DISCRETE FUNCTION REPRESENTATIONS UTILIZING DECISION DIAGRAMS AND SPECTRAL TECHNIQUES

By

Whitney Jeanne Townsend

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Masters of Science
in Computer Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

August 2002

DISCRETE FUNCTION REPRESENTATIONS

UTILIZING DECISION DIAGRAMS

AND SPECTRAL TECHNIQUES

By

Whitney Jeanne Townsend

Approved:

Mitchell A. Thornton Associate Professor of Electrical and Computer Engineering (Director of Thesis)	Robert B. Reese Associate Professor of Electrical and Computer Engineering (Committee Member)
R. Rainey Little Associate Professor of Computer Science (Committee Member)	James C. Harden Graduate Coordinator of Computer Engineering

A. Wayne Bennett
Dean of the College of Engineering

Name: Whitney Jeanne Townsend

Date of Degree: August 3, 2002

Institution: Mississippi State University

Major Field: Computer Engineering

Major Professor: Dr. Mitchell A. Thornton

Title of Study: DISCRETE FUNCTION REPRESENTATIONS UTILIZING DECISION DIAGRAMS AND SPECTRAL TECHNIQUES

Pages in Study: 60

Candidate for Degree of Master of Science

All discrete function representations become exponential in size in the worst case. Binary decision diagrams have become a common method of representing discrete functions in computer-aided design applications. For many functions, binary decision diagrams do provide compact representations. This work presents a way to represent large decision diagrams as multiple smaller partial binary decision diagrams.

In the Boolean domain, each truth table entry consisting of a Boolean value only provides local information about a function at that point in the Boolean space. Partial binary decision diagrams thus result in the loss of information for a portion of the Boolean space. If the function were represented in the spectral domain however, each integer-valued coefficient would contain some global information about the function. This work also explores spectral representations of discrete functions, including the implementation of a method for transforming circuits from netlist representations directly into spectral decision diagrams.

DEDICATION

to Judith and Jeanne

ACKNOWLEDGEMENTS

Sincere thanks are extended to Dr. Bob Reese and Dr. Rainey Little for serving on my committee. Appreciation is also extended to Dr. Alan Mishchenko for his help with the CUDD package.

Thanks are also extended to Dr. Jim Harden for the interest he has taken in my studies and to Donna McMurray for all of the assistance she has provided to me.

My heartfelt gratitude and utmost regard must be expressed to my advisor, Dr. Mitchell Thornton, whose encouragement and support has made all that I have accomplished possible.

Finally, acknowledgement is extended to the National Science Foundation for supporting this research under grant # CCR-0000891.

TABLE OF CONTENTS

Page

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF TABLES vi

LIST OF FIGURES vii

CHAPTER

I. INTRODUCTION 1

Motivation for this Work 1
Contributions of this Work 2
Binary Decision Diagrams 2
Spectral Computations 2
Spectral Decision Diagrams 3
Outline 3

II. BINARY DECISION DIAGRAMS 6

Properties of Binary Decision Diagrams 7
Construction of Binary Decision Diagrams 8
Limitations to Binary Decision Diagram Representations 11
Partial Binary Decision Diagrams 11
Experimental Results for Partial Binary Decision Diagrams 13
Concluding Comments on Binary Decision Diagrams 17

III. SPECTRAL COMPUTATIONS 19

Walsh Spectrum Computation 20
R-Encoding and *S*-Encoding 20
Algebraic Groups 21
Cayley Graphs 23
S-Encoding Extension 24
Other Possible Operators 27
Equivalence 28
Computation of the Adjacency Matrix 30
Concluding Comments on Spectral Computations 31

CHAPTER Page

IV. SPECTRAL DECISION DIAGRAMS 32
Background 33
Transforms 33
Walsh 34
Arithmetic 35
Reed-Muller 35
Methodology 36
Walsh 38
Arithmetic 38
Reed-Muller 39
Walsh Transformation Example 39
Experimental Results 44
Walsh 44
Arithmetic 44
Reed-Muller 51
Concluding Comments on Spectral Decision Diagrams 51

V. CONCLUSIONS 52
Summary 52
Binary Decision Diagrams 52
Spectral Computations 53
Spectral Decision Diagrams 53
Future Research 53
Binary Decision Diagrams 54
Spectral Computations 54
Spectral Decision Diagrams 54

REFERENCES 56

LIST OF TABLES

TABLE Page

2.1 ITE Forms 10

2.2 pBDD Experimental Results 18

4.1 Walsh Spectral Decision Diagrams 45

4.2 Walsh Runtime Comparison 46

4.3 Arithmetic Spectral Decision Diagrams 47

4.4 Arithmetic Runtime Comparison 48

4.5 Reed-Muller Spectral Decision Diagrams 49

4.6 Reed-Muller Runtime Comparison 50

LIST OF FIGURES

FIGURE Page

2.1 BDD for the Example Function, $f = xy + z$ 8

2.2 Pseudo-Code for the ITE algorithm 9

2.3 BLIF Netlist for the Example Function 13

2.4 The Example Function Represented as a Complete BDD 14

2.5 The Complete Karnaugh Map for the Example Function 14

2.6 The First pBDD for the Example Function 15

2.7 Karnaugh Map for the First pBDD 15

2.8 The Second pBDD for the Example Function 16

2.9 Karnaugh Map for the Second pBDD 16

3.1 Computation of the Walsh Transform Matrix for $n = 3$ 21

3.2 Calculation of the Walsh Spectrum for the Example Function 22

3.3 *R*-Encoding and *S*-Encoding of the Example Function 22

3.4 The Output Vector for the Example Function Expressed Graphically 22

3.5 *R*-Encoded Adjacency Matrix for the Example Function 25

3.6 *R*-Encoded Cayley Graph for the Example Function 25

3.7 *S*-Encoded Adjacency Matrix for the Example Function 26

3.8 *S*-Encoded Cayley Graph for the Example Function 26

3.9 All Possible Binary Boolean Operators 27

3.10 *R*-Encoded Adjacency Matrix for the Inverse of the Example Function 29

3.11 *R*-Encoded Cayley Graph for the Inverse of the Example Function 29

3.12 Transformation Diagram 31

4.1 Walsh Transform Matrix 35

4.2 Arithmetic Transform Matrix 35

4.3 Reed-Muller Transform Matrix 35

FIGURE Page

4.4 Pseudo-Code for *Traverse* 36

4.5 Pseudo-Code for *TransformLow* 37

4.6 Pseudo-Code for Walsh *TransformLow* Terminals 38

4.7 Pseduo-Code for Walsh *TransformHigh* Terminals 38

4.8 Pseudo-Code for Arithmetic *TransformLow* Terminals 38

4.9 Pseudo-Code for Arithmetic *TransformHigh* Terminals 38

4.10 Pseudo-Code for Reed-Muller *TransformLow* Terminals 39

4.11 Pseudo-Code for Reed-Muller *TransformHigh* Terminals 39

4.12 BDD Before Transformation 41

4.13 For f_1 x_{2dd} Is Transformed to x_{2sd} 42

4.14 For f_1 x_{1dd} Is Transformed to x_{1sd} 42

4.15 For f_2 x_{2dd} Is Transformed to x_{2sd} 43

4.16 For f_2 x_{1dd} Is Transformed to x_{1sd} 43

CHAPTER I

INTRODUCTION

All discrete function representations become exponential in size with respect to the number of variables in the worst case. Binary decision diagrams have become a common method for representing discrete functions in *Computer-Aided Design* (CAD) applications. For many discrete functions, binary decision diagrams do provide compact representations.

Motivation for this Work

There remain functions for which a *Binary Decision Diagram* (BDD) representation will still reach exponential size. This may occur due to either a bad variable ordering choice made during BDD construction or it may be due to the intrinsic nature of the function being represented. Thus there is interest in finding ways to represent such large functions as multiple smaller partial binary decision diagrams.

In the Boolean domain, each truth table entry consisting of a Boolean value only provides local information about a function at that point in the Boolean space. If the function were represented in the spectral domain however, each integer-valued coefficient could possibly contain global information about the function. Thus there is also interest in finding ways to represent a function as a *Spectral Decision Diagram* (SDD).

The work presented within this thesis is a portion of a larger project investigating statistical equivalence checking for combinational circuits sponsored by the National Science Foundation. This project involves extracting pairs of Haar spectral coefficients from two circuit models and computing a decreasing error probability for each matching pair. The relationship of the work in this thesis to the larger project is that within the larger project the two circuit models to be compared are represented first as partial binary decision diagrams that are then transformed into Haar spectral decision diagrams.

Contributions of this Work

The contributions of each of the primary sections of this work are described within this section. The work presented in each part is described in greater detail in the corresponding chapter.

Binary Decision Diagrams

This work has resulted in the development of a set of computer programs that take as input a textual description of a circuit and create as outputs multiple partial binary decision diagrams that together represent the complete function. These computer programs have been implemented using the *Colorado University Decision Diagram* (CUDD) package [38]. Experimental results are presented here and also in [49].

Spectral Computations

This work has extended the graph-based method for the calculation of the Walsh spectrum described by A. Bernasconi and B. Codenotti in [2]. These extensions include *S*-encoding and the exploration of other possible field relations. A new algebraic group whose corresponding Cayley graph represents the spectrum for the inverse of a function using the equivalence operator is identified. Furthermore a transformation diagram was discovered which provides a "fast" method for producing the adjacency matrix via transposition. Publication of these extensions can also be found in [50]. Examination of this matrix also contributed in small part to the more rigorous mathematical proofs developed by M. A. Thornton for the calculation of the Chrestenson spectrum as presented in [42].

Spectral Decision Diagrams

This work has resulted in the development of a computer program that implements the graph-based algorithm for computing the Walsh transform described by M. A. Thornton and R. Drechsler in [45]. It has also resulted in the development of computer programs to implement the graph-based algorithms for computing the arithmetic transform and the Reed-Muller transform described by M. A. Thornton, R. Drechsler, and D. M. Miller in [44]. These programs further enhance the algorithms as previously described by extending them to transform multi-output functions. Experimental results for these implementations are presented here and also in [48]. A second set of computer programs has also been developed that improves upon the previous versions by caching "skipped" nodes during decision diagram traversals and those results are presented in comparison with the previous work.

Outline

This work is presented in the following manner. In Chapter II properties of binary decision diagrams are reviewed and one common method for BDD construction is examined. Limitations to the use of these representations are discussed. The concept of partial binary decision diagrams is illustrated and a method for pBDD creation is described. Experimental results obtained from the creation of partial binary decision diagrams are presented. This chapter concludes with a few comments on binary decision diagrams.

