE, is computed from $\mathbf{T}(f)$ but each subsequent FPE is generated from the previous FPE using the fast transform for

$$\mathbf{A}^{\mathbf{v}}(f) = \left[\bigotimes_{i=n}^{1} e_{u_i v_i} \right] \mathbf{A}^{\mathbf{u}}(f) \tag{18}$$

Since a Gray code sequence is used, the distance d(u, v) = 1 is at its minimum in (18). Consequently, the total number of modulo-2 additions required is reduced from (14) to

$$n \cdot 2^{n-1} + (2^n - 1) \cdot 2^{n-1} = 4^n + (n-1) \cdot \frac{2^n}{2} \tag{19}$$

For large n, the speed-up factor is

$$\frac{n \cdot 4^n}{4^n + (n-1) \cdot 2^n} \approx n \tag{20}$$

Incidentally, our analysis here provides the theoretical interpretation for an empirical tabulation([Zha84], Table 1) of n versus the speed-up factor which gradually approaches n.

The Gray code approach to the minimization of FBEs is first published by Davio et al.([7], Algorithm 6.2). Unfortunately, later authors[WuC82, Bes83, Zha84] seem unaware of the results. A cumbersome map-folding procedure is used in [WuC82], applicable only to the minimization of functions with $n \leq 4$ variables. The map-folding method is improved in [Bes83] using signal flow diagrams, but the result is, in fact, identical to the Gray code approach. In [Zha84], the signal flow diagrams are called flow graphs and the Gray code minimization procedure is re-stated and then extended to multiple output networks and multi-valued functions. The extensions are nevertheless "brute-force" and do not seem practical because of the computational complexities involved in the algorithms. Further research is required to obtain practical solutions to these extended minimization problems.

Although the binary Gray code approach to the minimization of FPEs have been presented elsewhere, our unified description in terms of the generalized fast transform (Algorithm 1) simplifies its analysis and its extension to the minimization of FBEs. In this extension, the 3^n FBEs are generated in an n-trit *ternary Gray code* sequence of their bases. Using Theorem 1,

$$C^{\beta}(f) = \left[\bigotimes_{i=n}^{1} e_{\alpha_i\beta_i} \right] C^{\alpha}(f) \tag{21}$$

Since the distance $d(\alpha, \beta) = 1$ is minimized, the amount of computation is reduced from the brute-force approach.

The ternary Gray code sequence of the bases can also be generated using Algorithm 2–1, which starts from basis zero (i.e. the 0–FBE). Since $T(f) = C^M(f)$ is an FBE itself, Algorithm 2–1 may be modified to start from the basis $M = (2 \cdots 2)$. In this case, the number of modulo-2 additions required for generating the 3^n FBEs is

$$\left(3^{n}-1\right)\cdot 2^{n-1} = \frac{6^{n}-2^{n}}{2} \tag{22}$$

Compared with (17) for the brute-force approach, the speed-up factor for large n is

$$\frac{2\cdot n\cdot 6^{n}}{3\cdot(6^{n}-2^{n})} \approx \frac{2\cdot n}{3} \tag{23}$$

So far we have not mentioned the number of arithmetic operations required in computing the weights for FPE or FBE minimization. For FPEs, this number is

$$2^{n}\cdot(2^{n}-1) = 4^{n}-2^{n} \tag{24}$$

and for FBEs, this number is

$$3^{n}\cdot(2^{n-1}) = 6^{n}-3^{n} \tag{25}$$

These numbers are the same for both the brute-force and Gray-code approaches, which differ only in their way of generating the expansions. To simplify the comparison of these approaches, we assign a unit computational cost to both a modulo-2 addition and an arithmetic addition.[1] Also, we note that the cost of generating the next element in a Gray code sequence using Algorithm 2–1 is of order O(n), which is negligible compared with generating the expansions and their weights. These assumptions allow the time complexities of these approaches to be compared in Table 1.

5.4.4. Parity Spectrum Approach

Instead of generating the FPEs or FBEs explicitly, the *parity spectrum* approach to the minimization of FPEs and FBEs computes the weights of the expansions simultaneously from the parity spectrum. This is possible because, from Chapter 4, the coefficients of the FPEs and FBEs can be identified with subfunction parities in the parity spectrum. This approach eliminates the computation required for

[1]Strictly speaking, the computational cost of an n-bit arithmetic addition is of order n times that of a modulo-2 addition.

Time Complexity	Approach	
	Brute-force	Gray Code
FPE	$O(n \cdot 4^n)$	$O(4^n)$
FBE	$O(n \cdot 6^n)$	$O(6^n)$

Table 1: Time Complexities for Brute-force and Gray Code Minimization

generating the e: ; nsions. Furthermore, since coefficients from different FPEs or FBEs can be *equivalent* (i.e. from the same subfunction parity), additional savings in time is obtained by computing the weights simultaneously.

The transforms frforms from the parity spectrum $\mathbf{P}(f)$ to the vector $\mathbf{W}(f)$ of weights of FPEs and the vector $\mathbf{\Omega}(f)$ of weights of FBEs can also be described in terms of Kronecker products of elementary Boolean matrices, except that the transform process now involves real rather than modulo-2 matrix multiplications. As is used for scalars throughout this dissertation, the dot symbol $\cdot$ is also used to denote real multiplications of matrices and vectors.

Let $Z_n = [1 \cdots 1]$ denotes the row vector of 2^n 1's, then the weight of the α–FBE can be computed as

$$\omega^{\alpha}(f) = Z_n \cdot \mathbf{C}^{\alpha}(f) \tag{26}$$

Let $\hat{I}_0 = [1\ 0\ 1]$; $\hat{I}_1 = [0\ 1\ 1]$; $\hat{I}_2 = [1\ 1\ 0]$. It may be seen that $I_0 = \begin{bmatrix} \hat{I}_2 \\ \hat{I}_1 \end{bmatrix}^t$; $I_1 = \begin{bmatrix} \hat{I}_2 \\ \hat{I}_0 \end{bmatrix}^t$; and $I_2 = \begin{bmatrix} \hat{I}_0 \\ \hat{I}_1 \end{bmatrix}^t$. Using real matrix multiplications, we easily obtain the

following lemmas:

Lemma 6: $\hat{l}_i = [1\ 1]\cdot k_i$, where $i \in \{0, 1, 2\}$. □

In other words, $\hat{l}_i$ can be formed by collapsing the rows in k_i using modulo-2 addition.

Lemma 7:

$$\omega^{\alpha}(f) = \left[\hat{l}_{\alpha_n} \otimes \cdots \otimes \hat{l}_{\alpha_1}\right]\cdot \mathbf{P}(f) \tag{27}$$

Proof: From Lemma 7 and the real counterpart of the Kronecker product result in Lemma 2-1(b), Section 2.5 of Chapter 2:

$$\hat{\mathbf{L}}_{\alpha} = \left[\hat{l}_{\alpha_n} \otimes \cdots \otimes \hat{l}_{\alpha_1}\right] = \left[[1\ 1]\cdot k_{\alpha_n}\right] \otimes \cdots \otimes \left[[1\ 1]\cdot k_{\alpha_1}\right]$$

$$= \mathbf{Z}_n \cdot \left[k_{\alpha_n} \otimes \cdots \otimes k_{\alpha_1}\right]$$

$$= \mathbf{Z}_n \cdot \mathbf{K}_{\alpha}$$

Therefore,

$$\hat{\mathbf{L}}_{\alpha}\cdot \mathbf{P}(f) = \mathbf{Z}_n \cdot \mathbf{K}_{\alpha} \cdot \mathbf{P}(f) = \mathbf{Z}_n \cdot \left[\mathbf{K}_{\alpha} \cdot \mathbf{P}(f)\right]$$

Since $\mathbf{K}_{\alpha}$ has a single 1 per row, the real matrix multiplication between $\mathbf{K}_{\alpha}$ and $\mathbf{P}(f)$ can be replaced by a modulo-2 matrix multiplication to obtain

$$\hat{\mathbf{L}}_{\alpha}\cdot \mathbf{P}(f) = \mathbf{Z}_n \cdot \left[\mathbf{K}_{\alpha}\mathbf{P}(f)\right]$$

From Theorem 4, $\mathbf{K}_{\alpha}\mathbf{P}(f) = \mathbf{C}^{\alpha}(f)$, and result follows from (26).

Lemma 7 can be applied to simultaneously compute a subset of the 3^n FBE weights. For certain subsets, one can make use of the Kronecker product structure of the transforms to reduce the computational effort required. Intuitively, the savings in computation is achieved by exploiting the sharing of coefficients among FBEs. Thus a partial weight common to several FBEs in the subset need only be computed once.

For minimization of FPEs and FBEs, we are interested in transforms that compute all of the FPE weights $\mathbf{W}(f)$ and all of the FBE weights $\Omega(f)$. Let $\mathbf{L}_n^t = \bigotimes_{i=n}^{1} \begin{bmatrix} \hat{l}_0 \\ \hat{l}_1 \end{bmatrix}$ and $\hat{\mathbf{L}}_n = \bigotimes_{i=n}^{1} \begin{bmatrix} \hat{l}_0 \\ \hat{l}_1 \\ \hat{l}_2 \end{bmatrix}$, so that $\mathbf{L}_1^t = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$, $\hat{\mathbf{L}}_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$, and $\mathbf{L}_n^t$ is the transpose of $\mathbf{L}_n$. ($\mathbf{L}_n$ is the transform for computing $\mathbf{P}(f)$ from $\mathbf{T}(f)$ in Lemma 4).

Algorithm 4: Fast Transform from $\mathbf{P}(f)$ to $\mathbf{W}(f)$

```
type
  WVec : array [0 · · · 2^n–1] of integer;

procedure L^t-trans(n:integer; P:PSpec; var W:WVec);
  procedure L^t-trans-aux(n,lbp,lpw:integer; var W:WVec);
  var
    tempW : array [0 · · · 2^(n–1)–1] of integer;
  begin
    if n = 1 then
      a := P[lbp]; b := P[lbp + 1]; c := P[lbp + 2];
      W[lbw] := a+c; W[lbw+1] := b+c;
    else
      L^t-trans-aux(n–1, lbp, lbw, W);
      L^t-trans-aux(n–1, lbp + 3^(n–1), lbw + 2^(n–1), W);
      L^t-trans-aux(n–1, lbp + 2·3^(n–1), 0, tempW);
      for k := 0 to 2^(n–1)–1 do
        W[lbw + k] := W[lbw + k] + tempW[k];
        W[lbw + 2^(n–1) + k] := W[lbw + 2^(n–1) + k] + tempW[k];
  end L^t-trans-aux;
begin
  L^t-trans-aux(n, 0, 0, W);
end L^t-trans;
```

Theorem 6:

(a) $\mathbf{W}(f) = \mathbf{L}_n^t \cdot \mathbf{P}(f)$

(b) $\Omega(f) = \hat{\mathbf{L}}_n \cdot \mathbf{P}(f)$

Proof: The u-th row of $\mathbf{L}_n^t$, $0 \le u \le N$, is the vector $\hat{I}_{u_n} \otimes \cdots \otimes \hat{I}_{u_1}$. Also, the α-th row of $\hat{\mathbf{L}}_n$, $0 \le \alpha \le M$, is the vector $\hat{I}_{\alpha_n} \otimes \cdots \otimes \hat{I}_{\alpha_1}$. Thus the results follow from Lemma 7. □

By making use of the Kronecker product structure, Algorithms 4 and 5 implement the fast $\mathbf{L}_n^t$ and $\hat{I}_n$ transforms. Their time complexities are measured by the numbers of real additions required.

Theorem 7: The fast $\mathbf{L}_n^t$ and $\hat{\mathbf{L}}_n$ transforms require $2 \cdot (3^n - 2^n)$ and $n \cdot 3^n$ arithmetic additions, respectively.

