

1991

Spectral Methods for Boolean and Multiple-Valued Input Logic Functions

Bogdan Jaroslaw Falkowski
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Let us know how access to this document benefits you.

Recommended Citation

Falkowski, Bogdan Jaroslaw, "Spectral Methods for Boolean and Multiple-Valued Input Logic Functions" (1991). *Dissertations and Theses*. Paper 1152.
<https://doi.org/10.15760/etd.1151>

This Dissertation is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

SPECTRAL METHODS FOR BOOLEAN AND MULTIPLE-VALUED
INPUT LOGIC FUNCTIONS

by

BOGDAN JAROSŁAW FALKOWSKI

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY
in
ELECTRICAL AND COMPUTER ENGINEERING

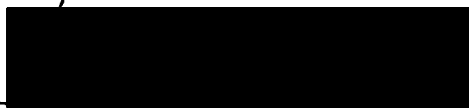
Portland State University
1991

AN ABSTRACT OF THE DISSERTATION OF Bogdan Jarosław Falkowski for the
Doctor of Philosophy in Electrical and Computer Engineering presented May 2, 1991.

Title: Spectral Methods for Boolean and Multiple-Valued Input Logic Functions.

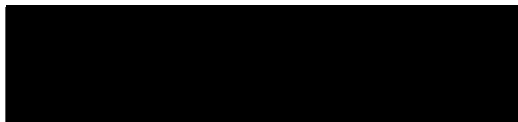
APPROVED BY THE MEMBERS OF THE DISSERTATION COMMITTEE:


Marek A. Perkowski, Chair


Lee W. Casperson


Małgorzata B. Chrzanowska-Jeske


Jack C. Riley


Maria E. Balogh

Spectral techniques in digital logic design have been known for more than thirty years. They have been used for Boolean function classification, disjoint decomposition, parallel and serial linear decomposition, spectral translation synthesis (extraction of

linear pre- and post-filters), multiplexer synthesis, prime implicant extraction by spectral summation, threshold logic synthesis, estimation of logic complexity, testing, and state assignment.

This dissertation resolves many important issues concerning the efficient application of spectral methods used in the computer-aided design of digital circuits. The main obstacles in these applications were, up to now, memory requirements for computer systems and lack of the possibility of calculating spectra directly from Boolean equations. By using the algorithms presented here these obstacles have been overcome. Moreover, the methods presented in this dissertation can be regarded as representatives of a whole family of methods and the approach presented can be easily adapted to other orthogonal transforms used in digital logic design. Algorithms are shown for Adding, Arithmetic, and Reed-Muller transforms. However, the main focus of this dissertation is on the efficient computer calculation of Rademacher-Walsh spectra of Boolean functions, since this particular ordering of Walsh transforms is most frequently used in digital logic design.

A theory has been developed to calculate the Rademacher-Walsh transform from a cube array specification of incompletely specified Boolean functions. The importance of representing Boolean functions as arrays of disjoint ON- and DC- cubes has been pointed out, and an efficient new algorithm to generate disjoint cubes from non-disjoint ones has been designed. The transform algorithm makes use of the properties of an array of disjoint cubes and allows the determination of the spectral coefficients in an independent way. By such an approach each spectral coefficient can be calculated separately or all the coefficients can be calculated in parallel. These advantages are absent in the existing methods. The possibility of calculating only some coefficients is very important since there are many spectral methods in digital logic design for which the values of only a few selected coefficients are needed.

Most of the current methods used in the spectral domain deal only with completely specified Boolean functions. On the other hand, all of the algorithms introduced here are valid, not only for completely specified Boolean functions, but for functions with don't cares. Don't care minterms are simply represented in the form of disjoint cubes.

The links between spectral and classical methods used for designing digital circuits are described. The real meaning of spectral coefficients from Walsh and other orthogonal spectra in classical logic terms is shown. The relations presented here can be used for the calculation of different transforms. The methods are based on direct manipulations on Karnaugh maps. The conversion starts with Karnaugh maps and generate the spectral coefficients.

The spectral representation of multiple-valued input binary functions is proposed here for the first time. Such a representation is composed of a vector of Walsh transforms, each vector is defined for one pair of the input variables of the function. The new representation has the advantage of being real-valued, thus having an easy interpretation. Since two types of codings of values of binary functions are used, two different spectra are introduced. The meaning of each spectral coefficient in classical logic terms is discussed. The mathematical relationships between the number of true, false, and don't care minterms and spectral coefficients are stated. These relationships can be used to calculate the spectral coefficients directly from the graphical representations of binary functions. Similarly to the spectral methods in classical logic design, the new spectral representation of binary functions can find applications in many problems of analysis, synthesis, and testing of circuits described by such functions.