Chapter III examines one of the basic spectral transformations used in this work, the Walsh transform. The chapter begins by reviewing the traditional linear algebra-based method for computing this spectrum. This is followed by a discussion of the two ways to encode a function, *R*-encoding and *S*-encoding. It then describes a newer graph-based approach to calculating the Walsh spectrum that utilizes the concepts of Cayley groups and Cayley color graphs. Next the chapter presents extensions to the graph-based approach beginning with an *S*-encoding extension and the examination of other possible operators. A new Cayley group based on equivalence is highlighted. The final extension presented is a transformation matrix created solely by transpositions of the output vector for a function. This chapter concludes with comments on spectral computations.

Chapter IV presents spectral decision diagrams. The chapter begins with a discussion of previous spectral applications in computer-aided design. This is followed by a presentation of each of the three transforms presented in this work, the Walsh transform, the arithmetic transform, and the Reed-Muller transform. The methodology used in transforming a circuit into each of these types of spectral decision diagrams is described next. Finally experimental results for the implementation of each of the transforms are presented.

Chapter V concludes this work by providing a summary of each of the preceding chapters. It highlights the contributions of this work and describes the future research opportunities arising from this work.

CHAPTER II

BINARY DECISION DIAGRAMS

Binary decision diagrams provide compact representation for discrete functions [29, 12, 21, 11, 7]. For this reason, these diagrams have seen extensive use in VLSI CAD. For some functions however, the binary decision diagram representation grows exponentially large with respect to the number of inputs.

Partial binary decision diagrams have been studied in other works as a method of determining an efficient variable ordering for binary decision diagram construction [25, 36, 19]. Presented in this chapter is a technique for partitioning a binary decision diagram into multiple binary decision diagrams each representing a subset of the information contained by the complete binary decision diagram for a function.

This chapter begins with a review of binary decision diagrams and some of their important properties. It then examines one method by which binary decision diagrams are constructed and discusses some of the constraints on their usage. Next the concept of partial binary decision diagrams is illustrated by an example function. This is followed by the presentation of experimental results for several benchmark circuits represented as partial binary decision diagrams. Closing thoughts on decision diagram representations conclude the chapter.

Properties of Binary Decision Diagrams

A *Binary Decision Diagram* (BDD) is a directed acyclic graph, $G = (V, E)$. Every BDD has two different types of vertices, terminal vertices and non-terminal vertices. The terminal vertices represent the Boolean values, 0 and 1, while the non-terminal vertices represent variables of the function represented by the BDD. Each non-terminal vertex has exactly two outgoing edges, one of which is labeled by the Boolean constant 1 (or *then*) and the other by the Boolean constant 0 (or *else*). The graph begins at a single non-terminal node, known as the root, which has no incoming edges. As shown by R. E. Bryant in [7], two very important properties that a BDD has are to be ordered and to be reduced. An ordered BDD is one in which each variable is encountered no more than once in any path and always in the same order along each path. A reduced BDD observes the following two properties. First, there are no redundant nodes in which both of the two edges leaving the node point to the same next node present within the graph. Should such a node exist, it is removed and the incoming edges redirected to the following node. Second, isomorphic subgraphs are shared, that is, if two nodes point to identical subgraphs, rather than repeat both subgraphs, the two nodes point to the same subgraph. These two properties allow a BDD representation to be canonical for a given variable ordering. A BDD that is both ordered and reduced is called a *Reduced Ordered Binary Decision Diagram* (ROBDD). In this work all references to BDDs shall imply ROBDDs. A BDD for the function, $f = xy + z$ is shown in Figure 2.1.

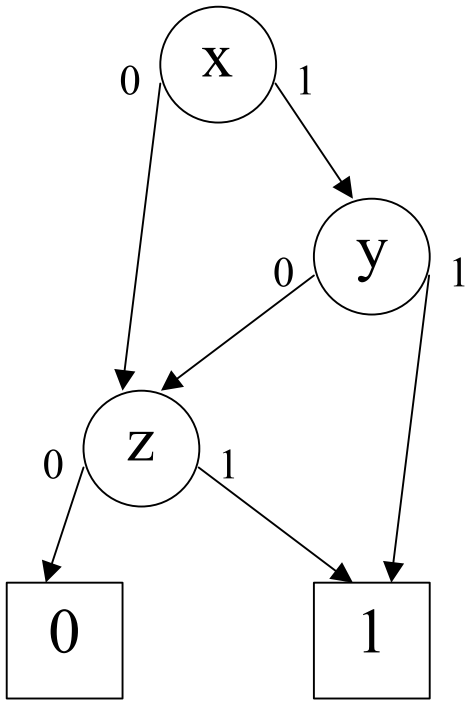


Figure 2.1 BDD for the Example Function, $f = xy + z$

Construction of Binary Decision Diagrams

BDDs are constructed by first creating individual BDDs for each variable of the function and then using the APPLY operation to build the BDD from these individual variable BDDs. The APPLY operation requires two BDDs and a Boolean operation to be applied to these BDDs as inputs and produces a resulting BDD as output. One efficient way to implement APPLY as described by K. S. Brace, R. L. Rudell, and R. E. Bryant in [4] is to use the *If-Then-Else* (ITE) operator. The ITE operator is a recursive form of the Shannon expansion theorem shown below in Equation 2.1 [37].

$$f = \overline{x_1} f_0 \oplus x_1 f_1 \qquad (2.1)$$

If $f = Z$, $x = f$, $f_0 = h$, and $f_1 = g$, then the Shannon decomposition shown in Equation 2.1 can be expressed by the following ITE shown in Equation 2.2, in which, if $(f = 1)$, then (g), else (h).

$$Z = ite(f, g, h) \qquad (2.2)$$

The terminal cases for this recursion are shown in Equation 2.3.

$$f = ite(1, f, g) = ite(0, g, f) = ite(f, 1, 0) \qquad (2.3)$$

The complement of a BDD is formed by the ITE expression shown in Equation 2.4.

$$ite(f, 0, 1) = \overline{f} \qquad (2.4)$$

All of the possible binary Boolean operators can be implemented as ITE expressions as illustrated in Table 2.1. Pseudo-code for the ITE algorithm is shown in Figure 2.2.

```
ite $(f, g, h)$
    if (terminal)
        return (result);
    else
        let x be the top variable of $(f, g, h)$;
        $T = ite\ (f_x, g_x, h_x)$;
        $E = ite\ (f_{\bar{x}}, g_{\bar{x}}, h_{\bar{x}})$;
        if T = E, return (T);
        R = newnode (x, T, E);
        return (R);
```

Figure 2.2 Pseudo-Code for the ITE Algorithm

Table 2.1

ITE Forms

Output	**Expression**	**ITE Expression**
0000	0	0
0001	$f \bullet g$	$ite(f, g, 0)$
0010	$f \bullet \overline{g}$	$ite(f, \overline{g}, 0)$
0011	f	f
0100	$\overline{f} \bullet g$	$ite(f, 0, g)$
0101	g	g
0110	$f \oplus g$	$ite(f, \overline{g}, g)$
0111	$f + g$	$ite(f, 1, g)$
1000	$\overline{f + g}$	$ite(f, 0, \overline{g})$
1001	$\overline{f \oplus g}$	$ite(f, g, \overline{g})$
1010	$\overline{g}$	$ite(g, 0, 1)$
1011	$f + \overline{g}$	$ite(f, 1, \overline{g})$
1100	$\overline{f}$	$ite(f, 0, 1)$
1101	$\overline{f} + g$	$ite(f, g, 1)$
1110	$\overline{f \bullet g}$	$ite(f, \overline{g}, 1)$
1111	1	1

Limitations to Binary Decision Diagram Representations

While BDDs are compact representations for many functions, they can reach exponential size with regard to the number of inputs for some functions. There are several reasons why this occurs. One of the reasons is that a bad ordering was chosen for the variables when the BDD was constructed. The size attained by a BDD is influenced greatly by the variable ordering chosen, however finding the best variable ordering during BDD construction is NP-hard. [3]. Therefore although heuristic techniques are used, exponential sizes can still occur. Another reason for their occurrence is that there exist circuits, such as the multiplier circuits identified by R. E. Bryant in [6], for which BDDs will always reach exponential sizes. A method for partitioning such BDDs has been examined by A. Narayan, J. Jain, M. Fujita, and A. Sangiovanni-Vincentelli in [33].

Partial Binary Decision Diagrams

A method for constructing a *partial Binary Decision Diagram* (pBDD) is developed in this work. This method for pBDD construction employs the notion of a third terminal node within the BDD containing the unrepresented portion of the circuit that is known as the *Unknown* (U) terminal. It is invoked during the construction of a BDD from a textual description or netlist, therefore the entire BDD is never constructed. The type of netlist used as input for this method is the *Berkeley Logic Interchange Format* (BLIF). An example BLIF for a small circuit with four inputs and 1 output is shown in Figure 2.3.

The example circuit for the previous BLIF file, $f = \overline{w}xz + wxy + w\overline{x}\,\overline{z}$, is shown completely represented as a BDD in Figure 2.4 and also in a Karnaugh map representation in Figure 2.5. Note that the multiple constant terminals shown are added solely to simplify the illustration. In the actual BDD only two constant nodes are present.

In Figure 2.6, the *else* edge from variable w is redirected to the U terminal to construct a pBDD. Figure 2.7 shows the corresponding Karnaugh map for this pBDD with U's replacing the Boolean constants 1 and 0 for those columns of the map in which $w = 0$. In Figure 2.8, the *then* edge from variable w is redirected to the U terminal to construct a second pBDD. Figure 2.9 shows the Karnaugh map for this pBDD with U's now replacing the Boolean constants 1 and 0 for those columns of the map in which $w = 1$.

Generation of pBDDs is achieved by modifying the code implementing the ITE function used during decision diagram construction. The modifications to the ITE algorithm have been implemented using the *Colorado University Decision Diagram* (CUDD) package [38]. The CUDD ITE functions are highly optimized and thus much more complex than the pseudo-code presented earlier that represents only the functionality of the algorithm. Within CUDD there is a recursive ITE function. It is modification within this function to the *else* and *then* children of a node during construction that allows a portion of the BDD to be redirected to the U terminal. This is done by replacing either f, g, or h within the recursive call to ITE with the pointer for U. Thus the branch becomes redirected to U as shown for the example function in Figure 2.6 and Figure 2.8. Other modifications to the CUDD code include that all functions called by the pBDD code during construction must be modified to no longer expect only two terminal nodes, but also to now check for the possibility of a third terminal.

A set of six programs was developed, each of which modifies the CUDD ITE function in a different place thus resulting in the production of a distinct pBDD for each program executed. Additionally the total number of nodes created during construction was also restricted to decrease the possibility that a pBDD could become even larger than the BDD for the complete function. Experimental results from these modifications are presented in the following section.

```
.model example
.inputs wxyz
.outputs f
.names wxzy f
01-1 1
111- 1
10-0 1
.end
```

Figure 2.3 BLIF Netlist for the Example Function

Experimental Results for Partial Binary Decision Diagrams

The experimental results presented here have been computed on a SUN Ultra 10. In all over one hundred benchmark circuits were tested using the set of programs containing the modified code. Table II provides a summary of the results obtained for several benchmark circuits using the modified ITE code. These circuits were chosen for inclusion in Table II based upon the number of nodes in the complete BDD, selecting those circuits with over 500 nodes. The column labeled BDD shows the number of nodes resulting if the circuit was built completely, while each subsequent column shows the results from a different modification to the ITE algorithm during diagram construction.