Proof: For Algorithm 4, the number of arithmetic additions satisfies the recurrence relation

$$T_n = 2^n + 3 \cdot T_{n-1} \; ; \; T_1 = 2 \tag{28.1}$$

Let $T_n = Y_n \cdot 3^n$, then

$$Y_n = \frac{2^n}{3^n} + Y_{n-1}; \; Y_1 = \frac{2}{3} \tag{28.2}$$

Evaluating Y_n as the sum of a geometric series and substituting back, we get

$$T_n = 2 \cdot (3^n - 2^n) \tag{28.3}$$

For Algorithm 5, the number of arithmetic additions satisfies

$$T_n = 3^n + 3 \cdot T_{n-1}; \; T_1 = 3 \tag{29.1}$$

Again let $T_n = Y_n \cdot 3^n$ and solve

$$Y_n = 1 + Y_{n-1}; \; Y_1 = 1 \tag{29.2}$$

to get

Algorithm 5: Fast Transform from $\mathbf{P}(f)$ to $\Omega(f)$

```
type
  ΩVec : array [0 · · · 3^n−1] of integer;

procedure L̂-trans(n:integer; P:PSpec; var Ω:ΩVec);
  procedure L̂-trans-aux(n,lbp,lpω:integer);
  begin
    if n = 1 then
      a := P[lbp]; b := P[lbp + 1]; c := P[lbp + 2];
      Ω[lbω] := a+c; Ω[lbω+1] := b+c; Ω[lbω+2] := a+b;
    else
      L̂-trans-aux(n−1, lbp, lbω);
      L̂-trans-aux(n−1, lbp + 3^(n−1), lbω + 3^(n−1));
      L̂-trans-aux(n−1, lbp + 2·3^(n−1), lbω + 2·3^(n−1));
      for k := 0 to 3^(n−1)−1 do
        a := Ω[lbω + k];
        b := Ω[lbω + 3^(n−1) + k];
        c := Ω[lbω + 2·3^(n−1) + k];
        Ω[lbω + k] := a+c;
        Ω[lbω + 3^(n−1) + k] := b+c;
        Ω[lbω + 2·3^(n−1) + k] := a+b;
  end L̂-trans-aux;
begin
  L̂-trans-aux(n, 0, 0);
end L̂-trans;
```

$$T_n = n \cdot 3^n \tag{29.3}$$

□

The parity spectrum approach to the minimization of FPEs for a Boolean function $f(x_n \cdots x_1)$ defined by its truth vector representation $\mathbf{T}(f)$ can be carried out as follows. The $\mathbf{L}_n$ transform is first used to derive the parity spectrum $\mathbf{P}(f)$. Applying

the $\mathbf{L}_n^t$ transform to $\mathbf{P}(f)$ yields the weights of all FPEs (i.e. $\mathbf{W}(f)$). The optimal polarities are then determined from the minimal weight in $\mathbf{W}(f)$, and the appropriate $\mathbf{K}$ transforms are used to extract the optimal FPEs. Similarly, the optimal FBEs are found by using the $\hat{\mathbf{L}}_n$ transform to obtain $\Omega(f)$ from which the optimal bases are determined.

The minimization of FPEs and FBEs expansions using the $\mathbf{L}_n^t$ and the $\hat{\mathbf{L}}_n$ transforms is also considered in [Bio73], based on *Boolean difference*. Our manner of attack using subfunction parities leads to generalized transforms, simplified proofs, and efficient computer implementations.

The time complexities of the minimization algorithms are $O(3^n)$ for FPEs and $O(n \cdot 3^n)$ for FBEs. Compared with the Gray code minimization approach, the parity spectrum approach is faster (see Table 2). This is because FPEs and FBEs are not explicitly generated and the weights are computed simultaneously by exploiting the sharing of coefficients among the expansions. Also, the generation of a Gray code sequence is no longer necessary.

However, the parity spectrum approach requires more space (see Table 3). In the brute-force and Gray code approaches, only one FPE or FBE (of 2^n bits) needs to be stored at a time as the expansions are generated, and the space requirement is $O(2^n)$. For the parity spectrum approach, the requirement increases considerably to $O(3^n)$, since the entire parity spectrum (of 3^n bits) has to be stored.

5.5. Summary

In this chapter, generalized Boolean matrix transforms among the FPEs, FBEs and the parity spectrum of a Boolean function are described in a unified manner. Based on the Kronecker product structure of these transforms, fast recursive algorithms are developed.

Using the transforms among FPEs and FBEs, the Gray code approach to the minimization of FPEs is reviewed and then extended to FBEs. Combining the use of transforms between the parity spectrum and the FPEs and FBEs and transforms that

Time Complexity	Approach	
	Parity Spectrum	Gray Code
FPE	$O(3^n)$	$O(4^n)$
FBE	$O(n \cdot 3^n)$	$O(6^n)$

Table 2: Time Complexities for Gray Code and Parity Spectrum Minimization

Space Complexity	Approach	
	Parity Spectrum	Gray Code
FPE	$O(3^n)$	$O(2^n)$
FBE	$O(3^n)$	$O(2^n)$

Table 3: Space Complexities for Gray Code and Parity Spectrum Minimization

simultaneously compute from the parity spectrum the weights of all FPEs or all FBEs, the parity spectrum approach to minimization is also described.

Compared with the Gray code approach, the parity spectrum approach is more efficient but requires more storage space. It may be possible to develop a hybrid approach that is adaptable to either space or time limitations. Further investigation is required.

Certain heuristic minimization algorithms for general modulo-2 expressions (e.g. [Rob82]) start from a minimal FPE and then search for mixed-polarity reductions.

These algorithms may be more effective if an optimal FBE is used, since FBEs are more general than FPEs. New heuristic reduction rules need to be devised for such a minimization approach.

Finally, some of the transforms developed in this paper can also be used in a generalized fault detection technique called *constrained parity testing*, which detects faults in a combinational circuit by verifying subfunction parities of its output function. Results for this new testing technique are developed in the next chapter.

CHAPTER 6

CONSTRAINED PARITY TESTING

6.1. Introduction

In this chapter, we develop the *constrained parity testing* technique for detecting *multiple stuck-at faults* in *single-output combinational networks*. Logic fault testing involves the application of a sequence of input stimuli to the *network under test (NUT)* and comparing the output responses from the NUT to their expected values. Any discrepancy indicates a fault. In recent years, the traditional test vector testing method has given way to *compaction techniques* more suitable for *built-in self-test (BIST)* schemes[McC85]. These techniques reduce the volume of test data by compacting the test responses from the NUT into a bit vector called a *signature*. Testing is performed by verifying this signature instead of the individual responses.

A number of compaction techniques employ signatures composed of *parity bits*. Tzidon et al.[Tzi78] and Carter[Car82] use as a signature the parity on the number of *ones* of all responses from the NUT. Akers'[Ake88] parity bit signature contains the network parity plus subparities formed from constraining one of the network inputs at logic *zero*. More recently, Damarla & Karpovsky[Dam89] derive signatures from coefficients in the *complement-free Reed-Muller canonical expansion* of the implemented function. It follows from Chapter 4 that these coefficients correspond to subparities resulting from constraining a subset of the network inputs at logic *zero*. This chapter presents a generalization of the above parity-based techniques to constrained parity testing, where each of constrained inputs can be fixed at either *zero* or *one*. Since all inputs can be constrained simultaneously, this new technique is also a generalization of the traditional test vector method. The proposed constrained parity

testing technique provides more flexibility and versatility in the choice of tests. The signatures derived are usually short and provide high stuck-at fault coverage. Moreover, the tests in a signature can be chosen to require only a small subset of the input combinations, thus reducing the test time required in the exhaustive testing of large networks.

Section 6.2 introduces the constrained parity testing scheme. Implementation independent testability conditions for single- and multiple-input stuck-at faults are established in Sections 6.3 and 6.4, respectively. In Section 6.5, a *spanning parity signature (SPS)* is introduced to detect *vacuous* faults, which includes all the input stuck-at faults and over 99 percent of all stuck-at faults. The SPS is considered in Section 6.6 for testing all stuck-at faults in networks with small number of fanout lines, and a method of deriving tests for non-vacuous faults is described. For networks with large fanouts, a hybrid scheme by combining with *syndrome testing* is proposed in Section 6.7 to eliminate or reduce the need for expensive fault simulation.

6.2. The Testing Scheme

In the proposed constrained parity testing scheme, a subset of the subparities in $\mathbf{P}(f)$ is selected to test a combinational network implementing the function f. This subset of $\mathbf{P}(f)$ is called a *constrained parity signature* of the NUT. During test time, input combinations are applied to the NUT and the signature is collected and verified at the network output using flip-flops and the necessary control and clocking circuitry. Any distortion in the collected signature indicates a fault in the NUT.

The general compaction testing scheme is shown in Figure 1. In Figure 2, an example of constrained parity testing is shown for a NUT implementing a three-variable function $f(x_3, x_2, x_1)$ with $\{p_{021}, p_{202}\}$ chosen as the signature. Note that all input patterns can be applied, although the subset {000, 001, 011, 100, 101} is adequate for the verification of p_{021} and p_{202}.

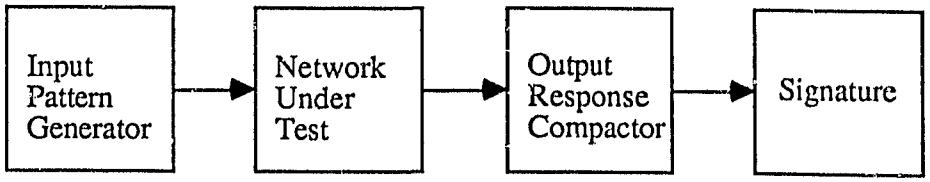


***Figure* 1:** Compaction Testing Scheme

6.3. Single-input Stuck-at Faults

A *stuck-at fault* occurs when a line in the (NUT) is permanently stuck at 0 or 1. Let the NUT implement $f(x_n \cdots x_1)$. Denote the two *single-input stuck-at faults* involving an input line, say x_n, by x_n–s–a–0 and x_n–s–a–1 respectively. Theorem 1 derives the conditions for the faults to be detected by verifying a subparity p_α at the network output.

***Theorem* 1:** Let $\alpha \sim (\alpha_n \cdots \alpha_1)$, $\alpha' \sim (\alpha_{n-1} \cdots \alpha_1)$, and $u_n \in \{0, 1\}$. Then x_n–s–a–u_n is p_α–testable iff $p_{2\alpha'} = 1$ and $\alpha_n \neq u_n$. i.e. If $p_{2\alpha'} = 1$, then

(a) $p_{2\alpha'}$ is a test for both x_n–s–a–0 and x_n–s–a–1;

(b) $p_{1\alpha'}$ is a test for x_n–s–a–0; and

(c) $p_{0\alpha'}$ is a test for x_n–s–a–1.

***Proof*:** Let $\hat{p}_\alpha$ be the faulty subparity due to x_n–s–a–u_n. Then $\hat{p}_{0\alpha'} = \hat{p}_{1\alpha'}$ so that $\hat{p}_{2\alpha'} = \hat{p}_{0\alpha'} \oplus \hat{p}_{1\alpha'} = 0$. Therefore, $p_{2\alpha'} = 1$ is a test for both x_n–s–a–0 and x_n–s–a–1. Now for x_n–s–a–0, $\hat{p}_{1\alpha'} = p_{0\alpha'}$, and the fault is $p_{1\alpha'}$-testable iff $p_{0\alpha'} \neq p_{1\alpha'}$ (i.e. $p_{2\alpha'} = 1$). Similarly, $p_{0\alpha'}$ is a test for x_n–s–a–1 iff $p_{2\alpha'} = 1$. □