A new algorithm is shown that converts the disjoint cube representation of Boolean functions into fixed-polarity Generalized Reed-Muller Expansions (GRME). Since the known fast algorithm that generates the GRME, based on the factorization of

the Reed-Muller transform matrix, always starts from the truth table (minterms) of a Boolean function, then the described method has advantages due to a smaller required computer memory. Moreover, for Boolean functions, described by only a few disjoint cubes, the method is much more efficient than the fast algorithm.

By investigating a family of elementary second order matrices, new transforms of real vectors are introduced. When used for Boolean function transformations, these transforms are one-to-one mappings in a binary or ternary vector space. The concept of different polarities of the Arithmetic and Adding transforms has been introduced. New operations on matrices : horizontal, vertical, and vertical-horizontal joints (concatenations) are introduced.

All previously known transforms, and those introduced in this dissertation can be characterized by two features: "ordering" and "polarity". When a transform exists for all possible polarities then it is said to be "generalized". For all of the transforms discussed, procedures are given for generalizing and defining for different orderings. The meaning of each spectral coefficient for a given transform is also presented in terms of standard logic gates.

There exist six commonly used orderings of Walsh transforms: Hadamard, Rademacher, Kaczmarz, Paley, Cal-Sal, and X. By investigating the ways in which these known orderings are generated the author noticed that the same operations can be used to create some new orderings. The generation of two new Walsh transforms in Gray code orderings, from the straight binary code is shown. A recursive algorithm for the Gray code ordered Walsh transform is based on the new operator introduced in this presentation under the name of the "bi-symmetrical pseudo Kronecker product". The recursive algorithm is the basis for the flow diagram of a constant geometry fast Walsh transform in Gray code ordering. The algorithm is fast ($N \log_2 N$ additions/subtractions), computer efficient, and is implemented in an iterative architecture.

TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the dissertation of Bogdan Jarosław Falkowski presented May 2, 1991.


Marek A. Perkowski, Chair

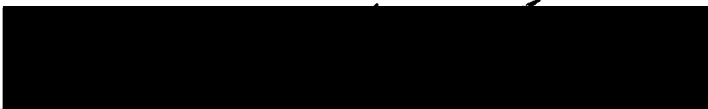

Lee W. Casperson



Małgorzata K. Chrzanowska-Jeske


Jack C. Riley


Maria E. Balogh

APPROVED:


H. Chik M. Erzurumlu, Dean, School of Engineering and Applied Science


C. William Savery, Vice Provost for Graduate Studies and Research

ACKNOWLEDGEMENTS

I wish especially to thank my mother Danuta Zofia Falkowska for all the help and financial support that have allowed me to complete this research.

I am very grateful to Prof. Marek Perkowski for his guidance and inspiration during my research at Portland State University. I would like to express my gratitude to the Final Oral Examination committee members: Prof. Lee W. Casperson, Prof. Małgorzata Chrzanowska-Jeske, Prof. Jack Riley, and Prof. Maria Balogh. In particular, I would like to thank Prof. Jack Riley for numerous helpful comments.

I would also like to thank Ingo Schäfer for implementing many of the presented algorithms in C code. Special thanks and gratitude to my "second" mother Ms. Shirley Clark for her constant support and making Portland State a home for me.

TABLE OF CONTENTS

		PAGE
	ACKNOWLEDGEMENTS	iii
	LIST OF TABLES	viii
	LIST OF FIGURES	ix
CHAPTER		
I	INTRODUCTION	1
II	PROPERTIES OF DISCRETE WALSH TRANSFORMS	11
	II.1 Relations between Discrete Fourier and Walsh Transforms	11
	II.2 Walsh Functions	13
III	ESSENTIAL RELATIONS BETWEEN CLASSICAL AND SPECTRAL APPROACHES TO ANALYSIS, SYNTHESIS, AND TESTING OF COMPLETELY AND INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS	20
	III.1 Description	20
	III.2 Basic Properties of Rademacher-Walsh Spectrum	21
	III.3 Links between Spectral and Classical Logic Design	27