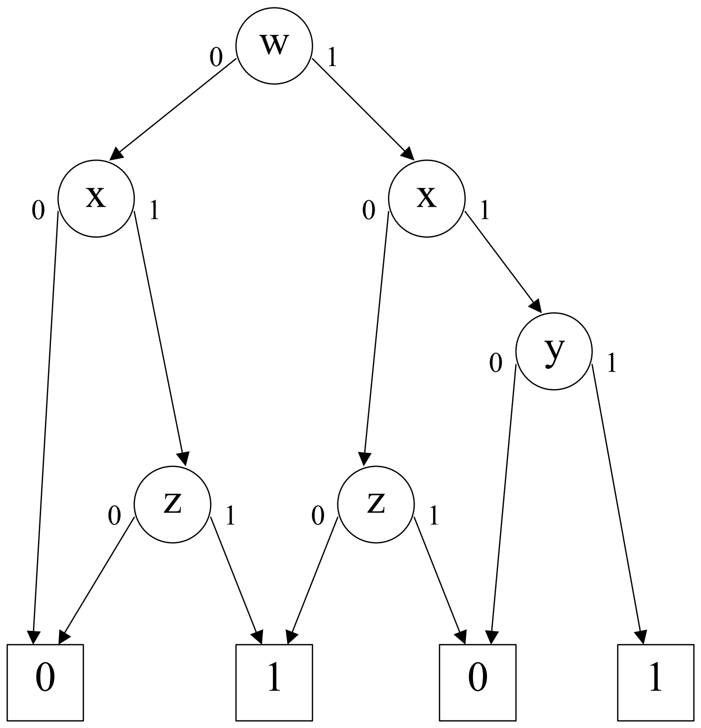
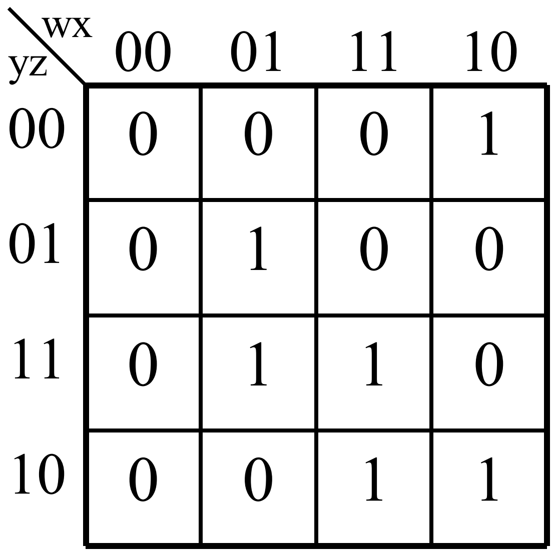


Figure 2.4 The Example Function Represented as a Complete BDD



yz \ wx	00	01	11	10
00	0	0	0	1
01	0	1	0	0
11	0	1	1	0
10	0	0	1	1

Figure 2.5 The Complete Karnaugh Map for the Example Function

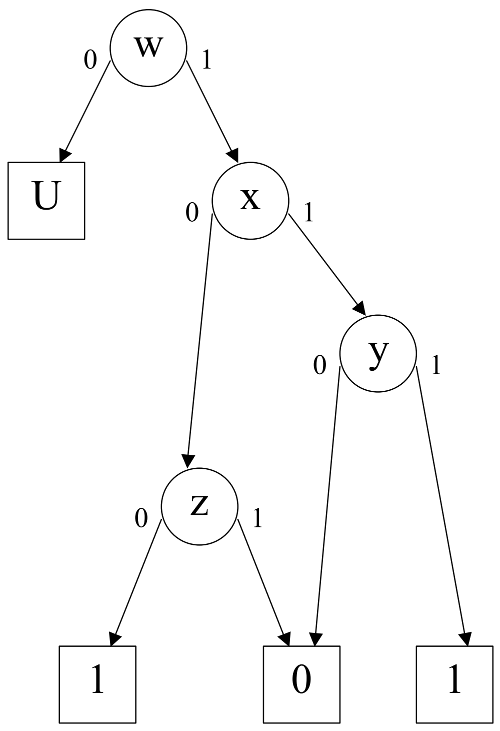


Figure 2.6 The First pBDD for the Example Function

yz \ wx	00	01	11	10
00	U	U	0	1
01	U	U	0	0
11	U	U	1	0
10	U	U	1	1

Figure 2.7 Karnaugh Map for the First pBDD

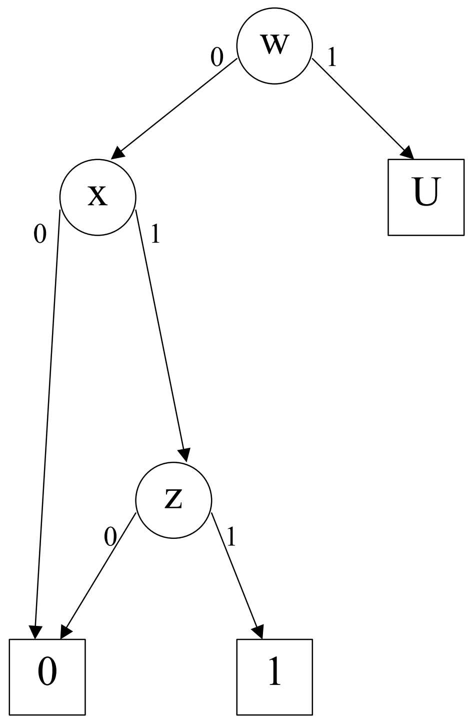


Figure 2.8 The Second pBDD for the Example Function

yz \ wx	00	01	11	10
00	0	0	U	U
01	0	1	U	U
11	0	1	U	U
10	0	0	U	U

Figure 2.9 Karnaugh Map for the Second pBDD

Concluding Comments on Binary Decision Diagrams

This chapter began by reviewing two important BDD properties. It then described a common BDD implementation technique. Two possible reasons why a BDD might reach exponential size were discussed as motivation for the present work. The first, that a poor variable ordering was chosen during decision diagram construction, and the second, that for some circuits exponential sizes will intrinsically occur. A method was described representing such BDDs by multiple pBDDs each of which represents a subset of the functionality of the circuit. This is accomplished by redirecting a portion of the functionality of the circuit to a third terminal node, U. An example function illustrated the technique by comparing pBDDs and Karnaugh map representations. Finally a table of experimental results showed several circuits represented as multiple pBDDs.

Table 2.2

pBDD Experimental Results

Circuit	BDD	pBDD1	pBDD2	pBDD3	pBDD4	pBDD5	pBDD6
alu4	933	83	294	81	73	271	87
apex1	1305	523	317	234	471	372	360
apex2	570	174	125	54	80	94	100
apex3	962	481	72	130	431	62	172
apex4	972	110	252	65	151	260	89
apex5	1095	115	369	224	234	442	200
apex6	744	405	89	320	410	61	301
bc0	587	44	113	58	30	101	47
cps	1096	471	519	396	349	454	438
dalu	1176	313	206	81	275	174	109
ex1010	1432	110	115	31	66	120	37
ex4	515	158	229	205	146	272	215
frg2	1396	405	211	226	462	212	260
intb	730	90	150	70	48	112	106
misex3	666	167	150	87	215	140	73
seq	1319	523	96	137	229	86	111
table3	786	72	430	168	117	404	112
table5	714	82	480	184	103	468	197
tial	929	77	205	105	101	205	101
vda	544	409	330	375	370	305	384
x1	626	288	437	352	197	452	348
x4	543	399	525	429	283	567	356

CHAPTER III

SPECTRAL COMPUTATIONS

Spectral techniques have found many uses in logic design. These include synthesis as described by M. A. Thornton and V. S. S. Nair in [47], testing as described by D. M. Miller and J. C. Muzio in [32], function classification as described by C. R. Edwards in [15], and verification as described by K. Radecka and Z. Zilic in [35]. Spectral methods have seen little practical application until recently however due to the computational cost for calculating the spectrum. Graph-based methods utilizing *Decision Diagram* (DD) [29, 12, 7] structures have been developed which decrease the cost for calculating the spectrum [44, 45, 31].

A. Bernasconi and B. Codenotti presented an alternative graph-based method using Cayley graphs to compute the spectrum for a function in [1,2]. This technique is of theoretical interest because it demonstrates the equivalence of the spectra of Cayley graphs and the Walsh spectra for Boolean functions.

This chapter presents several extensions to the graph-based method in [2]. In particular, alternative encodings and analysis of other possible field relations are explored. A group yielding a Cayley graph representing the spectrum for the inverse of a function is presented and a "fast" method for producing the adjacency matrix for the Cayley graphs of both groups is described.

This chapter begins with a review of the calculation of the Walsh spectrum by traditional linear algebra-based methods. This review is followed by presentation of the graph-based method from [2]. The remainder of the chapter details each of the extensions to the graph-based method.

Walsh Spectrum Computation

A function can be transformed from the Boolean domain into a number of alternative spectral domains. The traditional technique for the computation of the Walsh spectrum for a Boolean function is presented in [24]. The Walsh transform matrix if formed by the Kronecker product of n 1×1 matrices [20]. For the example function illustrated within this chapter, $n=3$ and the appropriate Walsh transform matrix is formed as shown in Figure 3.1. The use of this technique to compute the Walsh spectral coefficients for the example function, $f=\overline{x}_1\overline{x}_3+x_2\overline{x}_3+x_1\overline{x}_2x_3$, is shown in Figure 3.2.

***R*-encoding and *S*-encoding**

R-encoding is the term describing the common representation in which logic 1 is encoded by an integer 1 and logic 0 is encoded by an integer 0. An alternative representation known as *S*-encoding can also be defined in which logic 1 is encoded by an integer -1 and logic 0 is encoded by an integer +1 [24]. The example function is shown in Figure 3.3 on the left in *R*-encoding and on the right in *S*-encoding. Using S-encoding, the output vector for the example function can be expressed graphically as in Figure 3.4 The *S*-encoded spectrum for a function can be obtained directly by encoding both the transformation matrix and the output vector for the function utilizing *S*-encoding.