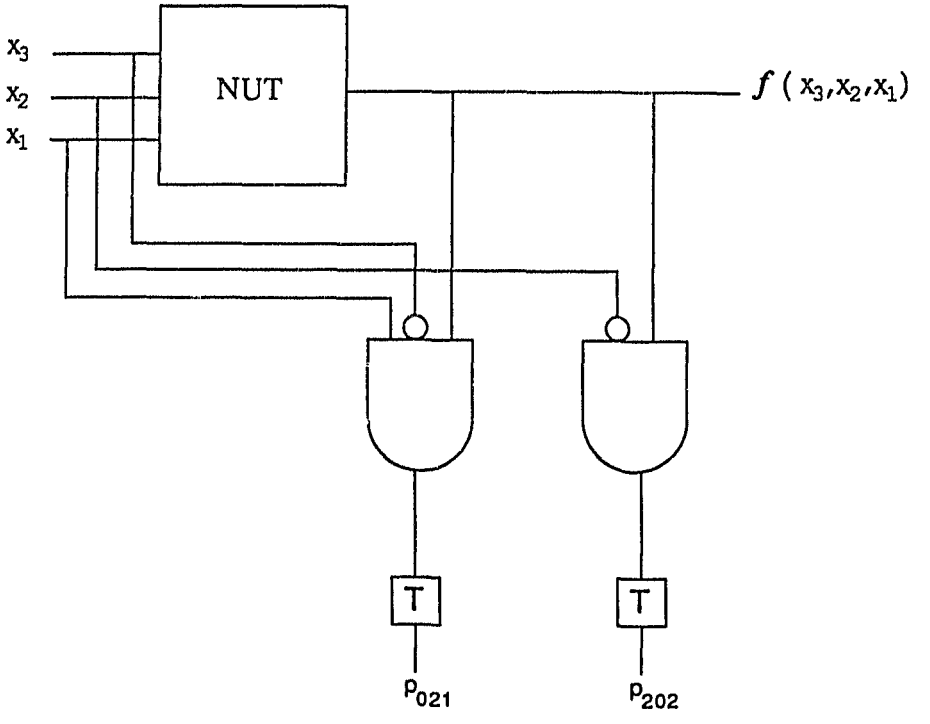


Figure 2: Constrained Parity Testing

When $\alpha \Rightarrow u$ (i.e. $\alpha_i = u_i \in \{0, 1\}\ \forall 1 \leq i \leq n$), $p_\alpha = p_u$ is a test vector and the above theorem states that p_u covers x_n–s–a–$\overline{u}_n$ iff $p_{2u'} = 1$, where $u \sim (u_n \cdots u_1)$ and $u' \sim (u_{n-1} \cdots u_1)$. A corresponding result in [Sel68a,b] defines test vectors for x_n–s–a–0 and x_n–s–a–1 by the *Boolean difference*[Ake59] equations $x_n f_{x_n} = 1$ and $\overline{x}_n f_{x_n} = 1$ respectively, where $f_{x_i} = f(x_n \cdots x_{i+1}, 0, x_{i-1} \cdots x_1) \oplus f(x_n \cdots x_{i+1}, 1, x_{i-1} \cdots x_1)$ is the first order Boolean difference of f with respect to x_i. The Boolean

difference result yields expressions for defining test vectors covering a fault, while Theorem 1 determines the fault coverage of a test vector without requiring the symbolic manipulation of Boolean expressions. The correspondence between these two results stems from the fact that a subparity can be interpreted as the value of a Boolean difference expression evaluated at an assignment of the variables (see Chapter 3).

In [Dam89], single-input stuck-at fault testability conditions are derived for coefficients in the complement-free Reed-Muller expansion (i.e. the 0-FBE) of f. From the results in Chapter 4, these conditions correspond to those for subparities p_α in Theorem 1(a) when $\alpha_i \in \{0, 2\}$ for all i. Thus the above theorem generalizes the earlier result of [Dam89] to coefficients in FBEs of any bases. Furthermore, the additional possibilities of (b) and (c) give greater flexibility in choosing signatures.

6.4. Multiple-input Stuck-at Faults

If several inputs $x_n \cdots x_{k+1}$ are stuck at $u_n \cdots u_{k+1}$, we have a multiple-input stuck-at fault denoted by $x_n \cdots x_{k+1}$–s–a–$u_n \cdots u_{k+1}$. In this case, Theorem 1 can be generalized to the result below:

Theorem 2: Let $\alpha = \beta\gamma$ where $\alpha \sim (\alpha_n \cdots \alpha_1)$, $\beta \sim (\beta_n \cdots \beta_{k+1})$, $\gamma \sim (\gamma_k \cdots \gamma_1)$ and α_i, β_j, $\gamma_k \in \{0, 1, 2\}$; and $\omega_2(\beta) = \sum_{j=n}^{k+1}(\beta_j = 2)$. Then the multiple-input stuck-at fault $x_n \cdots x_{k+1}$–s–a– $u_n \cdots u_{k+1}$ is p_α-testable iff

(a) $\omega_2(\beta) > 0$ and $p_\alpha = 1$; or

(b) $\omega_2(\beta) = 0$ and $p_\alpha \oplus p_{u_n \cdots u_{k+1}\gamma} = 1$.

Proof:

(a) The faulty network implements $\hat{f}(x_n \cdots x_1)$ which is redundant in a variable x_i whenever $k+1 \le i \le n$. From property P9 in Table 3–1 of Chapter 3, $\hat{p}_{\beta\gamma} = \hat{p}_\alpha = 0$ whenever $\omega_2(\beta) > 0$, and the fault is p_α-testable iff $p_\alpha = 1$.

(b) Since $\hat{p}_\alpha = p_{u_n \cdots u_{k+1}\gamma}$ and the fault is p_α-testable iff $p_\alpha \neq \hat{p}_\alpha$. Note that a necessary condition is that $\beta \neq (u_n \cdots u_{k+1})$, otherwise $\hat{p}_{\beta\gamma} = p_{\beta\gamma}$. □

When $\alpha \Rightarrow u$, $p_\alpha = p_u$ represents a test vector. This special case of Theorem 2(b) corresponds to a rather complex Boolean difference formulation which defines vectors covering $x_n \cdots x_{k+1}$–s–a–$u_n \cdots u_{k+1}$ by the equation[KuM75]

$$x_n^{u_n} f_{x_n} \oplus x_{n-1}^{u_{n-1}} f_{x_{n-1}} \oplus x_n^{u_n} x_{n-1}^{u_{n-1}} f_{x_n x_{n-1}} \oplus \cdots \oplus x_n^{u_n} \cdots x_{k+1}^{u_{k+1}} f_{x_n \cdots x_{k+1}} \qquad (4)$$

where $x_i^0 = x_i$, $x_i^1 = \overline{x}_i$, and $f_{x_i x_j} = (f_{x_i})_{x_j}$ etc. are Boolean differences of f with respect to subsets of $\{x_n \cdots x_{k+1}\}$. Our formulation using Theorem 2 is more intuitive and simplifies considerably the notation and proofs.

A special case of Theorem 2(a) is given in [Dam89] for Reed-Muller coefficients in the 0-FBE (i.e. p_α where $\alpha_i \in \{0, 2\}$). From Chapter 4, Theorem 2(a) covers coefficients from FBEs of arbitrary bases.

6.5. Vacuous Faults

A *vacuous fault*[Lui83, Lui86] causes the NUT to become independent of one or more of its input lines. Any multiple-input stuck-at fault is vacuous. In fact, any stuck-at fault involving internal lines and one or more of the input lines or the output line is vacuous. Consider a NUT of p lines including n inputs and one output. There are $3^p - 1$ possible stuck-at faults of which only 3^{p-n-1} faults do not involve the inputs and the output and are *potentially* non-vacuous. Thus a lower bound on the percentage of vacuous stuck-at faults is[Car82] $B(n) = \dfrac{3^p - 3^{p-n-1}}{3^p - 1} > 1 - \dfrac{1}{3^{n+1}}$. For $n > 3$, $B(n) > 99.5\%$.

A subparity $p_{\alpha_n \cdots \alpha_1}$ is said to *span* an input x_i iff $\alpha_i = 2$. For example, $p_{2\alpha'}$ spans x_n. From property P9 in Table 3–1, we have

Theorem 3: If $p_{2\alpha'} = 1$, then $p_{2\alpha'}$ is a test for all vacuous faults that cause the NUT to become independent of input x_n. □

Note the similarity of this theorem to Theorems 1(a) and 2(a).

A *spanning parity signature (SPS)* for a NUT implementing $f(x_n \cdots x_1)$ is a set of non-zero subparities $\{p_\alpha, p_\beta \cdots p_\gamma\}$ such that every x_i, $1 \le i \le n$, is spanned by at least one subparity in the signature. The next theorem follows directly from the previous one:

Theorem 4: A spanning parity signature (SPS) detects all vacuous faults in the NUT. □

An SPS for vacuous faults is a functional attribute — it is independent of the implementation of the NUT. A Boolean function can have many SPSs and a choice should be made to minimize the testing cost in terms of test time and test space (i.e. number of input patterns required and length of the SPS). Unfortunately, these two minimization criteria are often in conflict with each other.

Consider first the problem of minimizing the test time for an SPS. The following result offers an n-bit signature requiring at most 2·n input patterns:

Theorem 5: Let $f(x_n \cdots x_1)$ be irredundant (i.e. f depends on all of $x_n \cdots x_1$), then there exists an SPS of the form

$$S_v = \{\, p_{2u_{n-1}\cdots u_1},\ p_{v_n 2 v_{n-2}\cdots v_1} \cdots p_{w_n \cdots w_2 2} \,\}$$

where $u_i, v_j \cdots w_k \in \{0, 1\}$.

Proof: It is sufficient to show that if f depends on x_n, then $p_{2u'} = 1$ for some $u' \sim (u_{n-1} \cdots u_1)$. Suppose not, then $p_{2u'} = 0 = p_{0u'} \oplus p_{1u'}$ for all u'. This implies $f(0, x_{n-1} \cdots x_1) = f(1, x_{n-1} \cdots x_1)$ and f is independent of x_n, a contradiction. □

Each subparity in S_v above requires 2 input patterns for a total of 2·n patterns. If the subparities share input patterns, then as few as n+1 test patterns are required. For example $S_v = \{p_{2000}, p_{1200}, p_{1120}, p_{1112}\}$ only requires patterns {0000, 1000, 1100, 1110, 1111}.

Corollary 1: The upper bounds on the number of input patterns required by an SPS of Theorem 5 are:

n	upper bound
2, 3, 4	n + 1
5	n + 2
≥ 6	2·n − 4

Proof: Weiss[Wei72] describes bounds on the length of tests for terminal (i.e. inputs and output) stuck-at faults using the concept of *alternating 1-trees* in the T-cube representation of $f(x_n \cdots x_1)$. An alternating 1-tree of k variables ($k \leq n$) consists of k connected edges such that there is at most one edge of the 1-tree along any dimension of the T-cube and the pair of vertices in each edge have opposite function values. For example, if $f(x_5 \cdots x_1) = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 \vee x_1x_2x_3x_5$, then $f_{00000} = 1$, $f_{01000} = f_{00100} = f_{00010} = f_{00001} = 0$ and these vertices in the T-cube representation of f form an alternating 1-tree of 4 variables. It can be seen that each edge in an alternating 1-tree corresponds to a subparity in the SPS of Theorem 5, so that if $f(x_n \cdots x_1)$ has an alternating 1-tree of k variables, a maximum of $2 \cdot n - k + 1$ test patterns are required in the SPS. Weiss[Wei72] shows that for n = 2, 3, 4, an alternating 1-tree of n variables exists for any irredundant n-variable function. For n = 5, an alternating 1-tree of 4 variables exists, and for $n \geq 6$, an alternating 1-tree of 5 variables exists. Hence we arrive at the results above. □

The SPS in Theorem 5 minimizes the test time but not the test space in terms of signature length. Carter[Car82] employs a 1-bit SPS $\{p_{2\ldots2}\}$ for functions with odd parity. It requires the exhaustive application of all 2^n input patterns but is clearly optimal in length. Akers[Ake88] chooses an (n+1)-bit signature $\{p_{2\ldots2}, p_{02\ldots2} \cdots p_{2\ldots20}\}$ which is an SPS if $p_{2\ldots2}$ is odd or if two or more of the remaining subparities are odd. This SPS also requires an exhaustive application of inputs but is not optimal in length. In fact, an alternative of at most two bits to Akers' SPS always exists for any irredundant function. For if $p_{2\ldots2}$ is odd, it forms an SPS of length 1. Otherwise if any one of the other n subparities, say $p_{02\ldots2}$, is odd, then from Theorem 5 there exists $p_{2u'} = 1$ so that $\{p_{02\ldots2}, p_{2u'}\}$ is an SPS. Thus an SPS of ≤ 2 bits may not exist only if all subparities in Akers' signature are even.

The preceding discussion on Akers' SPS and its alternative implies that over $\frac{2^{n+1}-1}{2^{n+1}}$ of all n-variable functions have an SPS of ≤ 2 bits (ignoring corrections for redundant functions, to which Theorem 5 does not apply). We formalize this result with a much better bound in Theorem 6, after we establish some basic results for the probabilistic analysis of constrained parity signatures.