IV	WALSH TYPE TRANSFORMS FOR COMPLETELY AND INCOMPLETELY SPECIFIED MULTIPLE-VALUED INPUT BINARY FUNCTIONS	37
	IV.1 Description	37
	IV.2 Multiple-Valued Input Binary Functions	39
	IV.3 Definitions and Basic Properties of Two-Dimensional Maps for Multiple-Valued Input Binary Functions	40
	IV.4 Basic Properties of Hadamard-Walsh Spectra S and R for Two-Dimensional Maps Representing Multiple-Valued Input Binary Functions	46
	IV.5 Links between Spectral and Classical Logic Design	53
V	EFFECTIVE COMPUTER METHODS FOR THE CALCULATION OF THE RADEMACHER-WALSH SPECTRUM FOR COMPLETELY AND INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS	63
	V.1 Description	63
	V.2 Algorithm to Generate Disjoint Cubes	64
	V.3 Application of an Array Method for the Calculation of Spectral Coefficients of Completely and Incompletely Specified Boolean Functions	72
	V.4 Calculation of Spectral Coefficients for Systems of Completely and Incompletely Specified Boolean Functions	81

VI	A FAMILY OF ALL ESSENTIAL RADIX-2 ADDITION AND SUBTRACTION MULTI-POLARITY TRANSFORMS WITH ALGORITHMS AND INTERPRETATIONS IN THE BOOLEAN DOMAIN	87
	VI.1 Description	87
	VI.2 Definitions of Essential Radix-2 Matrices	89
	VI.3 Generalized Arithmetic and Adding Transforms	92
	VI.4 Links of Arithmetic and Adding Transforms with Classical Logic Design	100
VII	THE CALCULATION OF GENERALIZED REED-MULLER CANONICAL EXPANSIONS FROM DISJOINT CUBE REPRESENTATIONS OF BOOLEAN FUNCTIONS	106
	VII.1 Description	106
	VII.2 Basic Definitions and Properties	107
	VII.3 Algorithm to Generate GRME from Disjoint Cubes	112
VIII	GENERALIZATIONS, ORDERINGS AND CIRCUIT INTERPRETATIONS OF SPECTRAL TRANSFORMS.	118
	VIII.1 Description	118
	VIII.2 Generalized Walsh Transform	119
	VIII.3 Different Orderings of Arithmetic, Adding and Reed-Muller Transforms	121
	VIII.4 Circuit Interpretations of Different Spectral Coefficients	123

IX	ALGORITHM AND ARCHITECTURE FOR GRAY CODE	
	ORDERED FAST WALSH TRANSFORM	125
	IX.1 Description	125
	IX.2 Gray Code Ordered Walsh Transforms:	
	Relationships and Methods of	
	Generation	128
	IX.3 Fast Walsh Transform for Gray Code	
	Ordered Transform	131
	IX.4 Constant Geometry Iterative Architecture for	
	a Gray Code Ordered Transform	136
X	CONCLUSION	139
	REFERENCES	146

LIST OF TABLES

TABLE		PAGE
I	Complete Rademacher-Walsh Spectrum S	81
II	Spectrum S Tot-ON of a System of Boolean Functions	85
III	Spectrum S Tot-DC of a System of Boolean Functions	86
IV	Adjustment Operator between Cube and Polarity.	110
V	Exact Matching Operator between Cube and Piterm.	111
VI	Cube Adjustment to Polarities 0000 and 0101	114
VII	Generation of Expanded Set of Piterms for Polarity 0000	114
VIII	Generation of Expanded Set of Piterms for Polarity 0101.	115
IX	Reed-Muller Spectral Coefficients for Polarity 0000	115
X	Reed-Muller Spectral Coefficients for Polarity 0101.	116

LIST OF FIGURES

FIGURE		PAGE
1.	Kaczmarz Ordered Walsh Functions	14
2.	Values of the Kaczmarz Ordered Walsh Transform	15
3.	Hadamard Ordered Walsh Functions	17
4.	Hadamard Ordered Discrete Walsh Transform	18
5.	Calculation of Rademacher-Walsh S Spectrum of Completely Specified Function	26
6.	Calculation of Rademacher-Walsh S Spectrum of Incompletely Specified Function	27
7.	Standard Trivial Functions for Completely Specified Boolean Function	33
8.	Standard Trivial Functions for Incompletely Specified Boolean Function	35
9.	Set of Two-Dimensional Maps for 3-Variable Binary Function	45
10.	Spectrum R for Completely Specified Map of Z and Y Variables	48
11.	Spectrum S for Completely Specified Map of Z and Y Variables	51
12.	Spectrum R for Incompletely Specified Map of X and Y Variables	52