Alternatively the *R*-encoded coefficients can be converted to *S*-encoded coefficients by using Equation 3.1 for the zeroth coefficient and Equation 3.2 for all of the remaining $n-1$ coefficients.

$$s_0 = 2^n - 2r_0 \tag{3.1}$$

$$s_i = -2r_i \forall i \subset \{1,2,...,n\} \tag{3.2}$$

Algebraic Groups

An alternative approach for the computation of the Walsh spectrum for a Boolean function based on algebraic groups and graph theory is described in [1, 2]. This technique

$$W^3 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix} \otimes \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix} \otimes \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix}$$

$$W^3 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix} \otimes \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}$$

$$W^3 = \begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 & -1 & -1 & +1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \end{bmatrix}$$

Figure 3.1 Computation of the Walsh Transform Matrix for $n = 3$

$$\begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 & -1 & -1 & +1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 0 \\ -2 \\ 0 \\ 2 \\ 0 \\ 2 \end{bmatrix}$$

Figure 3.2 Calculation of the Walsh Spectrum for the Example Function

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

x_1	x_2	x_3	f
0	0	0	−1
0	0	1	+1
0	1	0	−1
0	1	1	+1
1	0	0	+1
1	0	1	−1
1	1	0	−1
1	1	1	+1

Figure 3.3 *R*-Encoding and *S*-Encoding of the Example Function

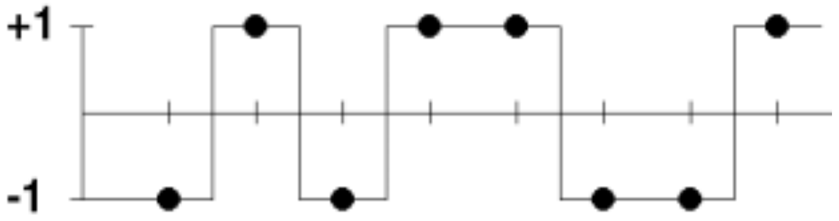


Figure 3.4 The Output Vector for the Example Function Expressed Graphically

for the computation of the Walsh spectrum for a function, f, relies upon representing the Boolean function based upon a specific definition of a group.

Recall that a group, $(M, *)$, consists of a set, M, and a binary operator on, M, $*$, such that closure, associativity, and identity hold and that inverses exist. That is, for all $m_i * m_j \in M$, the element is a uniquely defined element of M (closure). That $m_i * (m_j * m_k) = (m_i * m_j) * m_k$ holds for all $m_i, m_j, m_k \in M$ (associativity). That there exists an identity element, $e \in M$, such that, $e * m_i = m_i$ and $m_i * e = m_i$ for all $m_i \in M$ (identity). Finally that there exists an inverse element $m_i^{-1} \in M$ such that $m_i * m_i^{-1} = e$ and $m_i^{-1} * m_i = e$ for each $m_i \in M$ (inverses exist).

The algebraic group, $(M, \oplus)$, used in the technique described in [2] characterizes the Boolean function, $f : B^n \to B$. $(M, \oplus)$, is an algebraic group in which M consists of all possible minterms in B^n, that is, all points in the space defined by B^n and $\oplus$ is the binary operator for the group. This group has an identity element corresponding to an n-length bit string of all zeros and additionally for each element $m_i \in M$, $m_i^{-1} = m_i$.

Cayley Graphs

The Cayley graph is a structure that is used to relate an algebraic group to graph theory [9, 52]. The Cayley graph corresponding to a group representing the Boolean functions, f, has a vertex set, V, in which each $v_i \in V$, uniquely corresponds to an element of the set $m_i \in M$. The edge set, E, is given in Equation 3.3.

$$E = \{(m_i m_j) \in B^n \times B^n \mid f(m_i \oplus m_j) = 1\} \tag{3.3}$$

The adjacency matrix, A, for this Cayley graph is a matrix of size $2^n \times 2^n$ with $a_{ij} = 1$ if $f(m_i \oplus m_j) = 1$ and with $a_{ij} = 0$ otherwise. A is a symmetric matrix because $m_i \oplus m_j = m_j \oplus m_i$. The adjacency matrix for the example function is shown in Figure 3.5 and the corresponding Cayley graph described by A is shown in Figure 3.6.

The spectrum of a graph is defined as the set of eigenvalues for the adjacency matrix representing it in [9]. The theorems and proofs given in [2] demonstrate that the spectrum of the Cayley graph representing the group as defined in [2], which in turn represents some Boolean function, f, is identical to the Walsh spectrum utilizing *R*-encoding for the Boolean function.

All graphs have an adjacency matrix in which an edge is denoted by a logic 1 and the absence of an edge is denoted by a logic 0. The characteristic equation for the adjacency matrix yields the eigenvalues for the graph. The characteristic polynomial $C(\lambda)$ for the adjacency matrix given in Figure 3.5 is shown in Equation 3.4.

$$C(\lambda) = \lambda^8 - 8\lambda^7 + 16\lambda^6 + 16\lambda^5 - 80\lambda^4 + 64\lambda^3 \tag{3.4}$$

Solving $C(\lambda) = 0$ yields the eigenvalues, $\lambda_i \forall i = \{1, 2, ..., 8\} = \{4, 2, 0, -2, 0, 2, 0, 2\}$. These eigenvalues are the Walsh spectral coefficients for f as verified in Figure 3.2.

S-Encoding Extension

The first extension to the technique presented in [2] was to verify that in a manner analogous to that used for the matrix-based calculation of the Walsh spectrum as discussed in [24], *S*-encoding of each element, $m_i \in M$, results in a graph whose eigenvalues directly yields the *S*-encoded coefficients for the Boolean function, f. The

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Figure 3.5 *R*-Encoded Adjacency Matrix for the Example Function

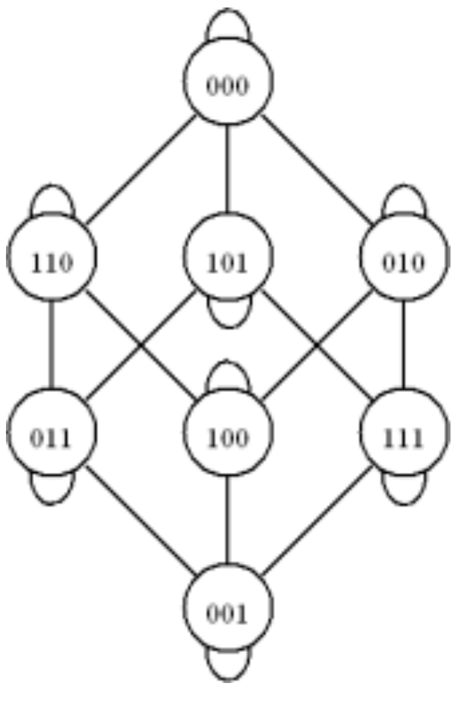


Figure 3.6 *R*-Encoded Cayley Graph for the Example Function

$$B = \begin{bmatrix} -1 & +1 & -1 & +1 & +1 & -1 & -1 & +1 \\ +1 & -1 & +1 & -1 & -1 & +1 & +1 & -1 \\ -1 & +1 & -1 & +1 & -1 & +1 & +1 & -1 \\ +1 & -1 & +1 & -1 & +1 & -1 & -1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & -1 & +1 \\ -1 & +1 & +1 & -1 & +1 & -1 & +1 & -1 \\ -1 & +1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & -1 & -1 & +1 & +1 & -1 & +1 & -1 \end{bmatrix}$$

Figure 3.7 *S*-Encoded Adjacency Matrix for the Example Function

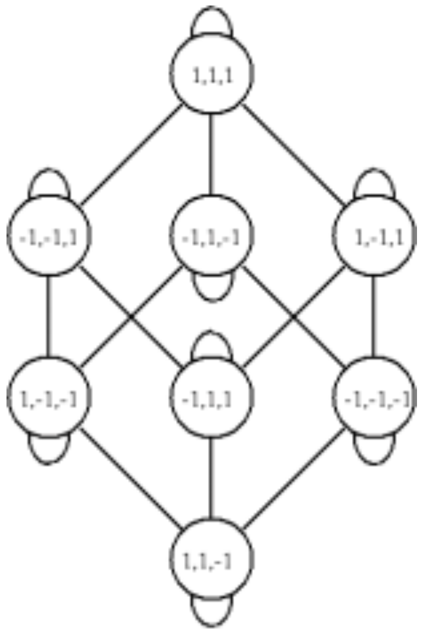


Figure 3.8 *S*-Encoded Cayley Graph for the Example Function

adjacency matrix, B, which results for the example function when utilizing *S*-encoding is shown in Figure 3.5 and the corresponding Cayley graph described by B, is shown in Figure 3.7. The characteristic polynomial for the adjacency matrix, B, is shown in Equation 3.5.

$$C(\lambda) = \lambda^8 + 8\lambda^7 - 128\lambda^5 - 256\lambda^4 \tag{3.5}$$

Solving the characteristic polynomial for this graph yields the S-encoded Walsh coefficients, $\lambda_i \forall i = \{1,2,...,8\} = \{0,-4,0,4,0,-4,0,-4\}$. Note that the topology of the graph in Figure 3.8 is unchanged from that of Figure 3.6, only the encoding of the vertices is different.

Other Possible Operators

All the remaining fifteen Boolean functions of two variables were considered as possible alternative operators to $\oplus$ in the formation of other algebraic groups. Only the two non-unate functions XOR ($\oplus$) and equivalence (XNOR, $\equiv$) were found to satisfy the definition of a group using the mapping operation from [2]. The sixteen possible Boolean operators are shown in Figure 3.9.

x	y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 3.9 All Possible Binary Boolean Operators

Equivalence

The algebraic group defined using equivalence as the Boolean operator, $(M, \equiv)$, proves to have properties similar to the correspondence between the two operators, $\oplus$ and $\equiv$. This group has an identity element corresponding to an *n*-length bit string of all ones and for each element $m_i \in M$, $m_i^{-1} = m_i$. The adjacency matrix, C, for the example function using this definition for an algebraic group is shown in Figure 3.10 and the corresponding Cayley graph is shown in Figure 3.11.