Lemma 1: Let the signature $S = \{p_\alpha, p_\beta \cdots p_\gamma\}$ be k distinct subparities extracted from coefficients of an n-variable FBE. Then S is uniformly distributed among all n variable functions. i.e.

$$\text{Prob}\left\{(p_\alpha = a_1) \wedge (p_\beta = a_2) \wedge \cdots \wedge (p_\gamma = a_k)\right\} = \left[\frac{1}{2}\right]^k$$

where $a_i \in \{0, 1\}$ and $1 \leq i \leq k \leq 2^n$.

Proof: Given a fixed basis δ, the δ-FBE uniquely represents a Boolean function. There are 2^n coefficients in the δ–FBE with 2^{2^n} possible assignments (of 0's and 1's) representing all the 2^{2^n} possible n-variable functions. Hence any subset of k of these coefficients is uniformly distributed. □

Corollary 2: Let p_α and p_β be two distinct subparities. Then for $a, b \in \{0, 1\}$,

(a) $\text{Prob}\left\{p_\alpha = 0\right\} = \text{Prob}\left\{p_\alpha = 1\right\} = \frac{1}{2}$

(b) $\text{Prob}\left\{(p_\alpha = a) \wedge (p_\beta = b)\right\} = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$

Proof:

(a) Obvious, since every subparity appears in at least one FBE.

(b) Using the P-cube representation of the parity spectrum and the FBEs developed in Chapter 4, it can be visualized that any two subparities constitute the vertices of at least one embedded Boolean hypercube which represents an FBE. Thus any two subparities can be interpreted as coefficients of the same FBE, and

result follows from Lemma 1. □

Lemma 1 and Corollary 2 are of significance because subparities are not always independent of one another. For example, $\text{Prob}\left\{p_{02} = p_{12} = p_{22} = 0\right\} = \frac{1}{4}$, since $p_{22} = p_{02} \oplus p_{12}$. Lemma 1 can actually be generalized to any set of *independent* subparities from the parity spectrum. A vigorous proof for such a generalization is necessarily complex and is not pursued here. Nevertheless, our discussion should provide useful hints for analyzing *aliasing probabilities*[McC85] of constrained parity signatures based on a uniform distribution of faulty functions.

Returning to the problem of minimizing the test space, we refine Theorem 6 in [Dam89] to the following result:

Theorem 6: For n > 5, over 99.99% of all n-variable functions have an SPS of ≤ 2 bits.

Proof: We give an upper bound on the probability that such an SPS does not exist by considering only subparities from the 0–FPE of $f(x_n \cdots x_1)$. This simplifies the analysis as we know from Lemma 1 that these subparities are independent of one another.

Let $\alpha \sim (\alpha_n \cdots \alpha_1)$ where $\alpha_i \in \{0, 2\}$, and let $\overline{\alpha} \sim (\overline{\alpha}_n \cdots \overline{\alpha}_1)$ where $\overline{0} = 2$ and $\overline{2} = 0$. Then $\{p_\alpha, p_{\overline{\alpha}}\}$ is an SPS iff $p_\alpha = p_{\overline{\alpha}} = 1$. Now for a fixed α,

$$\text{Prob}\left\{(p_\alpha \neq 1) \vee (p_{\overline{\alpha}} \neq 1)\right\} = 1 - \text{Prob}\left\{(p_\alpha = 1) \wedge (p_{\overline{\alpha}} = 1)\right\} = \frac{3}{4}$$

Let $1 \leq k \leq n-1$, then the combined probability for all α such that $\omega_2(\alpha) = k$ is

$$\text{Prob}\left\{(p_\alpha \neq 1) \vee (p_{\overline{\alpha}} \neq 1),\ \forall \omega_2(\alpha) = k\right\} = \left[\frac{3}{4}\right]^{\binom{n}{k}}$$

except in a special case when n is even and $k = \frac{n}{2}$. If we combine the probabilities for all k, then each $\{p_\alpha, p_{\overline{\alpha}}\}$ pair is also counted as $\{p_{\overline{\alpha}}, p_\alpha\}$. Hence,

$$\text{Prob}\left\{(p_\alpha \neq 1) \vee (p_{\overline{\alpha}} \neq 1),\ \forall 1 \leq \omega_2(\alpha),\ \omega_2(\overline{\alpha}) \leq n-1\right\} = \left[\frac{3}{4}\right]^{\frac{1}{2}\cdot\sum_{k=1}^{n-1}\binom{n}{k}}$$

$$= \left[\frac{3}{4}\right]^{2^{n-1}-1}$$

When $\omega_2(\alpha) = 0$ or n, $p_\alpha = p_{2\ldots2}$ or $p_{\overline{\alpha}} = p_{2\ldots2}$. Therefore,

$$P(n) = \text{Prob}\left\{(p_{2\ldots2} \neq 1) \wedge \left[(p_\alpha \neq 1) \vee (p_{\overline{\alpha}} \neq 1),\ \forall 1 \leq \omega_2(\alpha),\ \omega_2(\overline{\alpha}) \leq n-1\right]\right\}$$

$$= \frac{1}{2}\cdot\left[\frac{3}{4}\right]^{2^{n-1}-1}$$

Clearly,

$$P(n) \geq \text{Prob}\left\{f(x_n \cdots x_1) \text{ does not have an SPS of} \leq 2 \text{ subparities}\right\}$$

and the theorem follows since $P(n) < 0.007\%$ for all $n > 5$. □

For an example of a "long" SPS the XOR function $f(x_n \cdots x_1) = x_1 \oplus \cdots \oplus x_n$ has $p_\alpha = 0$ for all $\omega_2(\alpha) > 1$ (see property P15 in Table 3–1). Thus it requires a minimal SPS length of n.

In Table 1, the optimal SPS test time and test space of ten sample functions are derived using properties of the parity spectrum listed in Table 3–1. The first nine functions are taken from Table II of [Dam89], which gives only sub-optimal test time and does not consider the test space requirements. The tenth function in Table 1 is

$$k(x_n \cdots x_1) = x_1^{e_{11}} \cdots x_r^{e_{1r}}\, x_{r+1} \vee x_1^{e_{21}} \cdots x_r^{e_{2r}}\, x_{r+2}$$

$$\vee \cdots \vee x_1^{e_{(n-r)1}} \cdots x_r^{e_{(n-r)r}}\, x_n \qquad (5)$$

where $n \geq 6$, $r \geq \log_2(n-r)$, $e_{ij} \in \{0, 1\}$, $x_i^0 = x_i$, $x_i^1 = \overline{x}_i$, and $e_{i1} \cdots e_{ir}$ and

	Function		time ($\leq 2n$)	space ($\leq n$)
1.	*AND*	$x_1 \Lambda \cdots \Lambda x_n$	$n+1$	1
2.	*NAND*	$\overline{x_1 \Lambda \cdots \Lambda x_n}$	$n+1$	1
3.	*OR*	$x_1 \vee \cdots \vee x_n$	$n+1$	1
4.	*NOR*	$\overline{x_1 \vee \cdots \vee x_n}$	$n+1$	1
5.	*XOR*	$x_1 \oplus \cdots \oplus x_n$	$n+1$	n
6.	*XNOR*	$\overline{x_1 \oplus \cdots \oplus x_n}$	$n+1$	n
7.	*QUADRATIC*	$x_1x_2 \oplus x_1x_3 \oplus \cdots \oplus x_{n-1}x_n$	$n+1$	$\left\lceil \frac{n}{2} \right\rceil$
8.	*THRESHOLD*	$f(X) = 1$ iff $\sum_{i=1}^{n} x_i = 2$	$n+1$	$\begin{cases} 2 & \text{if } n = 4m \text{ or } 4m+1 \\ 1 & \text{otherwise} \end{cases}$
9.	*THRESHOLD*	$f(X) = 1$ iff $\sum_{i=1}^{n} x_i = c$	$n+1$	$\leq \left\lceil \frac{n}{2} \right\rceil$
10.	*KUHL-REDDY*	$k(X)$	$\geq 2(n-r)$	$\leq n-r+1$

***Table* 1:** Optimal Time/Space Complexities for an SPS

$e_{j1} \cdots e_{jr}$ differ in at least one position whenever $i \neq j$, $\forall\ 1 \leq i, j \leq n-r$. This function is first described by Kuhl and Reddy[Kuh78] as a counter-example to the conjecture that $(n + 1)$ is an upper bound on the length of terminal stuck-at fault tests. Since tests for stuck-at faults involving x_{r+i} and x_{r+j} are different whenever $i \neq j$, at least $2(n - r)$ tests are required[Kuh78].

6.6. General Stuck-at Faults

The vast number of possible stuck-at faults in a NUT makes it difficult to derive an optimal constrained parity signature to detect all of them. For $n > 3$, over 99.5% of all stuck-at faults are vacuous. Thus the derivation is greatly simplified by

incorporating an SPS, which is usually short, into the signature. Since an SPS can also detect non-vacuous faults, it should be selected accordingly.

Disregarding stuck-at faults involving inputs and/or output that are obviously vacuous, the number of *potentially* non-vacuous faults is cubic in the number of internal lines in the NUT. To further simplify the analysis of these faults, we use the concepts of *minimal checkpoints*[Bos71], *fault equivalence*[She72], *fault collapsing*[She72], and *connectivity considerations*[Ake88, Lui86]. We also generalize an observation by Akers[Ake88] to derive tests for non-vacuous stuck-at faults that *becomes vacuous* if only a *subnetwork* of the NUT is considered. These fault reduction/test generation techniques will be described below as we consider the problem of constrained parity testing of all stuck-at faults in three progressively more complex types of irredundant networks — *fanout-free networks*, *internally fanout-free networks*, and *general irredundant networks*.

6.6.1. Fanout-free Networks

Two faults are said to be *equivalent* if they cause the NUT to implement the same faulty function[Sch72]. Bossen and Hong[Bos71] define the *minimal checkpoints* of a network to be the set of all fanout-free input lines and all the branches of each (input or internal) fanout line, and show that any stuck-at fault is equivalent to a stuck-at fault involving these checkpoints.

In a *fanout-free network (tree network)*, the minimal checkpoints are the input lines, and any stuck-at fault is vacuous since it is equivalent to a multiple-input stuck-at fault. Thus an SPS is a signature for all stuck-at faults.

Tzidon et al.[Tzi78] show that a fanout-free network composed of AND/OR/NAND/NOR/NOT gates only implements a function with odd parity, so that $\{p_2 \ldots_2\}$ is an SPS. This SPS requires exhaustive application of input patterns, and an alternative such as from Theorem 5 or the proof of Theorem 6 may be used in order to reduce the test time.

For an n-input fanout-free network composed also of XOR/XNOR gates, it is shown in [Lui86] that the parity $p(f) = p_2 \ldots_2$ is even and an n-bit spectral signature called the *height* $H(f)$ is a test for all stuck-at faults. Testing using $H(f)$ is also exhaustive and requires sophisticated compactor circuitry. Thus it is more economical to use an SPS, which is usually less than n bits and requires non-exhaustive inputs.

Only a small number of Boolean functions can be implemented by fanout-free networks of the aforementioned logic gates. Note that it is possible to implement fanout-free networks using more complex gate types, and an SPS can still detect all stuck-at faults in such a network.

6.6.2. Internally Fanout-free Networks

An *internally fanout-free (IFF) network* has fanouts from its input lines only. (For example, a PLA is an IFF network.) Thus the minimal checkpoints are the fanout-free input lines and all input fanout branches. Since a stuck-at fault involving all fanout branches of an input disconnects the input from the network output, it is vacuous. Hence it is shown in [Lui86] that the number of potentially non-vacuous checkpoint stuck-at faults is at most $U(n) = (\prod_{i=1}^{n}(3^{k_i}-2^{k_i})) - 1$, where k_i is the number of fanout branches of input x_i, $1 \leq i \leq n$. The obvious approach is to analyze each of the U(n) faults, discard those that are vacuous, and derive tests for the non-vacuous ones. Since U(n) increases exponentially with the number of fanout branches, this approach may not be viable if U(n) is too large. Fortunately, more faults can be shown to be vacuous by fault equivalence considerations.