13.	Spectrum S for Incompletely Specified Map of X and Y Variables	53
14.	Standard Trivial Functions for Completely Specified Two-Dimensional Map	57
15.	Standard Trivial Functions for Incompletely Specified Two-Dimensional Map	59
16.	Algorithm Generating Disjoint Cube Representation in Stages	73
17.	Incompletely Specified Boolean Functions F[1] and F[2]	84
18.	Nine Different Matrices of Order 2×1	90
19.	First 45 Possible Matrices with the Four Basic Types Marked	91
20.	Calculation of Arithmetic Transform for a Completely Specified Function	94
21.	Calculation of Arithmetic and Adding Transforms	95
22.	Calculation of Arithmetic and Adding Transforms for an Incompletely Specified Function	97
23.	Calculation of Arithmetic Transform for a Function for Polarity 0011	99
24.	Calculation of an Inverse Arithmetic Transform for a Function for Polarity 0011	100
25.	Standard Trivial Functions for Polarity 0000	103
26.	Standard Trivial Functions for Polarity 0011	105
27.	Calculation of Generalized Rademacher-Walsh Spectrum	120

28.	Calculation of Arithmetic Spectrum in Rademacher Ordering	122
29.	Relationships among 3 Orderings of Walsh Functions . . .	129
30.	Comparison of Order and Sequency for all Walsh Orderings	130
31.	Matrices for Two Walsh Gray Ordered Transforms	131
32.	Transformations of the Matrix A	132
33.	Kronecker and Pseudo Kronecker Product of Two Matrices	134
34.	Relationships between Natural and Gray Code Orders . . .	134
35.	a)Principle of Generation of Matrix F, b)Factorization for N=8	135
36.	Flow Diagram for the Recursive Gray Code Transform . . .	136
37.	An Iterative Block of the Fast Walsh Transform in Gray Code Ordering	137

CHAPTER I

INTRODUCTION

Computer aided design tools have been used for both the design and verification of digital systems for many years. Such tools have been applied to all the stages of the design of such systems, from integrated circuit technology to the design of complex computer architectures. The growing complexity of recently introduced Very Large Integrated Circuits (VLSI) has made the use of such sophisticated design tools indispensable.

A complete synthesis system should generate a layout from the high-level description of a VLSI system. The target technology, the design constraints and the cost functions should also be defined. Silicon compilers have been designed to carry out the entire synthesis process [1]. The approach used is to break the synthesis process into stages and to optimize the chip area and performance stage by stage [2].

The following basic components make parts of a synthesis system:

- Layout including floor planning, partitioning, placement, routing, and compaction.
- Logic synthesis, including combinational and sequential logic.
- Procedural design and module generation.

Since this dissertation concentrates on only one aspect of the synthesis process, combinational logic, then the developments in this area will be described in more detail.

Over the last few years a lot of attention has been paid to the logic minimization of two-level circuits. A series of very efficient logic minimizers have been developed:

ESPRESSO-II, ESPRESSO-IIC, and ESPRESSO-MV [3], [4] and [5]. When a logic function is implemented using a Programmable Logic Array (PLA), logic minimization reduces the area occupied by the PLA and improves its electrical performance. Since the algorithmic complexity of complete logic minimization problem is very high, heuristic logic minimizers are used when medium and, in particular, large functions have to be minimized.

It has been found, however, that some control logic can have more efficient implementation in the form of multi-level circuits [2]. The optimization criterion of all multi-level logic synthesis systems is to minimize the area occupied by the logic equations and at the same time satisfy the timing constraints. Other considerations, such as testability, should also be considered in the synthesis process. The first multi-level synthesis system was the IBM Logic Synthesis System [6], [7]. Other systems are: Yorktown Silicon Compiler [8], SOCRATES [9], and MIS [10].

Two basic techniques are used for multi-level design [2]:

- Global optimization, where the logic function is re-factored into an optimal multi-level form.
- Peephole optimization, where local transformations are applied to globally optimized logic functions.

New global optimization algorithms have appeared to be effective in partitioning complex logic functions [11]. In addition, rule based systems [7], [9] have been found to be effective in the design of large systems.

Logic synthesis for recently developed field-programmable gate arrays and PLDs creates new requirements for design automation systems because of fundamental architectural differences with respect to existing technologies. A high demand exists for the methods that produce circuit realizations with EXOR gates [12], [13]. "A four-input XOR (in Xilinx 2000 Logic Cell Arrays) uses the same space and is as fast as a four-

input AND gate." - quoted from [12]. Logic design for Xilinx devices is therefore limited by fan-in - not by logic complexity as in PLDs." - quoted from [12]. "Any system which flattens functions into 2-level AND-OR form, or which factors based on the "unate paradigm" (as do MIS-II, BOLD, and Synopsys), is going to have problems with strongly non-unate functions like parity, addition or multiplication. Since these sorts of functions occur frequently in real designs, synthesis tools need reasonable ways of handling them." - quoted from [13].