As is shown in Figure 3.11, the topology of this Cayley graph, for the same example function used previously, is quite different from the Cayley graph produced by the algebraic group, $(M, \oplus)$, as shown in Figure 3.6. Of particular interest is the presence of self-loops in Figure 3.6 and their absence in Figure 3.11. This is determined by the value of $m_0 \oplus m_0$. For the example function, f, and the Cayley graph corresponding to the algebraic group, $(M, \oplus)$, the first computation is determined as shown in Equation 3.7 and the result is a minterm.for the example function.

$$(m_0 \oplus m_0) = (000 \oplus 000) = 000 \tag{3.7}$$

For the Cayley graph corresponding to the algebraic group, $(M, \equiv)$, the first computation is determined as shown in Equation 3.8 and the result is not a minterm for the example function.

$$(m_0 \equiv m_0) = (000 \equiv 000) = 111 \tag{3.8}$$

It is the value of this first calculation that determines the presence of absence of self-loops in the corresponding Cayley graph for the function. If the result is a minterm of the function, self-loops will appear at all vertices in the graph; if the result is not a minterm

$$C = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Figure 3.10 *R*-Encoded Adjacency Matrix for the Inverse of the Example Function

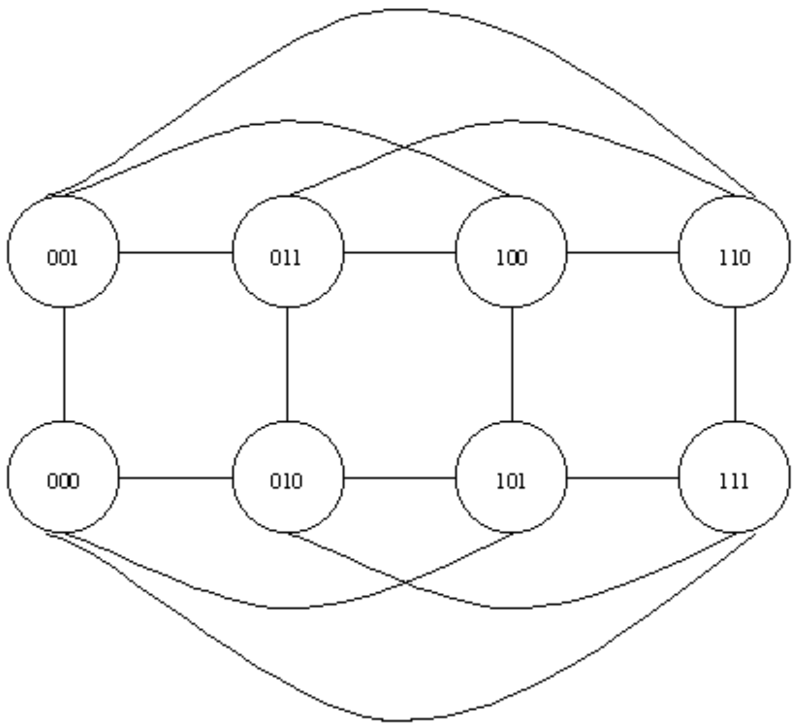


Figure 3.11 *R*-Encoded Cayley Graph for the Inverse of the Example Function

of the function, self-loops will not appear in the corresponding Cayley graph. The characteristic polynomial for adjacency matrix, C, is shown in Equation 3.9.

$$C(\lambda) = \lambda^2 - 16\lambda^6 - 16\lambda^5 + 48\lambda^4 - 64\lambda^3 \tag{3.9}$$

Solving the characteristic polynomial for the new Cayley group, $(M, \equiv)$, produces the Walsh spectrum for the inverse of the example function directly, $\lambda_i \forall i = \{1, 2, ..., 8\} = \{4, -2, 0, 2, 0, -2, 0, -2\}$.

As in the previous section on *S*-encoding, if the example function, f, is *S*-encoded using the algebraic group, $(M, \equiv)$, the *S*-encoded Walsh coefficients for the inverse of the example function can also be obtained directly from the resulting Cayley graph.

Computation of the Adjacency Matrix

During the computations required to obtain the adjacency matrix for a function using the definition of an algebraic group, a method was discovered which greatly minimizes the computational cost of producing the adjacency matrix for a function under consideration. In a method similar to the "fast transform butterfly diagrams" described by M. A. Thornton and R. Drechsler in [47] it becomes possible to obtain all the other $n_1, ..., n_n$ rows of the adjacency matrix from the n_0 row by a series of transpositions as shown in Figure 3.12.

Additionally, because the first row of the adjacency matrix for the Cayley graph corresponding to the algebraic group, $(M, \oplus)$, consists of the equation, $e * m_i = m_i \forall i \subset \{1, 2, ..., n\}$, the first row of the adjacency matrix can be obtained directly from a transposition of the output vector for the function. Conversely in the Cayley graph

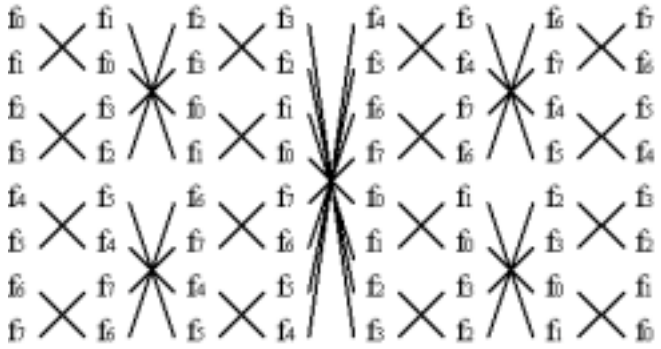


Figure 3.12 Transformation Diagram

corresponding to the algebraic group, $(M, \equiv)$, the equation, $e * m_i = m_i \forall i \subset \{1, 2, ..., n\}$, occurs in the n_n row of the adjacency matrix and thus the transformation can proceed in a similar manner from right to left.

Concluding Comments on Spectral Computations

In this chapter two ways of calculating the spectrum for a function have been reviewed. The first was a linear algebra-based method and the second was a more recent approach based on algebraic groups and graph theory. In this work a new algebraic group whose corresponding Cayley graph represents the spectrum for the inverse of a function is presented. Additionally it is shown that both the group, $(M, \oplus)$, and the group, $(M, \equiv)$, can be used to directly calculate *S*-encoded coefficients by the *S*-encoding of each element, $m_i \in M$. Finally a "fast" method for calculating the adjacency matrix by transposition is presented.

CHAPTER IV

SPECTRAL DECISION DIAGRAMS

Spectral methods and decision diagrams have been applied to many areas of digital systems design. These include synthesis [22, 47, 30, 8, 34, 24, 14, 26], function classification [24, 15], partitioning techniques [51], testing [10, 32, 23, 41], and verification [35, 46, 28]. Spectral techniques can offer a view of a problem that illuminates different properties than are readily evident in the functional domain. Traditionally, the high computational cost of computing spectral coefficients has many of the practical application of many of these techniques using the linear algebra-based methods of computation or the graph-based techniques described in the previous chapter. The emergence of graph-based algorithms utilizing *Decision Diagram* (DD) [29, 12, 7] representations now permit the spectrum to be calculated more efficiently. Decision diagrams are the state-of-the-art representation for Boolean functions in *Computer-Aided Design* (CAD) applications. It is thus very attractive to consider decision diagrams when considering alternatives such as spectral techniques.

This chapter considers the transformation of binary decision diagrams into spectral decision diagrams, a fundamental step in the application of spectral techniques to any area. This chapter begins with a brief review of several different varieties of decision diagrams. It then discusses the three transformations implemented in this work, the Walsh transform, the arithmetic transform, and the Reed-Muller transform. Next a section on methodology examines representative pseudo-code for each of the transformation algorithms, followed by a presentation of experimental results.

Background

The use of decision diagrams as a compact representation of discrete functions provides for a variety of ways that spectra may be computed or represented [44, 45, 16, 17, 18, 40, 31, 39]. In this method, a multi-output circuit represented as a *Binary Decision Diagram* (BDD) is transformed into a *Spectral Decision Diagram* (SDD). The resulting SDD is represented as a *Multi-Terminal Binary Decision Diagram* (MTBDD) [8] in which each non-terminal node has two outgoing edges, one edge representing the Boolean value 0 and the other edge representing the Boolean value 1. No edge complementation is used within the MTBDD. It is possible to further reduce the size of the SDD using edge negations as described in [31]. The terminal nodes of an MTBDD can take on any integer value allowing for the representation of spectral coefficient values.

Transforms

One property that distinguishes the different varieties of DDs is the decomposition type represented at each internal node. For BDDs, this decomposition type is a Shannon expansion, therefore each child of the node represents a cofactor about the variable represented by the node. Traversing a decision diagram and transforming each Shannon node encountered with the 2×2 matrix for the desired transform accomplishes the transformation from the Boolean domain to the spectral domain. When all nodes have been transformed in this manner, the result is the SDD for the circuit with the spectral coefficients present at the terminal nodes.

A recursive Kronecker product definition [20] used within these algorithms is shown in Equation 4.1. G in Equation 4.1 is replaced by the appropriate 1×1 matrix, depending upon the transformation desired.

$$G^n = \bigotimes_{i=1}^{n} G^1 \tag{4.1}$$

The transformation occurs in a depth-first fashion, and no node can be transformed until all nodes below it have been transformed. Special consideration must be given to those portions of the BDD in which a variable is present in both polarities and thus is not present on the path, a so-called "skipped" node. The nodes below this "skipped" node must be transformed as if the "skipped" node was present. It is often the case that "skipped" nodes occur in BDDs since this results from the application of the reduction rules. Were a BDD not constrained by these reduction rules, it would become an exponential Shannon tree, which is a binary tree representation of a function containing $2^n - 1$ non-terminal nodes and 2^n terminal nodes. The fewer the number of skipped nodes in a BDD, the more closely the BDD approaches an exponentially sized Shannon expansion tree.

Walsh

The 1×1 Walsh transformation matrix is shown in Figure 4.1. The Walsh transform is applied over the integer field.

$$W^1 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix}$$

Figure 4.1 Walsh Transform Matrix

Arithmetic

The 1×1 arithmetic transformation matrix is shown in Figure 4.2. The arithmetic transform is also applied over the integer field.

$$A^1 = \begin{bmatrix} +1 & 0 \\ -1 & +1 \end{bmatrix}$$

Figure 4.2 Arithmetic Transform Matrix

It is worth noting that SDDs describing the arithmetic transform of a function are in fact *Binary Moment Diagrams* (BMD) [5]. The inverse transform can also be easily implemented allowing for techniques to transform directly from BDDs to BMDs and vice versa. It is easy to see that BMDs result from the application of the arithmetic transform since examination of the 2×2 matrix in Figure 4.2 results in a pseudo-Boolean decomposition of the original function.

Reed-Muller

The Reed-Muller 1×1 transformation matrix is shown in Figure 4.3. The Reed-Muller transform is applied over $GF(2)$.

$$M^1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Figure 4.3 Reed-Muller Transform Matrix

A Reed-Muller SDD is in fact a *Functional Decision Diagram* (FDD) [13, 27] without the presence of complemented edges. The inverse transform can also be easily implemented which provides for a technique to transform directly from BDDs to FDDs and vice versa. The Reed-Muller transformation matrix in Figure 4.3 is actually a matrix representation of a positive-Davio decomposition [26].