Example 1: Consider the IFF network (Figure 3) from [Lui86] realizing $f(x_4 \cdots x_1) = x_4x_3 \vee \overline{x}_2 \oplus x_1$. There are $U(n) = 5^2-1 = 24$ candidate non-vacuous faults at the checkpoints a, b, c and d (Figure 3). Let the faults a/0, bc/10 denotes respectively line a s-a-0, lines bc s-a-10 etc. Then the following 8 of the 24 faults ad/00, ad/01, ad/11, bc/00, bc/10, bc/11, ac/11 and bd/00 are equivalent to the vacuous faults cd/00, cd/01, ab/11, ab/00, ab/01, cd/11, e/1 and e/1 respectively. Thus

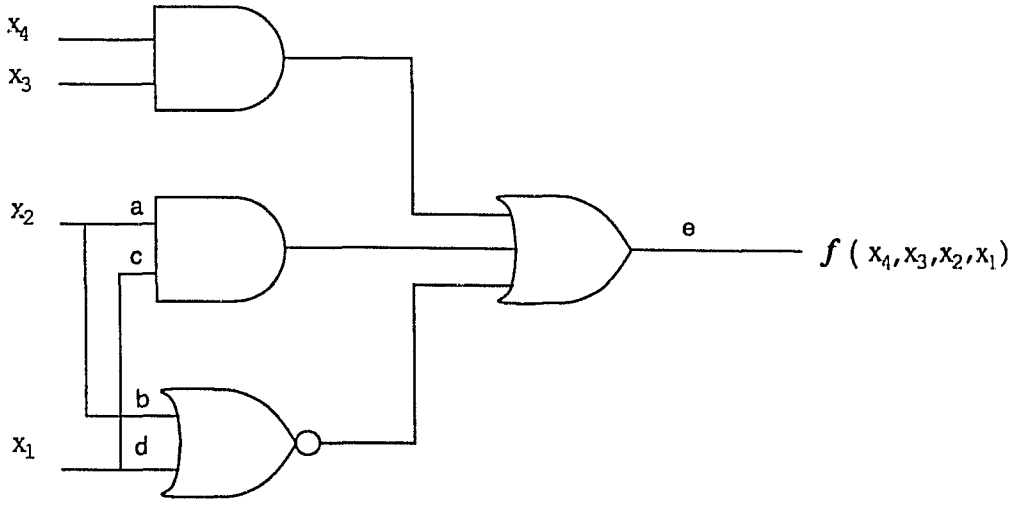


Figure 3: An IFF Implementation

these faults are vacuous and are covered by an SPS. □

After we have discarded all vacuous faults, there remains the task of analyzing the truly non-vacuous ones and deriving constrained parity tests for them. As pointed out by Akers[Ake88], analysis by fault simulation is expensive for compaction techniques since we have to simulate the behaviour of the faulty network for all possible input patterns. Here, we generalize an observation in [Ake88] to a new technique for deriving tests for certain non-vacuous faults. This technique is especially easy to apply for stuck-at faults among input fanout branches.

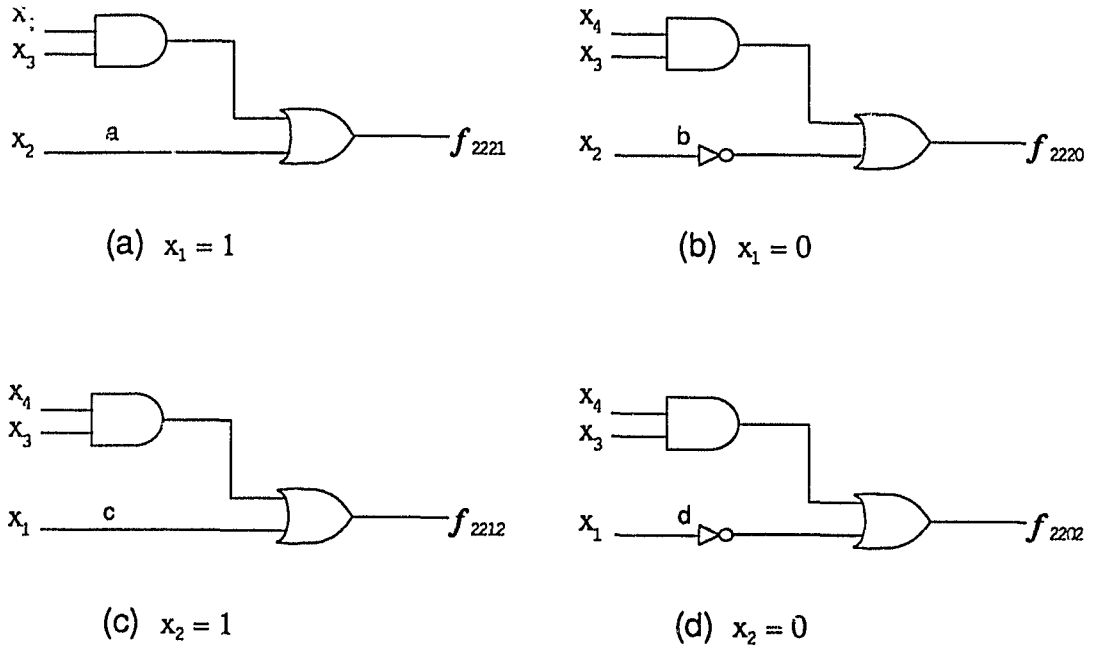


Figure 4: Subnetworks from the IFF Implementation

The idea is that by constraining a set of inputs, say $x_k \cdots x_1$, at $u_k \cdots u_1$, the NUT is reduced into a *subnetwork* that implements the subfunction $f_{2 \cdots 2}{}_{u_k \cdots u_1}$. Now if a non-vacuous fault in the NUT is vacuous in this subnetwork, then the *subparities of this subnetwork* may be used to detect this fault!

Example 2: We continue with the analysis of the NUT in Example 1. Using fault collapsing[Sch72], the remaining 16 faults can be reduced into 8 sets of equivalent faults {a/0, c/0, ac/00, ac/01, ac/10}, {a/1}, {b/0}, {b/1, d/1, bd/11, bd/10,

bd/01}, {c/1}, {d/0}, {ad/10} and {bc/01}.

Consider the first two fault sets which are equivalent to the two non-vacuous stuck-at faults at line a, a fanout branch of x_2. If we constrain the input x_1 at 1 so that line b, the other fanout branch of x_2, is blocked, then a/0 and a/1 becomes vacuous in this subnetwork for f_{2221} (Figure 4a). Therefore, $p_{\alpha_4\alpha_3 21}$ is a test for both a/0 and a/1 if $p_{\alpha_4\alpha_3 21} = 1$ for some $\alpha_4, \alpha_3 \in \{0, 1, 2\}$. Similarly, $p_{\beta_4\beta_3 20} = 1$ is a test for both b/0 and b/1 (Figure 4b), $p_{\gamma_4\gamma_3 12} = 1$ is a test for both c/0 and bc/10 (Figure 4c), and $p_{\delta_4\delta_3 02} = 1$ is a test for both d/1 and ad/01 (Figure 4d). Thus an SPS can be chosen to also cover all the non-vacuous faults.

In Table 2, the parity spectrum of $f(x_4 \cdots x_1)$ is easily hand-computed from its truth table (Table 2 may be interpreted as a planar projection of a 4-variable P-cube).

		β (x_2x_1)								
	$P_{\alpha\beta}$	00	01	02	10	11	12	20	21	22
	00	1	0	1	0	1	1	1	1	0
	01	1	0	1	0	1	1	1	1	0
	02	0	0	0	0	0	0	0	0	0
α	10	1	0	1	0	1	1	1	1	0
(x_4x_3)	11	1	1	0	1	1	0	0	0	0
	12	0	1	1	1	1	1	1	1	0
	20	0	0	0	0	0	0	0	0	0
	21	0	1	1	1	0	1	1	1	0
	22	0	1	1	1	0	1	1	1	0

Table 2: Parity Spectrum for the IFF Implementation

By inspection, a possible SPS for all (vacuous and non-vacuous) stuck-at faults in the NUT is $S_1 = \{p_{0020}, p_{0021}, p_{1202}, p_{2112}\}$.

Note that the above procedure may not always yield the best solution. In the example, since the faulty parities $\hat{p}_{2222}$ for all the non-vacuous faults are odd[Lui86], S_1 can be replaced by a shorter signature $S_2 = \{p_{2222}, p_{1221}, p_{2112}\}$. On the other hand, S_2 requires the exhaustive application of all input patterns while S_1 is non-exhaustive. □

Part of our on-going research focuses on the further development of methods for non-vacuous faults. It is hoped that the method based on subnetwork consideration as illustrated in the above example can be automated and incorporated into a fault simulator/test generator for constrained parity testing.

6.6.3. General Irredundant Networks

For networks with internal fanouts, all such fanout branches are also checkpoints. The same ideas of connectivity considerations and fault equivalence as illustrated for IFF networks can be used to identify and eliminate all the vacuous faults from further analysis. Tests for the non-vacuous faults can then be derived by considering subnetworks that cause these faults to become vacuous or by fault simulation. This approach is only feasible to networks with small number of fanouts.

If a network has a high number of fanouts, the cost of fault simulation and test generation can be very high. Although an SPS provides high stuck-at fault coverage, it is not a sufficient test by itself because stuck-at faults of low multiplicities occurs more frequently in practice[Gol77], and an SPS is not guaranteed to provide high coverage for them. One approach is to abandon the complete stuck-at fault test requirement and derive instead a constrained parity signature that covers all stuck-at faults up to a designated multiplicity. An SPS can still be incorporated into the final signature to reduce fault simulation cost and to ensure that all vacuous faults (and hence most stuck-at faults) are covered. This is the approach taken by the hybrid testing scheme described in the next section.

6.7. Hybrid Syndrome Testing

Syndrome testing[Sav80] applies all input patterns to the NUT and verifies the ones count of its responses. Now the parity of the syndrome is equal to the parity of the implemented function. Thus if the function parity is odd, syndrome testing covers all vacuous faults[Tzi78, Car82, Lui86].

It is well known[Tzi78] that a *unate* line, which does not directly or indirectly fanout to reconverging paths with unequal inversion parity, is syndrome-testable for both single stuck-at faults. An *internally unate network* is one where only input lines can be non-unate. For an internally unate network implementing a function with odd parity, syndrome testing covers all single stuck-at faults and all vacuous faults. Since single stuck-at faults occur the most frequently in practice and most stuck-at faults are vacuous, syndrome testing is effective even though some stuck-at faults may not be covered.

For an internally unate network implementing a function with an even parity, syndrome testing usually does not cover all vacuous faults. In fact, single stuck-at faults involving non-unate inputs may not even be covered[Tzi78]. Thus it is suggested in [Lui86] that an odd superfunction of one extra input be implemented instead so that syndrome testing can still be effective. However, since the ever increasing density of VLSI chips has already made packaging difficult by requesting more external pins per chip, the cost of an extra pin for testing may not be practical. Furthermore, there remains the problem of detecting stuck-at faults in networks not designed for syndrome testability. These problems can be solved if we combine syndrome testing with constrained parity testing. In particular, the syndrome coupled with an SPS is sufficient to cover all single stuck-at faults and all vacuous faults in an internally unate network without the need for expensive fault simulation.

For networks with small number of non-unate lines, the hybrid syndrome scheme can still be used as long as sufficient subparities are selected to cover all single stuck-at faults that are syndrome-untestable. In this case, fault simulation may be needed to identify syndrome-untestable faults and to derive constrained parity tests

for them.

6.8. Summary

In this Chapter, we consider the testing of single-output irredundant combinational networks using parities and subparities of the implemented functions. This technique is a generalization of many existing ones including test vector testing, parity testing[Car82], parity signature testing[Ake88], and Reed-Muller coefficient testing[Dam89].

Testability conditions for single- and multiple-input stuck-at faults are established. By considering vacuous faults, an implementation independent spanning parity signature (SPS) is introduced to cover most stuck-at faults in the network under test (NUT). Many SPSs are possible and results are presented for choosing one that is short in test time and/or signature length. The SPSs for some standard networks are also investigated.

Using the SPS as a basis, the problem of testing all stuck-at faults in a NUT with small number of fanouts is considered, and a method of deriving tests for non-vacuous faults by considering subnetworks of the NUT is described. A hybrid scheme combined with syndrome testing is also proposed to eliminate or reduce the need for expensive fault simulation in the testing of networks with high number of fanout lines.

The proposed technique also applies to sequential circuits since their testing can be reduced to the race-free testing of combinational circuits by using design for testability techniques such as Level Sensitive Scan Design (LSSD)[Wil84]. Further research is required to implement a fault simulator/test generator for constrained parity testing of stuck-at faults. The extension to the bridging fault model[Mei74] should also be investigated. The testing of multiple output networks, in particular programmable logic arrays (PLAs), is also of interest. Finally, it may be possible to extend the testing technique to multi-valued logic networks.

CHAPTER 7

CONCLUSION AND FUTURE RESEARCH

7.1. Summary of Contributions

In this research, a new, intuitive approach to the study of Boolean functions has been developed. This approach is based on the *parity spectrum*, which is a vector of the 3^n *parities of subfunctions* of an n-variable Boolean function. By applying the parity spectrum approach to problems in the *modulo-2 logic design* and *fault detection* of digital logic networks, it is shown that this new approach simplifies, unifies and extends many previous results in these areas.

In Chapter 3, the parity spectrum is defined and related to the classical theory of *Boolean difference* for defining and examining various formal properties of Boolean functions. Values of Boolean differences are made more comprehensible by interpreting them as subfunction parities in the parity spectrum. Fundamental properties of the parity spectrum, some of which have been previously described using Boolean difference and other concepts, are established in Table 3–1.

The usefulness of the parity spectrum in modulo-2 logic design stems from its connections to the 2^n *fixed polarity* and the 3^n *fixed basis* modulo-2 canonical expansions *(FPEs* and *FBEs)* of an n-variable function. In Chapter 4, the 2^n coefficients of an FPE, previously expressed in cumbersome Boolean difference notation, are more directly interpreted as subfunction parities. Using the new $\circledast$ operator and a *ternary* notation, a new, compact algebraic representation for the FPEs of an n-variable function is derived from its parity spectrum. This representation is then extended to the FBEs.

From the intuitive nature of the parity spectrum, a geometric representation for the FPEs and FBEs is also developed in Chapter 4. The *parity hypercube (P-cube)*, which is a ternary hypercube of 3^n vertices corresponding to all the subfunction parities in the parity spectrum, is a graphical exposition of all the FPEs and FBEs of an n-variable function. Each FPE or FBE is represented by one of the 3^n Boolean hypercubes embedded in the P-cube. These embedded Boolean hypercubes intersect with one another, and their intersections represent coefficients being shared among different FPEs and FBEs. The number of vertices (i.e. coefficients) shared by two such cubes is 2^d, where d is the *distance* between the polarities or bases of the two FPEs or FBEs represented. Although the P-cube ceases to be a useful visual aid for $n > 3$, the insights gained from studying 2– and 3–dimensional P-cubes can be generalized to higher dimensions without difficulty. By highlighting the relationship among the FPEs and FBEs visually, it becomes easier to conceptualize algorithms that manipulate these expansions. For example, all the transforms and minimization algorithms for FPEs and FBEs in Chapter 5 can be geometrically interpreted using the P-cube.

The complete, generalized set of *Boolean matrix transforms* among FPEs and FBEs are derived in Chapter 5 and presented in a unified manner in terms of *Kronecker products* of elementary matrices. The Kronecker product approach unifies and extends the description of previous transforms developed mainly for FPEs and simplifies the derivation of efficient recursive algorithms for computing these transforms. These algorithms are presented in a compact, pseudo-PASCAL notation suitable for computer implementation and complexity analysis.

Transforms among FPEs and FBEs of an n-variable function are square matrices of dimension $2^n \times 2^n$. New, non-square matrix transforms of dimensions $2^n \times 3^n$ and $3^n \times 2^n$ are also described in Chapter 5 for computing the parity spectrum from an FPE or FBE and vice versa. Again using a Kronecker product approach, fast recursive algorithms are developed for these transforms.

Chapter 5 also tackles the *minimization* problems for FPEs and FBEs. By selecting an FPE or FBE with the *minimal weight* (i.e. the smallest number of terms), the cost of implementing a modulo-2 design of a Boolean function is optimized. Using

transforms among FPEs and FBEs, the *Gray code approach* to the minimization of FPEs is reviewed and extended to FBEs. The alternative *parity spectrum approach*, which makes use of transforms involving the parity spectrum, is also described. This approach also involves transforms from the parity spectrum to the *weights* of the FPEs and FBEs, and these transforms are also expressed in terms of Kronecker products of elementary matrices. The parity spectrum minimization approach has been previously formulated in [Bio73] using the extended truth vector derived from Boolean differences, but our manner of attack simplies the formulation and provides algorithmic descriptions suitable for computer implementation. As shown in Tables 5–2 and 5–3, the Gray code minimization algorithms are of time complexities $O(4^n)$ for FPEs and $O(6^n)$ for FBEs, and both of them are of space complexities $O(2^n)$. In contrast, the time complexities of the parity spectrum minimization algorithms are $O(3^n)$ for FPEs and $O(n \cdot 3^n)$ for FBEs, while the space complexities are $O(3^n)$ for both FPEs and FBEs. In short, the parity spectrum approach is faster than the Gray code approach, but it requires more storage.

In Chapter 6, the *constrained parity testing* scheme, which generalizes the traditional test vector testing and several other parity based techniques, is proposed. A *signature* in this compaction testing scheme is comprised of subfunction parities from the parity spectrum. Since more freedom is allowed in the choice of tests, the obtained signature usually is shorter, takes less time to apply, and provides better fault coverage than those from the less general techniques.

The constrained parity testing is applied in Chapter 6 to the detection of *multiple stuck-at faults* in *single-output combinational logic networks*. Testability conditions for detecting *single-* and *multiple-input* stuck-at faults using subfunction parities are established. The choice of tests for all the input stuck-at faults is simplified by considering the more general set of *vacuous* faults, which includes over 99.5% of all possible multiple stuck-at faults in a network with $n > 3$ inputs. A *spanning parity signature (SPS)* is derived to detect all vacuous faults. This implementation independent signature is at most n bits in length and can require the application of $2 \cdot n$ or less test patterns. The optimal space and time requirements for an SPS of some standard

functions are tabulated in Table 6–1.

The SPS is considered for detecting all possible multiple stuck-at faults. For fanout-free networks all stuck-at faults are vacuous and an SPS is a complete signature. For networks with a small number of fanouts the SPS can be augmented with extra subfunction parities to cover all stuck-at faults. By considering vacuous faults in *subnetworks* of the network under test, a heuristic approach to the derivation of tests for non-vacuous fault is described.

Although most of the multiple stuck-at faults are covered, an SPS is not a sufficient test for logic networks in general because the probabilities of occurrences of different stuck-at faults are not the same. For a network already in operation, single faults occur the most frequently and an SPS is not guaranteed to cover all of them. If the network is internally unate, a hybrid scheme combined with *syndrome testing* is proposed since all single and most multiple stuck-at faults are covered by the syndrome and the SPS.

In summary, this research has developed a new, intuitive framework on which a Boolean function can be studied. This framework is applicable to the simplification, unification and extension of a number of results in modulo-2 logic design and fault detection. Consequently, the contributions are of theoretical interest and practical value.

7.2. Related Publications

Results in this dissertation are first presented in the form of six technical reports which have been submitted for publication. The parity spectrum concept in Chapter 3 has been published in [Lui90] and the parity spectrum approach to the minimization of FPEs in a part of Chapter 5 has been presented in [Lui88]. The remaining four reports submitted cover: the structure and representations of FPEs and FBEs[Lui90a] in Chapter 4, the transforms among FPEs and FBEs and the Gray Code minimization approach[Lui90b] in a part of Chapter 5, the transforms between the parity spectrum and the FPEs and FBEs and the parity spectrum minimization approach[Lui90c] in the remaining part of Chapter 5, and the constrained parity testing technique[Lui90d] in

Chapter 6.

Recently, in [Gre90], Green also describes transforms among FPEs and FBEs and the Gray code approach to the minimization of FBEs. Nevertheless, his results appear after ours in [Lui90b] and do not provide vigorous proofs nor algorithmic descriptions found in [Lui90b]. Another report that is related but has slipped our early attention is published by Tran[Tra89], who derives a tri-state map method for the minimization of general modulo-2 expansions. It is interesting to note that his tri-state maps are planar projections of the P-cube described in Chapter 4 and [Lui90a].

7.3. Future Research Directions

Other modulo-2 canonical expansions such as those suggested by Cohn[Coh62] and Davio et al.[Dav78] should be studied using the parity spectrum. The constrained parity testing technique should be applied to other logic fault models such as bridging faults[Mei74]. An extension of the parity spectrum to multi-valued logic should be pursued. In particular, the possibility of developing *modulo-k canonical forms* comparable to those in [Pra75, Kod75] should be considered. Also, *constrained modulo-k testing* of multi-valued logic networks should be investigated, as these networks are now becoming more practical (see for example the 4-valued multiplier in [Kaw87]). Finally, applications of the parity spectrum to other areas of switching theory not mentioned in this dissertation should be examined.

REFERENCES

[Ake59] S.B. Akers, Jr., "On a Theory of Boolean Functions", *J. Soc. Indust. Appl. Math.*, Vol. 7, No. 4, pp.487-498, December 1959.

[Ake88] S.B. Akers, "A Parity Bit Signature for Exhaustive Testing", *IEEE Transac. CAD*, Vol. 7, pp.333-338, March 1988. See also same titled paper, *Digest of Papers 1986 International Test Conference*, pp.48-53, September 1986.

[Bes83] Ph.W. Besslich, "Efficient Computer Method for EXOR Logic Design" *IEE Proc.*, Vol. 130, Part E, pp.203-206, 1983.

[Bio72] G. Bioul, M. Davio, "Taylor Expansions of Boolean Functions and of their Derivatives", *Philips Research Reports*, Vol. 27, pp.1-6, 1972.

[Bio73] G. Bioul, M. Davio, J.P. Deschamps, "Minimization of Ring-Sum Expansions of Boolean Functions", *Philips Research Reports*, Vol. 28, pp.17-36, 1973.

[Boo54] G. Boole, "An investigation of the Laws of Thought, on which are founded the Mathematical Theories of Logic and Probability", 1854. *Reprinted by New York: Dover Pub.*, 1954.

[Bos71] D.C. Bossen, S.J. Hong, "Cause-Effect Analysis for Multiple Fault Detection in Combinational Networks", *IEEE Transac. Comput.*, Vol. C-20, pp.1252-1257, November 1971.

[Boy68] C.B. Boyer, "A History of Mathematics", *New York: John Wiley & Sons*, 1968.

[Cal61] P. Calingaert, "Switching Function Canonical Forms Based on Commutative and Associative Binary Operations", *AIEE Transac.*, Vol. 79, pp.808-814, January 1961.

[Car82] W.C. Carter, "The Ubiquitous Parity Bit", *Proc. 12th International Symposium on Fault Tolerant Computing (FTCS-12)*, pp.289-296, June 1982.

[Coh60] M. Cohn, "Switching Function Canonical Forms over Integer Fields", *Ph.D. Dissertation*, Harvard University, U.S.A., December 1960.

[Coh62] M. Cohn, "Inconsistent Canonical Forms of Switching Functions", *IRE Transac. Electronic Comput.*, Vol. EC-11, pp.284-285, 1962.

[Dam87] T. Damarla, "Reed-Muller Transforms and their Applications for Fault Detection", *Ph.D. Dissertation*, Boston University Graduate School, U.S.A., 1987.

[Dam89] T.R. Damarla, M. Karpovsky, "Fault Detection in Combinational Networks by Reed-Muller Transforms", *IEEE Transac. Comput.*, Vol. C-38, pp.788-797, June 1989.

[Dav71] M. Davio, "Ring-Sum Expansions of Boolean Functions", *Symposium on Computers and Automata, Polytechnic Institute of Brooklyn*, pp.411-418, April 1971.

[Dav78] M. Davio, J.P. Deschamps, A. Thayse, "Discrete and Switching Functions", *McGraw-Hill, Inc.*, 1978.

[Die71] D.L. Dietmeyer, "Logic Design of Digital Systems", *Allyn & Bacon*, 1971.

[Eld59] R.D. Eldred, "Test Routines Based on Symbolic Logic Statements", *Journal of the ACM*, Vol. 6, No. 1, pp.33-36, January 1959.

[Eve67] S. Even, I. Kohavi, A. Paz, "On Minimal Modulo 2 Sums of Products for Switching Functions", *IEEE Transac. Electronic Comput.*, Vol. EC-16, pp.671-674, October 1967.