Methodology

The pseudo-code in this section illustrates the implementation of the graph-based transformation algorithm for multi-output functions. This is the framework for the technique used for all three algorithms by including the appropriate terminal manipulations after the check for terminal nodes in the *else* and *then* branches of the pseudo-code. For each output the pointer to the top node for that output is passed to a traversal function that controls the transformation. The pseudo-code for this traversal is shown in Figure 4.4.

```
Traverse (f)
    if (f is a terminal node) return
    Low = Traverse (Low(f))
    High = Traverse (High(f))
    LowTemp = TransformLow (Low, High)
    HighTemp= TransformHigh (Low, High)
    return (NewNonterminal (Index(f), HighTemp, LowTemp))
```

Figure 4.4 Pseudo-Code for *Traverse*

The *TransformLow* and *TransformHigh* transformation functions are identical except for the action taken once the terminal nodes are reached. For illustration the code for TransformLow that transforms the *else*-child is shown in Figure 4.5. The

TransformLow function has four possible courses of action: 1) if both nodes are terminal nodes it performs the required manipulation and returns the new terminal node; 2) if both nodes are non-terminal nodes at the same level, a new node is formed with children found by transforming the corresponding children of the two original nodes; 3) if the level of the *else*-child is greater than the level of the *then*-child, a "skipped" node is present in the *else* branch and must be considered; and 4) if the level of the *then*-child is greater than the level of the *else*-child, a "skipped" node is present in the *then* branch and must be considered. Detail of the handling of skipped nodes is also given in the pseudo-code of Figure 4.5.

```
TransformLow (g, h)
    if (g and h are terminals)
        return (appropriate new terminal manipulation)
    else if (Level(g) = Level(h))
            return (NewNonterminal (Index(g), TransformLow (Low(g), Low(h)),
                    TransformLow (High(g), High(h))))
    else if(Level(g) > Level(h))
            return (NewNonterminal (Index(h), TransformLow (Twice(g), Low(h)),
                    High(h)))
    else (Level(g) < Level(h))
           return (NewNonterminal (Index(g), TransformLow (Low(g), Twice(h)),
                   High(g)))
```

Figure 4.5 Pseudo-Code for *TransformLow*

"Skipped" nodes within the diagram must be considered, and their children transformed as if they were present. As this allowance must be made when transforming both the *else*-child and the *then*-child, an improvement was made to the original algorithm providing for caching of the result of a "skipped" node when it is first encountered so that the result is available on the subsequent encounter. Caching the result of this computation resulted in significant decreases in the computation time required for transformation.

Walsh

For the Walsh transform the appropriate manipulation in the *TransformLow* function of the pseudo-code is given in Figure 4.6. The corresponding manipulation in the *TransformHigh* function is given in Figure 4.7.

```
return (NewTerminal (Value(g) + Value(h)))
```

Figure 4.6 Pseudo-Code for Walsh *TransformLow* Terminals

```
return (NewTerminal (Value(g) - Value(h)))
```

Figure 4.7 Pseudo-Code for Walsh *TransformHigh* Terminals

Arithmetic

The arithmetic *TransformLow* terminal manipulation is described in pseudo-code in Figure 4.8. The corresponding manipulation for *TransformHigh* is given in Figure 4.9.

```
return (NewTerminal (Value(g) + 0))
```

Figure 4.8 Pseudo-Code for Arithmetic *TransformLow* Terminals

```
return (NewTerminal (Value(h) - Value(g)))
```

Figure 4.9 Pseudo-Code for Arithmetic *TransformHigh* Terminals

Reed-Muller

The *TransformLow* manipulation for the Reed-Muller transform is described in pseudo-code in Figure 4.10. The corresponding *TransformHigh* manipulation is given in Figure 4.10. Although similar to the *TransformLow* terminal manipulation of the arithmetic transform and the *TransformHigh* terminal manipulation of the Walsh transform the Reed-Muller *TransformLow* and *TransformHigh* terminal manipulations are applied over GF(2).

```
return (NewTerminal (Value(g) + 0))
```

Figure 4.10 Pseudo-Code for Reed-Muller *TransformLow* Terminals

```
return (NewTerminal (Value(g) + Value(h)))
```

Figure 4.11 Pseudo-Code for Reed-Muller *TransformHigh* Terminals

Walsh Transformation Example

For illustration of the algorithm, a small multi-output circuit with two outputs represented an AND gate and an OR gate is shown as a BDD in Figure 4.12. The two functions are represented using *S*-encoding. In Figure 4.13 the multi-output circuit begins its transformation.

In Figure 4.14 *Traverse* has traversed the *else* branch of x_{1dd}, encountered the constant node +1 and returned to traverse the *then* branch of x_{1dd}. In this branch it encounters the *then*-child, x_{2dd}. As x_{2dd} has two children that are terminals, the recursive traversal will stop and the transformation of the node proceeds. First *TransformLow* is called and returns a value of 0. Next *TransformHigh* is called and returns a value of 1. At this point x_{2dd} is now transformed into x_{2sd}.

In Figure 4.15 *Traverse* has now returned to x_{1dd}. The recursive traversal is now complete for this node and its transformation proceeds. First *TransformLow* is called, however within this call one terminal node and one non-terminal node are present. Due to the manner in which the BDD package is implemented, terminal nodes have a greater level than non-terminal nodes, therefore the > code within the algorithm is chosen. The "skipped" node x_{2dd} must now be considered. If this node had been present, both of its children would have been terminal nodes with a value of +1. Therefore the transformed "skipped" node would have an *else*-child with a value of +2 (1+1) and a *then*-child with a value of 0 (1-1). The values of these children are then used to calculate the coefficients for x_{1dd} to complete the transformation. For the transformed "skipped" node the children returned from *TransformLow* again have the same value of +2 (2+0) and (0+2). For the *then*-child node, x_{2sd}, TransformHigh returns an *else*-child with a value of +2 (2-0) and a *then*-child with a value of -2 (0-+2). x_{1dd} has now been transformed into x_{1sd} and the transformation of the AND gate is complete.

Figure 4.15 shows the completed transformation of x_{2sd} for the OR output. *Traversal* has recursively descended to x_{2dd} and found both of its terminal children. TransformLow has returned a value of 0 (+1 + -1) for the *else*-child and TransformHigh has returned a value of +2 (+1- -1) for the *then*-child respectively.

In Figure 4.16 the transformation of the OR output is complete. *Traversal* has traversed the *then* branch of x_{1dd}, and discovered the terminal node. This time the < portion of the algorithm for *TransformLow* has been called. When the "skipped" *then*-child node is considered, *TransformLow* returns an *else*-child value of -2 (-1 + -1) and a *then*-child value of 0 (-1 - -1). Using these values, for the *else*-child node, x_{2sd} *TransformLow* returns an e*lse*-child value of -2 (0+-2) and a *then*-child value of +2 (+2+0). Transformation of the *then*-child completes the transformation and the children of the "skipped" node both are computed to be +2 ((0 - -2) and (+2 - 0)).