[Fis74] L.T. Fisher, "Unateness Properties of AND-EXCLUSIVE-OR Logic Circuits", *IEEE Transac. Comput.*, Vol. C-23, pp.166-172, February 1974.

[Fle83] H. Fleisher, M. Tavel, J. Yeager, "Exclusive-OR representations of Boolean functions", *IBM J. Res. Develop.*, Vol.27, pp.412-416, July 1983.

[Fle87] H. Fleisher, M. Tavel, J. Yeager, "A Computer Algorithm for Minimizing Reed-Muller Canonical Forms", *IEEE Transac. Comput.*, Vol. C-36, pp.247-250, 1987.

[Flo56] I. Flores, "Reflected Number Systems", *IEEE Transac. Electronic Comput.*, Vol. EC-5, pp.79-82, June 1956.

[Fro77] R.A. Frohwerk, "Signature Analysis: A New Digital Field Service Method", *Hewlett-Packard Journal*, pp.2-8, May 1977.

[Goe81] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", *IEEE Transac. Comput.*, Vol. C-31, pp.215-222, March 1981.

[Gol77] L.H. Goldstein, "A Probabilistic Analysis of Multiple Faults in LSI Circuits", *IEEE Comput. Soc. Repository*, R-77-304, Long Beach, C.A., 1977.

[Gra81] A. Graham, "Kronecker Products and Matrix Calculus: with Applications", *Ellis Horwood Ltd., Chichester,* 1981.

[Gre87] D.H. Green, "Reed-Muller Expansions of Incompletely Specified Functions", *IEE Proc.*, Vol. 134, Part E, pp.228-236, September 1987.

[Gre90] D.H. Green, "Reed-Muller Canonical Forms with Mixed Polarity and their Manipulations", *IEE Proc.*, Vol. 137, Part E, pp.103-113, January 1990.

[Hay76] J.P. Hayes, "Transition Count Testing of Combinational Logic Circuits" *IEEE Transac. on Comput.*, Vol. C-25, pp.613-620, June 1976.

[Hur78] S.L. Hurst, "The Logical Processing of Digital Signals", *Crank, Russak & Company, Inc.*, 1978.

[Jag70] M. Jagadeesan, Y.H. Chuang, "Minimization of Boolean Functions in Modulo-2 Sum of Product Form", *1970 South Western IEEE Conference and Exhibition Record of Technical Papers* pp.473-477, 1970.

[Kau71] W.H. Kautz, "Testing faults in combinational cellular logic arrays", *Proc. 8th Annu. Symp. Switching and Automata Theory*, pp.161-174, October 1971.

[Kaw87] S. Kawahito, M. Kameyama, T. Higuchi, H. Yamada, "A High-speed Compact Multiplier Based on Multiple-Valued Bi-directional Current-Mode Circuits", *Proc. 17th Int'l Symposium on Multi-Valued Logic*, pp.172-180, May 1987.

[Kod75] K.L. Kodandapani, R.V. Setlur "Reed-Muller Canonical Forms in Multivalued Logic", *IEEE Transac. on Comput.*, Vol C-24, pp.628-636, June 1975.

[Kuh78] J.G. Kuhl, S.M. Reddy, "On the Detection of Terminal Stuck-Faults", *IEEE Transac. on Comput.*, Vol. C-27, pp.467-469, May 1978.

[KuM75] C.T. Ku, G.M. Masson, "The Boolean Difference and Multiple Fault Analysis", *IEEE Transac. Comput.*, Vol. C-24, pp.62-71, January 1975.

[Lec63] R. J. Lechner, "Transformations Among Switching Function Canonical Forms", *IEEE Transac. Elec. Comput.*, Vol. EC-12, pp.129-130, April 1963.

[Lui83] P.K. Lui, "The Application of Spectral Techniques to the Detection of Single and Multiple Stuck-at Faults in Irredundant Combinational Networks", *M.Sc. Dissertation*, University of Manitoba, Canada, January 1983.

[Lui86] P.K. Lui, J.C. Muzio, "Spectral Signature Testing of Multiple Stuck-at Faults in Irredundant Combinational Networks", *IEEE Transac. on*

Comput., Vol. C-35, pp.1088-1092, December 1986.

[Lui86b] P.K. Lui, "Fault Detecting Test Sets for Reed-Muller Type Networks - A Survey Report", *Technical report*, University of Victoria, Canada, July 1986.

[Lui88] P.K. Lui, J.C. Muzio, "A Fast Computer Algorithm for the Minimization of Fixed Polarity Modulo-2 Ring-Sum Expansions", *1988 Canadian Conference on Very Large Scale Integration*, October 1988.

[Lui90] P.K. Lui, J.C. Muzio, "Simplified Theory of Boolean Functions", *International Journal of Electronics*, Vol. 68, pp.329-341, March 1990.

[Lui90a] P.K. Lui, J.C. Muzio, "On the Structure of Modulo-2 Ring-Sum Canonical Expansions for Boolean Functions", *Technical Report*, University of Victoria, Canada, February 1989. *(To appear in International Journal of Electronics.)*

[Lui90b] P.K. Lui, J.C. Muzio, "Boolean Matrix Transforms for the Minimization of Modulo-2 Canonical Expansions", *Technical Report*, University of Victoria, Canada, May 1989. *(To appear in IEEE Transac. Comput.)*

[Lui90c] P.K. Lui, J.C. Muzio, "Boolean Matrix Transforms for the Parity Spectrum and the Minimization of Modulo-2 Canonical Expansions", *Technical Report*, University of Victoria, Canada, July 1989. *(Submitted to IEE Proc., part E,* June 1990.)

[Lui90d] P.K. Lui, J.C. Muzio, "Constrained Parity Testing", *Technical Report*, University of Victoria, Canada, January 1990. *(Submitted to Journal of Electronic Testing, Theory & Applications (JETTA),* June 1990.)

[Man84] M.M. Mano, "Digital Design", *Prentice-Hall Inc., Englewood Cliffs, N.J.*, 1984.

[Mar74] S.B. Marinkovic, Z. Tosic, "Algorithm for Minimal Polarized Polynomial Form Determination" *IEEE Transac. Comput.*, Vol. C-23, pp.1313-1315,

December 1974.

[McC85] E.J. McCluskey, "Built-in Self-Test Techniques", *IEEE Design & Test of Computers*, p.21-28, April 1985.

[Mei74] K.C. Mei, "Bridging and Stuck-at Faults", *IEEE Transac. on Comput.*, Vol. C-23, pp.720-727, July 1974.

[Mic86] A. Miczo, "Digital Logic Testing and Simulation", *Harper & Row, Publishers, Inc.*, 1986.

[Muk70] A. Mukhopadhyay, G. Schmitz, "Minimization of Exclusive Or and Logical Equivalence Switching Circuits", *IEEE Transac. Comput.*, Vol. C-19, pp.132-140, February 1970.

[Mul54] D.E. Muller, "Application of Boolean Algebra to Switching Circuit Design for Error Detection", *IEEE Transac. Comput.*, Vol. EC-3, pp.6-12, September 1954.

[Muz83] J.C. Muzio, D.M. Miller, "Spectral Fault Signatures for Internally Unate Combinational Networks", *IEEE Transac. Comput.*, Vol. C-32, pp.1058-1062, November 1983.

[Pra75] D.K. Pradhan, A.M. Patel, "Reed-Muller Like Canonical Forms for Multivalued Functions", *IEEE Transac. on Comput.*, Vol C-24, pp.206-210, February 1975.

[Pra78] D.K. Pradhan, "Universal Test Sets for Multiple Fault Detection in AND-EXOR Arrays", *IEEE Transac. Comput.*, Vol. C-27, pp.181-187, February 1978.

[Red72] S.M. Reddy, "Easily Testable Realizations for Logic Functions", *IEEE Transac. Comput.*, Vol. C-21, pp.1183-1188, November 1972.

[Red83] S.M. Reddy, M.K. Reddy, J.G. Kuhl, "On Testable Design for CMOS Logic Circuits", *Digest of Papers 1983 Int'l Test Conference*, pp.435-445, October 1983.

[Ree54] I.S. Reed, "A Class of Multiple-Error-Correcting Codes and their Decoding Scheme", *IRE Transac. Inform. Theory,* Vol. IT-4, pp.38-49, September 1954.

[Rei77] E.M. Reingold, J. Nievergelt, N. Deo, "Combinatorial Algorithms: Theory and Practice", *Prentice-Hall Inc., Englewood Cliffs, New Jersey,* 1977.

[Rob82] J.P. Robinson, C.L. Yeh, "A Method for Modulo-2 Minimization", *IEEE Transac. Comput.,* Vol. C-31, pp.800-801, August 1982.

[Sal75] K.K. Saluja, S.M. Reddy, "Fault Detecting Test Sets for Reed-Muller Canonic Networks", *IEEE Transac. Comput.,* Vol. C-24, pp.995-998, October 1975.

[Sal79] K.K. Saluja, E.H. Ong, "Minimization of Reed-Muller Canonical Expansion", *IEEE Transac. Comput.,* Vol. C-28, pp.535-537, July 1979.

[Sas90] T. Sasao, P. Besslich, "On the Complexity of Mod-2 Sum PLA's", *IEEE Transac. Comput.,* Vol. C-39, pp.262-266, February 1990.

[Sav80] J. Savir, "Syndrome-Testable Design of Combinational Circuits", *IEEE Transac. Comput.,* Vol. C-29, pp.442-451, June 1980. See also "Corrections to Syndrome-Testable Design of Combinational Circuits", *IEEE Transac. Comput.,* Vol. C-29, pp.1012, November 1980.

[Sel68a] F.F. Sellers, M.Y. Hsiao, L.W. Bearnson, "Analyzing Errors with the Boolean Difference", *IEEE Transac. Comput.,* Vol. C-17, pp.676-683, July 1968.

[Sel68b] F.F. Sellers, M.Y. Hsiao, L.W. Bearnson, "Error Detection Logic for Digital Computers", *McGraw-Hill, Inc.,* 1968.

[Sha38] C.E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits", *Transac. A.I.E.E.,* Vol. 57, pp.713-723, 1938.

[She72] D.R. Shertz, G. Metze, "A New Representation for Faults in Combinational Digital Circuits", *IEEE Transac. Comput.,* Vol. C-21, pp.858,

August 1972.

[Sus83] A.K. Susskind, "Testing by Verifying Walsh Coefficients", *IEEE Transac. Comput.*, Vol. C-32, pp.198-201, February 1983.

[Swa72] S. Swamy, "On Generalized Reed-Muller Expansions", *IEEE Transac. Comput.*, Vol. C-21, pp.1008-1009, September 1972.

[Tra89] A. Tran, "Tri-state Map for the Minimization of Exclusive-OR Switching Functions", *IEE Proc.*, Vol. 136, Part E, pp.16-21, January 1989.

[Tzi78] A. Tzidon, I. Berger, M. Yoeli, "A Practical Approach to Fault Detection in Combinational Networks", *IEEE Transac. Comput.*, Vol. C-27, pp.968-971, October 1978.

[van90] R.P. van Riessen, H.G. Kerkhoff, A. Kloppenburg, "Designing and Implementing an Architecture with Boundary Scan", *IEEE Design & Test of Computers*, pp.9-19, February 1990.

[Wad78] R.L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits", *Bell Systems Technical Journal*, pp.1449-1488, May 1978.

[Wei72] C.D. Weiss, "Bounds on the length of Terminal Stuck-Fault Tests", *IEEE Transac. on Comput.*, Vol. C-21, pp.305-309, March 1972.

[Wil84] T.W. Williams, "VLSI Testing", *IEEE Computer*, October 1984.

[WuC82] X. Wu, X. Chen, S.L. Hurst, "Mapping of Reed-Muller Coefficients and the Minimization of Exclusive-OR Switching Functions", *IEE Proc.*, Vol. 129, Part E, pp.15-20, 1982.

[Zha84] Y.Z. Zhang, P.J.W. Rayner, "Minimization of Reed-Muller Polynomials with Fixed Polarity", *IEE Proc.*, Vol. 131, Part E, pp.177-186, September 1984.