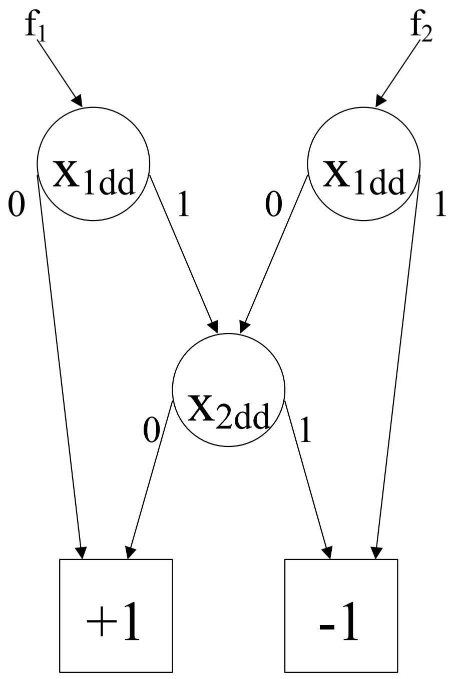


Figure 4.12 BDD Before Transformation

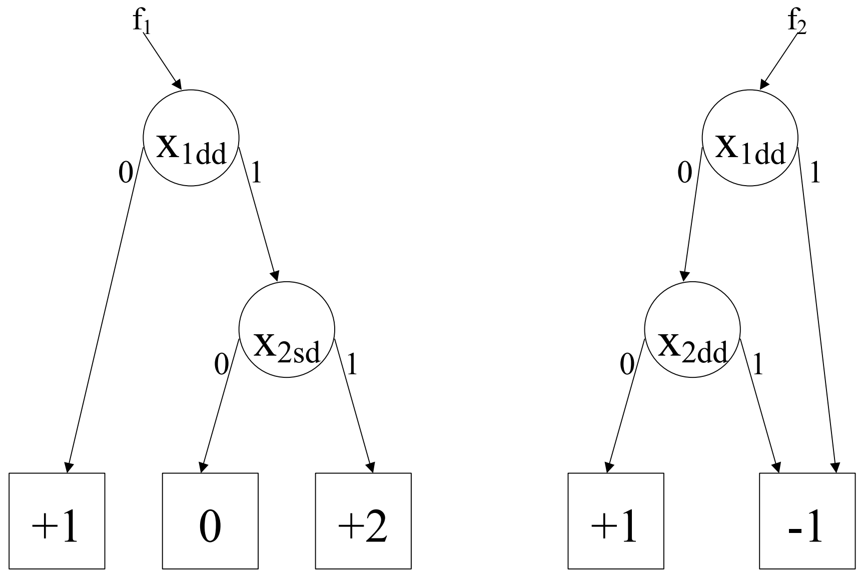


Figure 4.13 For f_1 x_{2dd} Is Transformed to x_{2sd}

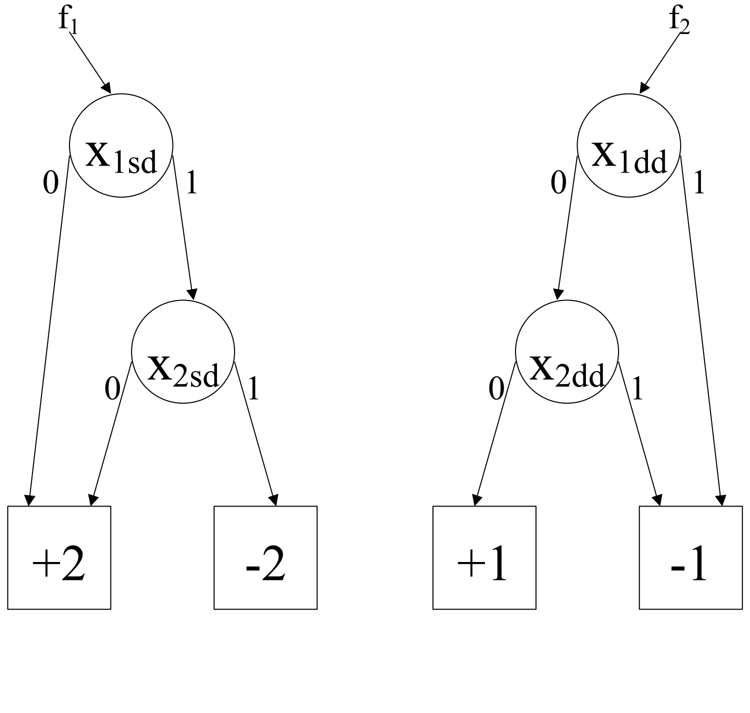


Figure 4.14 For f_1 x_{1dd} Is Transformed to x_{1sd}

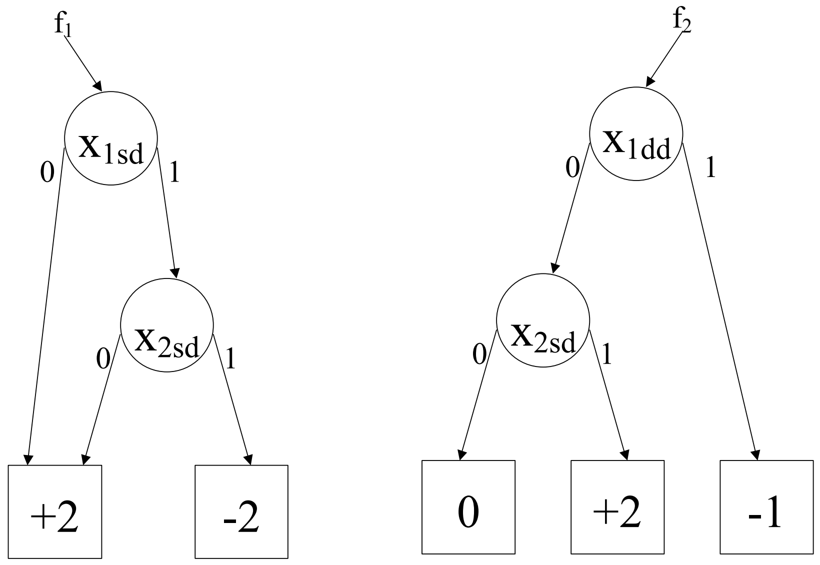


Figure 4.15 For f_2 x_{2dd} Is Transformed to x_{2sd}

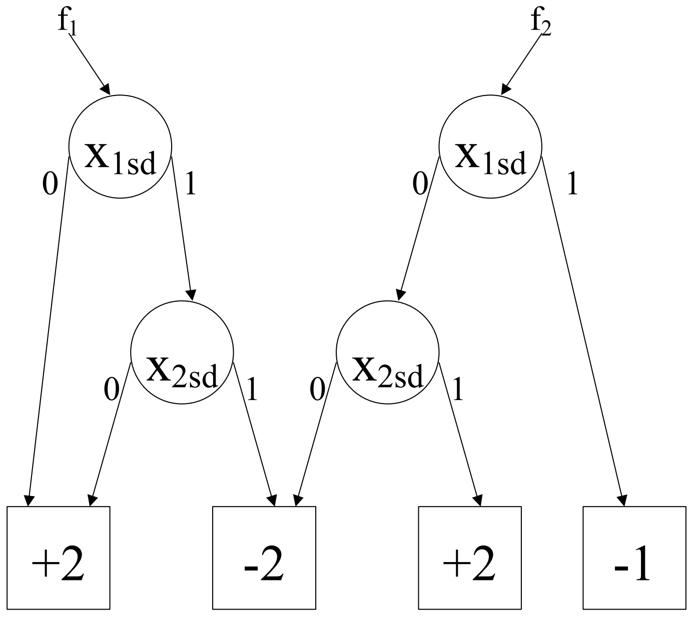


Figure 4.16 For f_2 x_{1dd} Is Transformed to x_{1sd}

Experimental Results

In this section experimental results are presented that have been carried out on a SUN Ultra 10. This code was implemented in conjunction with the *Colorado University Decision Diagram* (CUDD) package [38]. All runtimes are given in CPU seconds with a runtime limit set at 1 hour of CPU time.

Each transform was performed on over one hundred benchmark circuits. The circuits represented here were selected based upon the size of the Walsh SDD, restricting the set to diagrams that had over 5000 nodes as large enough to be of interest.

Walsh

Table 4.1 contains the number of nodes and the number of coefficients for each benchmark circuit undergoing the Walsh transformation. In Table 4.2 the runtimes are shown for the original multi-output algorithm and for the improved algorithm utilizing "skipped" node caching as well as the percentage of improvement achieved. Because CPU time was limited to one hour, the transformation of some circuits failed to complete. Approximately 10% of the total benchmark circuits attempted could not be completed within this time limitation.

Arithmetic

Table 4.3 contains the number of nodes and the number of coefficients for each benchmark circuit undergoing the arithmetic transform. In Table 4.4 runtimes are shown for the original multi-output algorithm and for the improved algorithm utilizing "skipped" node caching as well as the percentage of improvement achieved.

Table 4.1

Walsh Spectral Decision Diagrams

Circuit	Number of Inputs	Number of Outputs	Walsh	
			(SDD) Nodes	Coefficients
alu4	14	8	6554	171
apex5	117	88	7341	226
bc0	26	11	21026	2529
chkn	29	7	9772	863
cps	24	109	13388	790
duke2	22	29	6601	506
ex1010	10	10	6844	116
ex4	128	28	15112	384
frg2	143	139	6007	398
in2	19	10	6087	256
misex3	14	14	24527	386
tial	14	8	7685	171
vda	17	39	11302	935
x1	51	35	10809	1406

Table 4.2

Walsh Runtime Comparison

Circuit	Walsh		
	Original (sec)	Improved (sec)	% Change
alu4	0.94	0.83	11.7%
apex5	-	1900.35	-
bc0	32.52	15.61	52.0%
chkn	687.25	404.19	41.2%
cps	487.90	264.72	45.7%
duke2	15.18	9.42	37.9%
ex1010	0.20	0.19	5.0%
ex4	3.46	2.72	21.4%
frg2	1270.07	552.20	56.5%
in2	18.47	10.88	41.1%
misex3	2.85	2.27	20.4%
tial	1.07	0.92	14.0%
vda	9.21	4.57	15.4%
x1	465.79	274.27	41.1%

Table 4.3

Arithmetic Spectral Decision Diagrams

Circuit	Number of Inputs	Number of Outputs	Arithmetic	
			(BMD) Nodes	Coefficients
alu4	14	8	4652	39
apex5	117	88	5757	11
bc0	26	11	5019	11
chkn	29	7	2482	14
cps	24	109	3642	15
duke2	22	29	3057	11
ex1010	10	10	4604	50
ex4	128	28	3217	10
frg2	143	139	5209	13
in2	19	10	2822	22
misex3	14	14	3176	29
tial	14	8	3739	39
vda	17	39	4333	15
x1	51	35	2332	5

Table 4.4

Arithmetic Runtime Comparison

Circuit	Arithmetic		
	Original (sec)	**Improved (sec)**	**% Change**
alu4	0.80	0.68	15.0%
apex5	-	1690.51	-
bc0	28.98	13.54	53.3%
chkn	620.68	366.41	41.0%
cps	441.88	236.28	46.5%
duke2	13.35	8.78	34.2%
ex1010	0.17	0.17	0.0%
ex4	2.93	2.30	21.5%
frg2	1154.11	498.62	56.8%
in2	16.76	10.40	37.9%
misex3	2.60	1.94	25.4%
tial	0.96	0.82	14.6%
vda	8.34	4.30	48.4%
x1	434.46	256.90	40.9%

Table 4.5

Reed-Muller Spectral Decision Diagrams

Circuit	Number of Inputs	Number of Outputs	Reed-Muller (FDD) Nodes
alu4	14	8	1266
apex5	117	88	2777
bc0	26	11	3000
chkn	29	7	761
cps	24	109	1708
duke2	22	29	1236
ex1010	10	10	1520
ex4	128	28	1137
frg2	143	139	2272
in2	19	10	958
misex3	14	14	1140
tial	14	8	953
vda	17	39	2117
x1	51	35	1040

Table 4.6

Reed-Muller Runtime Comparison

Circuit	**Reed-Muller**		
	Original (sec)	**Improved (sec)**	**% Change**
alu4	0.71	0.59	16.9%
apex5	-	1561.31	-
bc0	27.26	12.42	54.4%
chkn	578.86	349.07	39.7%
cps	409.60	220.42	46.2%
duke2	12.57	8.04	36.0%
ex1010	0.15	0.14	6.7%
ex4	2.69	2.16	19.7%
frg2	1111.28	445.46	59.9%
in2	15.31	9.47	38.1%
misex3	2.36	1.73	26.7%
tial	0.86	0.73	15.1%
vda	7.92	3.90	50.8%
x1	408.72	246.08	39.8%

Reed-Muller

Table 4.5 contains the number of nodes for each benchmark circuit undergoing the Reed-Muller transformation. In Table 4.6 runtimes are shown for the original multi-output algorithm and for the improved algorithm utilizing "skipped" node caching as well as the percentage of improvement achieved.

Concluding Comments

An algorithm for computing the spectral coefficients for the Walsh, the arithmetic, and the Reed-Muller transforms of a multi-output circuit represented as a BDD has been presented. The results from the implementation of transformation algorithms with regard to the number of nodes and the number of coefficients in the resulting SDD has also been presented. An improvement to the algorithm that decreases the runtime by caching the results obtained when considering "skipped" nodes has been implemented and the comparison between the runtimes of these two methods has been shown.

The approach described here makes it practical to computer the spectra of large multi-output problems and extends the possibility of applying spectral techniques. The approach could be readily extended to the Haar spectral domain and also to transformation among the various spectral domains as described by M. A. Thornton, R. Drechsler, and D. M. Miller in [43].

CHAPTER V

CONCLUSIONS

This work has examined the representation of discrete functions in both the Boolean and the spectral domains. It has described some of the limitations of BDD representations as motivation for developing alternative representations in both the Boolean domain and the spectral domain. In this chapter the results of this work are summarized and opportunities for future research are discussed.

Summary

This section provides a summary of each of the primary sections of this work: binary decision diagrams, spectral computations, and spectral decision diagrams.

Binary Decision Diagrams

Chapter II began with a review of BDD properties and the construction of BDDs. Two possible reasons why a BDD might reach exponential size were discussed as motivation for representing a BDD as multiple pBDDs. A technique for partitioning a BDD into multiple pBDDs each representing a subset of the information in the complete BDD was described. This method redirects a portion of the functionality of the circuit to a third terminal node, U. Finally experimental results were shown for several circuits represented as multiple pBDDs.

Spectral Computations

Chapter III began by considering the traditional linear algebra-based method for computing the spectrum of a function. It then reviewed a more recent graph-based method based upon the concepts of algebraic groups and Cayley graphs. Several extensions to the graph-based method were then presented, including a method to directly calculate *S*-encoded coefficients and the consideration of other possible binary operators. A new Cayley group whose graph represents the spectrum for the inverse of a function was identified which uses equivalence as its binary operator. Finally a "fast" transformation matrix for calculating the adjacency matrix via transposition of the output vector for a function was presented.

Spectral Decision Diagrams

Chapter IV began by reviewing the three transforms to be implemented in this section: the Walsh transform, the arithmetic transform, and the Reed-Muller transform. Next the methodology used in the implementation of graph-based algorithms to compute the transformation for multi-output circuits was described. Experimental results for each of the transforms were presented. In addition an improved method of caching "skipped" nodes encountered during transformation was implemented for each transform and those results were compared with the results from the previous implementations.

Future Research

This section describes possible future research in each of the primary sections of this work: binary decision diagrams, spectral computations, and spectral decision diagrams. As the work presented within this thesis is a part of a larger project investigating statistical equivalence checking, one primary direction of the future research will be to continue to integrate this work into the larger project.

Binary Decision Diagrams

Future research effort could be directed towards improving the modifications to the ITE algorithm to decrease the overlap present among the pBDDs. The optimal solution would be to have a set of small disjoint pBDDs that together completely represent the functionality of the circuit. An additional area of research would be to prevent those occasions in which the altered ITE algorithm code actually generates a pBDD that is larger than the original complete BDD.

Spectral Computations

There has already been more research effort expended in the area of graph-based spectrum computations. M. A. Thornton has extended this work into multiple-valued Chrestenson spectrum computation [42]. One area of future effort would be to implement the use of decision diagrams to represent the adjacency matrix of a Cayley graph.

Spectral Decision Diagrams

There are many future research efforts possible in the area of spectral decision diagrams. It is likely that further improvements in the implementation of the algorithms presented here is possible due to the complexity of the CUDD program. Therefore a highly optimized version of this implementation is one future research area. Another area of interest is in the implementation of the multi-resolution Haar transform using the algorithm as described in [44] and then extending that algorithm and implementation to multi-output circuits. This transform is used in the larger project of which this thesis is a part due to its multi-resolution nature. Successful implementation of the Walsh transform was a first step towards the implementation of the Haar, which uses the same transformation matrix applied in a more complex manner. Yet another broad area of interest is in the transformation directly from one transform to another without returning to the Boolean domain as described by M. A. Thornton, R. Drechsler, and D. M. Miller in [43]. In addition to the implementations for the calculation of the spectra, there is a wealth of possibilities in exploring their applications to synthesis, testing, and verification through the use of spectral decision diagrams.

REFERENCES

[1] A. Bernasconi, B. Codenotti, and J. M. Vanderkam, "A characterization of bent functions in terms of strongly regular graphs", *IEEE Transactions on Computers*, vol. 50, pp. 984-985, September 2001.

[2] A. Bernasconi and B. Codenotti, "Spectral analysis of Boolean functions as a graph eigenvalue problem", *IEEE Transactions on Computers*, vol. 48, pp. 345-351, March 1999.

[3] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete", *IEEE Transactions on Computers*, vol. 45, pp. 993-1002, September 1996.

[4] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package", in *Design Automation Conference*, Orlando FL, pp. 40-45, June 1990.

[5] R. E. Bryant and Y.-A. Chen, "Verification of arithmetic circuits with binary moment diagrams", in *Design Automation Conference*, San Francisco CA, pp. 535-541, June 1995.

[6] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication", *IEEE Transactions on Computers*, vol. 40, pp. 205-213, February 1991.

[7] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation", *IEEE Transactions on Computers*, vol. 35, pp. 677-691, August 1986.

[8] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping", in *Design Automation Conference*, Dallas TX, pp. 54-60, June 1993.

[9] D. M. Cvetković, M. Doob, and H. Sachs, *Spectra of Graphs*. Academic Press, 1979.

[10] T. Damarla, "Generalized transforms for multiple valued circuits and their fault detection", *IEEE Transactions on Computers*, vol. 41, pp. 1101-1109, September 1992.

[11] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York, NY: McGraw-Hill, 1994.

[12] R. Drechsler and B. Becker, *Binary Decision Diagrams - Theory and Implementation*. Kluwer Academic Publishers, 1998.

[13] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski, "Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams", in *Design Automation Conference*, San Diego CA, pp. 415-419, June 1994.

[14] C. R. Edwards, "The design of easily tested circuits using mapping and spectral techniques", *Radio and Electronic Engineer*, vol. 47, pp. 321-342, July 1977.

[15] C. R. Edwards, "The application of the Rademacher-Walsh transform to Boolean function classification and threshold logic synthesis", *IEEE Transactions on Computers*, vol. 24, pp. 48-62, January 1975.

[16] B. J. Falkowski, "Relationship between arithmetic and Haar wavelet transforms in the form of layered Kronecker matrices", *Electronics Letters*, vol. 35, pp. 799-800, May 13, 1999.

[17] B. J. Falkowski, "Forward and inverse transformations between Haar wavelet and arithmetic functions", *Electronics Letters*, vol. 34, pp. 1084-1085, May 28, 1998.

[18] B. J. Falkowski and C.-H. Chang, "Forward and inverse transformations between Haar spectra and ordered binary decision diagrams of Boolean functions", *IEEE Transactions on Computers*, vol. 46, pp. 1272-1279, November 1997.

[19] S. J. Friedman and K. J. Supowit, "Finding the optimal variable ordering for binary decision diagrams", in *Design Automation Conference*, Miami Beach FL, pp. 348-356, June 1987.

[20] A. Graham, *Kronecker Products and Matrix Calculus: with Applications*. Ellis Horwood Limited and John Wiley & Sons, 1981.

[21] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Norwell, MA: Kluwer Academic Publishers, 1996.

[22] J. P. Hansen and M. Sekine, "Synthesis by spectral translation using Boolean decision diagrams", in *Design Automation Conference*, Las Vegas NV, pp 248-253, June 1996.

[23] T. C. Hsiao and S. C. Seth, "An analysis of the use of Rademacher-Walsh spectrum in compact testing", *IEEE Transactions on Computers*, vol. 33, pp. 931-937, October 1984.

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

[25] J. Jain, D. Moundanos, J. Bitner, J. A. Abraham, D. S. Fussell, and D. E. Ross, "Efficient variable ordering and partial representation algorithm", in *Proceedings of the 8th International Conference on VLSI Design*, New Delhi India, pp. 81-86, January 1995.

[26] M. Karpovsky, *Finite Orthogonal Series in the Design of Digital Devices*. Wiley and JUP, 1976.

[27] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams", in *Proceedings of the European Conference on Design Automation*, Brussels Belgium, pp. 43-47, March 1992.

[28] W. Kunz and D. K. Pradhan, "Recursive learning: a new implication technique for efficient solutions to CAD problems-test, verification, and optimization", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, pp. 1143-1158, September 1994.

[29] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design*. Berlin, Germany: Springer, 1998.

[30] D. M. Miller, "A spectral method for Boolean function matching", in *European Design and Test Conference*, Paris France, pg. 602, March 1996.

[31] D. M. Miller, "Graph algorithms for the manipulation of Boolean functions and their spectra", in *Congressus Numerantium*, pp. 177-199, 1987.

[32] D. M. Miller and J. C. Muzio, "Spectral fault signatures for single stuck-at faults in combinational networks", *IEEE Transactions on Computers*, vol. 33, pp. 765-768, August 1984.

[33] A. Narayan, J. Jain, M. Fujita, and A. Sangiovanni-Vincentelli, "Partitioned ROBDDs-a compact, canonical and efficiently manipulable representation for Boolean functions", in *International Conference on Computer-Aided Design*, San Jose CA, pp. 547-554, November 1996.

[34] M. A. Perkowski, M. Driscoll, J. Liu, D. Smith, J. Brown, L. Yang, A. Shamsapour, M. Helliwell, B. Falkowski, P. Wu, M. Ciesielski, and A. Sarabi, "Integration of logic synthesis and high-level synthesis into the DIADES design automation system", in *International Symposium on Circuits and Systems*, Portland OR, vol. 2, pp. 748-751, May 1989.

[35] K. Radecka and Z. Zilic, "Relating arithmetic and Walsh spectra for verification by implicit error modeling", in *International Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Mississippi State MS, pp. 205-214, August 2001.

[36] D. E. Ross, K. M. Butler, R. Kapur, and M. R. Mercer, "Fast functional evaluation of candidate OBDD variable orderings", in *Proceedings of the European Confernce on Design Automation*, Amsterdam Netherlands, pp 4-10, February 1991.

[37] T. Sasao, *Switching Theory for Logic Synthesis*. Norwell, MA: Kluwer Academic Publishers, 1999.

[38] F. Somenzi, CUDD: CU Decision Diagram Package Release 2.3.0. University of Colorado at Boulder, 1998.

[39] R. S. Stanković, "A note of the relation between Reed-Muller expansions and Walsh transforms", *IEEE Transactions on Electromagnetic Compatability*, vol. 24, pp. 68-70, February 1982.

[40] R. S. Stanković, T. Sasao, and C. Moraga, "Spectral transforms decision diagrams", in T. Sasao and M. Fujita (eds.), *Representation of Discrete Functions*. Kluwer Academic Publishers, pp. 55-92, 1996.

[41] A. K. Susskind, "Testing by verifying Walsh coefficients", *IEEE Transactions on Computers*, vol. 32, pp. 198-201, February 1983.

[42] M. A. Thornton, D. M. Miller, and W. J. Townsend, "Chrestenson spectrum computation using Cayley color graphs", submitted for publication in *International Symposium on Multiple-Valued Logic*, Boston MA, May 2002.

[43] M. A. Thornton, R. Drechsler, and D. M. Miller, "Transformations amongst the Walsh, Haar, arithmetic, and Reed-Muller spectral domains", in *International Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Mississippi State MS, pp. 215-225, August 2001.

[44] M. A. Thornton, R. Drechsler, and D. M. Miller, *Spectral Techniques in VLSI CAD*. Norwell, MA: Kluwer Academic Publishers, 2001.

[45] M. A. Thornton and R. Drechsler, "Spectral decision diagrams using graph transformations", in *Design, Automation and Test in Europe*, Munich Germany, pp. 713-717, March 2001.

[46] M. A. Thornton, R. Drechsler, and W. Günther, "A method for approximate equivalence checking", in *Proceedings of the 30th IEEE International Symposium on Multiple-Valued Logic*, Portland OR, pp 447-452, May 2000.

[47] M. A. Thornton and V. S. S. Nair, "Efficient calculation of spectral coefficients and their applications", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 1328-1341, November 1995.

[48] W. J. Townsend, M. A. Thornton, R. Drechsler, and D. M. Miller, "Computing Walsh, arithmetic, and Reed-Muller spectral decision diagrams using graph transformations", submitted for publication in *Great Lakes Symposium on VLSI*, New York NY, April 2002.

[49] W. J. Townsend and M. A. Thornton, "Partial binary decision diagrams", in *IEEE Southeastern Symposium on System Theory*, Huntsville AL, pp. 422-425, March 2002.

[50] W. J. Townsend and M. A. Thornton, "Walsh spectrum computations using Cayley graphs", in *IEEE Midwest Symposium on Circuits and Systems*, Dayton OH, pp. 110-113, August 2001.

[51] D. Varma and E. A. Trachtenberg, "Design automation tools for efficient implementation of logic functions by decomposition", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, pp. 901-916, August 1989.

[52] A. T. White, *Graphs, Groups and Surfaces*. North-Holland Publishing Company, 1973.