APPENDICES

APPENDIX I. Proof of Results in Table 2–1 (X9-X13)

Proof:

X9. (a) Using X5(a) and X8, $A \vee (B_1 \oplus \cdots \oplus B_{2k-1}) = \overline{\overline{A}(\overline{B_1} \oplus \cdots \oplus \overline{B_{2k-1}})} = \overline{\overline{A}(\overline{B}_1 \oplus \cdots \oplus \overline{B}_{2k-1})} = \overline{\overline{A}\,\overline{B}_1 \oplus \cdots \oplus \overline{A}\,\overline{B}_{2k-1}} = (A \vee B_1) \oplus \cdots \oplus (A \vee B_{2k-1})$.
(b) Using X5(b) and X8, $\overline{A}(B_1 \oplus \cdots \oplus B_{2k}) = \overline{A}(\overline{B}_1 \oplus \cdots \oplus \overline{B}_{2k}) = (\overline{A}\,\overline{B}_1 \oplus \cdots \oplus \overline{A}\,\overline{B}_{2k}) = (\overline{A \vee B_1}) \oplus \cdots \oplus (\overline{A \vee B_{2k}}) = (A \vee B_1) \oplus \cdots \oplus (A \vee B_{2k})$.

X10. Using X3 and X8, $A \oplus AB = A(1 \oplus B) = A\overline{B}$. Also, $\overline{A} \oplus AB = 1 \oplus A \oplus AB = 1 \oplus A\overline{B}$.

X11. Using X5, X7 and X10, $A \oplus B \oplus (A \vee B) = A \oplus \overline{B} \oplus (\overline{A \vee B}) = A \oplus \overline{B} \oplus \overline{A}\,\overline{B} = A \oplus A\overline{B} = AB$. Similarly, $A \vee B = A \oplus B \oplus AB$.

X12. Using X11(b), $A_1 \vee \cdots \vee A_k = A_1 \oplus (A_2 \vee \cdots \vee A_k) \oplus A_1(A_2 \vee \cdots \vee A_k) = \cdots = A_1 \oplus \cdots \oplus A_k$.

X13. From Shannon's decomposition (2–1.1) and X12:

$$f = \overline{x}_n f^0 \vee x_n f^1 = \overline{x}_n f^0 \oplus x_n f^1$$

since the two terms are mutually disjoint. If we put $\overline{x}_n = 1 \oplus x_n$, then

$$f = (1 \oplus x_n) f^0 \oplus x_n f^1 = f^0 \oplus x_n f^0 \oplus x_n f^1 = f^0 \oplus x_n(f^0 \oplus f^1)$$

Similarly by putting $x_n = 1 \oplus \overline{x}_n$, we have

$$f = f^1 \oplus \overline{x}_n(f^0 \oplus f^1).$$

APPENDIX II. Proof of Results in Table 3–1

Proof:

P1. As for equation (3–2).

P2. $p_\alpha(f \oplus g) = p(f_\alpha \oplus g_\alpha) = p(f_\alpha) \oplus p(g_\alpha)$ by P1 and commutativity of the $\oplus$ operator.

P3. From P2 and X10: $f \wedge g = f \oplus g \oplus (f \vee g)$.

P4. By rearranging terms in P3.

P5. If $\omega_2(\alpha) = 0$, then f_α is a constant 0 or 1. Thus $p_\alpha(\bar{f}) = \bar{f}_\alpha = \overline{p(f_\alpha)}$. Otherwise $w(T(f_\alpha)) + w(T(\bar{f}_\alpha)) = 2^q$ where $q = \omega_2(\alpha) > 0$, so that $p(f_\alpha) = p(\bar{f}_\alpha)$.

P6. Replacing α_i with x_i whenever $\alpha_i = 2$, we have $f(0, \alpha_{n-1} \cdots \alpha_1) = f_{0\alpha'} = f^0_{\alpha'}$, and $f(1, \alpha_{n-1} \cdots \alpha_1) = f_{1\alpha'} = f^1_{\alpha'}$. Therefore, $p_{0\alpha'}(f) = p_{\alpha'}(f^0)$ and $p_{1\alpha'}(f) = p_{\alpha'}(f^1)$. Now by Shannon's decomposition, we have $f(x_n\alpha_{n-1} \cdots \alpha_1) = \bar{x}_n f(0,\alpha_{n-1} \cdots \alpha_1) \oplus x_n f(1,\alpha_{n-1} \cdots \alpha_1)$, so $p_{2\alpha'}(f) = p(\bar{x}_n f_{0\alpha'}) \oplus p(x_n f_{1\alpha'}) = p(f_{0\alpha'}) \oplus p(f_{1\alpha'}) = p_{\alpha'}(f^0) \oplus p_{\alpha'}(f$

P7. Follows from P6.

P8. Also follows from P6, since $P^0 = [p_{00\cdots0} \cdots p_{02\cdots2}]^t$, $P^1 = [p_{10\cdots0} \cdots p_{12\cdots2}]^t$, and $P^2 = [p_{20\cdots0} \cdots p_{22\cdots2}]^t$.

P9. From P6(c), $p_{2\alpha'}(f) = p_{\alpha'}(f^0) \oplus p_{\alpha'}(f^1)$. Now, f is independent of x_n if and only if $f^0_{\alpha'} = f^1_{\alpha'}$, i.e. $p_{2\alpha'}(f) = 0 \ \forall\alpha'$.

P10. Obvious.

P11. Obvious.

For P12-P14, since f and h are variable disjoint, $p_{\alpha\beta}(f \oplus h) = p(f_\alpha \oplus h_\beta)$.

P12. (a) If $\omega_2(\alpha) = \omega_2(\beta) = 0$, then f_α and h_β are constants. So $p(f_\alpha \oplus h_\beta) = f_\alpha \oplus h_\beta = p(f_\alpha) \oplus p(h_\beta)$.

(b) If $\omega_2(\alpha) > 0$ and $\omega_2(\beta) = 0$, then

$$p(f_\alpha \oplus h_\beta) = \bigoplus_{u \in D(\alpha)} (f_u \oplus h_\beta) = \left[\bigoplus_{u \in D(\alpha)} f_u\right] \oplus \left[\bigoplus_{u \in D(\alpha)} h_\beta\right] = \bigoplus_{u \in D(\alpha)} f_u = p(f_\alpha)$$

since the cardinality $|D(\alpha)|$ is even so that $\bigoplus_{u \in D(\alpha)} h_\beta = 0$.

(c) Similar to (b).

(d) If $\omega_2(\alpha) > 0$ and $\omega_2(\beta) > 0$, then

$$p(f_\alpha \oplus h_\beta) = \bigoplus_{\substack{u \in D(\alpha) \\ v \in D(\beta)}} (f_u \oplus h_v) = \left[\bigoplus_{\substack{u \in D(\alpha) \\ v \in D(\beta)}} f_u\right] \oplus \left[\bigoplus_{\substack{u \in D(\alpha) \\ v \in D(\beta)}} h_v\right] = 0 \oplus 0 = 0$$

by commutativity of $\oplus$.

P13. (a) If $\omega_2(\alpha) = \omega_2(\beta) = 0$, then $p(f_\alpha \wedge h_\beta) = f_\alpha \wedge h_\beta = p(f_\alpha) \wedge p(h_\beta)$.

(b) If $\omega_2(\alpha) > 0$ and $\omega_2(\beta) = 0$, then

$$p(f_\alpha \wedge h_\beta) = \bigoplus_{u \in D(\alpha)} (f_u \wedge h_\beta) = \left[\bigoplus_{u \in D(\alpha)} f_u\right] \wedge h_\beta = p(f_\alpha) \wedge p(h_\beta)$$

by distributivity of $\oplus$.

(c) Similar to (b).

(d) If $\omega_2(\alpha) > 0$ and $\omega_2(\beta) > 0$, then

$$p(f_\alpha \wedge h_\beta) = \bigoplus_{u \in D(\alpha)} \left[\bigoplus_{v \in D(\beta)} (f_u \wedge h_v)\right] = \bigoplus_{u \in D(\alpha)} \left[f_u \wedge \bigoplus_{v \in D(\beta)} h_v\right]$$

$$= \left[\bigoplus_{u \in D(\alpha)} f_u\right] \wedge \left[\bigoplus_{v \in D(\beta)} h_v\right]$$

$$= p(f_\alpha) \wedge p(h_\beta)$$

by distributivity of $\oplus$.

P14. (a) If $\omega_2(\alpha) = \omega_2(\beta) = 0$, then $p(f_\alpha \vee h_\beta) = f_\alpha \vee h_\beta = p(f_\alpha) \vee p(h_\beta)$.

(b) If $\omega_2(\alpha) > 0$ and $\omega_2(\beta) = 0$, then

$$p(f_\alpha \vee h_\beta) = \bigoplus_{u \in D(\alpha)} (f_u \vee h_\beta) = \left[\bigoplus_{u \in D(\alpha)} f_u\right] \wedge \bar{h}_\beta = p(f_\alpha) \wedge \overline{p(h_\beta)}$$

since by X9: $\bar{A}(B_1 \oplus \cdots \oplus B_{2k}) = (A \vee B_1) \oplus \cdots \oplus (A \vee B_{2k})$.

(c) Similar to (b).

(d) If $\omega_2(\alpha) > 0$ and $\omega_2(\beta) > 0$, then

$$p(f_\alpha \wedge h_\beta) = \bigoplus_{u \in D(\alpha)} \left[\bigoplus_{v \in D(\beta)} (f_u \vee h_v)\right] = \bigoplus_{u \in D(\alpha)} \left[\bar{f}_u \wedge \bigoplus_{v \in D(\beta)} h_v\right]$$

$$= \left[\bigoplus_{u \in D(\alpha)} \bar{f}_u\right] \wedge \left[\bigoplus_{v \in D(\beta)} h_v\right]$$

$$= p(\bar{f}_\alpha) \wedge p(h_\beta) = p(f_\alpha) \wedge p(h_\beta)$$

by property P5(b) since $\omega_2(\alpha) > 0$.

For P15-P17, we make use of the following elementary properties of single variable functions:

$$p_0(x_i) = 0 \; ; \; p_1(x_i) = p_2(x_i) = 1.$$

P15. (a) If $\omega_2(\alpha) = 0$, then $\alpha_i \in \{0, 1\} \forall i$. By P12(a),

$$p_\alpha(x_n \oplus \cdots \oplus x_1) = p_{\alpha_n}(x_n) \oplus \cdots \oplus p_{\alpha_1}(x_1)$$

which is equal to 1 if and only if $\omega_1(\alpha)$ is odd.

(b) If $\omega_2(\alpha) = 1$, then without loss of generality let $\alpha_n = 2$. By P12(b),

$$p_\alpha(x_n \oplus \cdots \oplus x_1) = p_{\alpha_n}(x_n) = p_2(x_n) = 1.$$

(c) Otherwise $\omega_2(\alpha) \geq 2$ and by P12(d),

$$p_\alpha(x_n \oplus \cdots \oplus x_1) = 0.$$

P16. From P13,

$$p_\alpha(x_n \Lambda \cdots \Lambda x_1) = p_{\alpha_n}(x_n) \Lambda \cdots \Lambda p_{\alpha_1}(x_1)$$

which is equal to 1 if and only if $\omega_0(\alpha) = 0$.

P17. (a) If $\omega_2(\alpha) = 0$, then from P14(a),

$$p_\alpha(x_n \mathrm{V} \cdots \mathrm{V} x_1) = p_{\alpha_n}(x_n) \mathrm{V} \cdots \mathrm{V} p_{\alpha_1}(x_1)$$

which is equal to 1 if and only if $\alpha \neq 0$.

(b) If $\omega_2(\alpha) > 0$ and $\omega_1(\alpha) = 0$, then without loss of generality let $\alpha_n = \cdots = \alpha_{k+1} = 2$ and $\alpha_k = \cdots = \alpha_1 = 0$. From P14,

$$p_\alpha(x_n \mathrm{V} \cdots \mathrm{V} x_1) = \{p_2(x_n) \Lambda \cdots \Lambda p_2(x_{k+1})\} \Lambda \{\overline{p_0(x_k) \Lambda \cdots \Lambda p_0(x_1)}\}$$

$$= 1 \Lambda 1 = 1.$$

(c) Otherwise $\omega_2(\alpha) > 0$ and $\omega_1(\alpha) > 0$. Without loss of generality let $\alpha_n = 1$. Then by P14(c),

$$p_\alpha(x_n \mathrm{V} \cdots \mathrm{V} x_1) = \overline{p_1(x_n)} \Lambda p_{\alpha'}(x_{n-1} \mathrm{V} \cdots \mathrm{V} x_1) = 0.$$