The DIADES design automation system [14] methodology is oriented towards detecting the linear (EXOR) part of a Boolean function. It uses, among other methods, spectral ones to detect the EXOR part of a Boolean function. Since the DIADES system uses spectral methods together with the programs based on the "unate paradigm" then it can easily handle not only functions close to strongly unate but also strongly not-unate ones as well. The decomposition of Boolean functions with both pre- and post- linear parts by spectral means leads to highly testable circuit realizations that can be efficiently implemented in several technologies (LHS501 from Signetics, 2000 Logic Cell Arrays from Xilinx, and other EPLDs with EXOR gates). Currently only spectral methods allow for this kind of decomposition [15], [16].

This Dissertation covers the fundamental background information on orthogonal transforms and their use in generating spectral data of logical functions. The conventional binary data, exemplified by Boolean expressions, or truth tables, may be transformed into the spectral domain, yielding coefficients which may represent some global information on the function or functions being considered. This is in contrast to the conventional functional domain, where the information concerning the overall function or functions is available from an isolated item of data.

In digital logic design, spectral techniques have been used for more than thirty years. They have been applied to Boolean function classification [17], [21], [22], [34]

and [35], disjoint decomposition [22], [47]-[49] and [51], parallel and serial linear decomposition [15], [16], [21]-[23], [48] and [49], spectral translation synthesis (extraction of linear pre- and post-filters) [16], [22]-[25], [27], [48], and [49], multiplexer synthesis [22] and [29], prime implicant extraction by spectral summation [22], [25], [27] and [28], threshold logic synthesis [15], [17], and [21], logic complexity [15] and [52] and state assignment [15] and [50]. Spectral methods for testing of logical networks by verification of the coefficients in the spectrum have been developed [20], [22], [23], [25], [30]-[33], [36], [45] and [46]. It should be stressed that an important problem of finding the complement of a Boolean function that has high complexity in the Boolean domain [3] and [44], can be solved very easily in the spectral domain because complementing the Boolean function corresponds to changing the sign of every spectral coefficient [21], [22]. Tautology of a Boolean function can be verified by calculating a certain coefficient (dc coefficient). The problem of constructing optimal data compression schemes by spectral techniques has also been considered. The latter approach is very useful for compressing test responses of logical networks and memories [23], [25], [45] and [46]. The renewed interest in applications of spectral methods in logic synthesis is caused by their excellent design for testability properties and the possibility of performing the decomposition with gates other than the ones used in classical approaches [15], [48]-[50] and [52].

Two design automation systems have used spectral methods as the tool for designing of digital circuits [14], [16], [48], [49], and [52]. Computation of the spectrum is a complex operation that requires, in the general case, $n2^n$ operations of additions/subtractions when the Fast Walsh Transform [24] is used and the Boolean function has n input variables. In order to store the calculated spectrum 2^n memory locations are required [22], [24]. The SPECSYS (for SPECTral SYnthesis System) developed at Drexel University on a VAX 11/780 uses the Fast Walsh Transform [24] for the calculation of the spectrum and can process Boolean functions having only 20 input variables

[16] and [48]. The DIADES design automation system developed at Portland State University [14] does not have any limit on the number of input variables of Boolean functions that can be processed and uses the methods described in this Dissertation for the generation of spectral coefficients of Boolean functions.

The following orthogonal transforms are discussed in this work: Walsh, Reed-Muller, Adding, and Arithmetic. All the above transforms can be calculated by either of the two methods introduced in this Dissertation. The *first method* for the generation of spectra explains the links between classical and spectral methods. The *second method* is based on the disjoint cube representation of Boolean functions. In this dissertation, the main emphasis is placed on the efficient computer calculation of the Rademacher-Walsh spectrum of Boolean functions since this particular ordering of the Walsh transform is most frequently used in digital logic design [17], [21]-[24], and [51]. However, the methods presented are of a general nature, and therefore they can be applied easily not only to all new orthogonal transforms introduced in this Dissertation but also to any future orthogonal transforms of Boolean functions.

In this Dissertation two *new methods* for the calculation of the Rademacher-Walsh spectrum of incompletely specified Boolean functions are shown. Both methods presented can calculate a Walsh spectrum of any ordering since the algorithms are independent of the ordering of the spectral coefficients. The *first method*, that allows calculation of the spectrum directly from a Karnaugh map, is introduced here for better understanding of the meaning of spectral coefficients in classical logic terms. The *second method* has been implemented in the DIADES automation system [14].

This Dissertation answers many important questions concerning efficient application of spectral methods in the computer-aided design of digital circuits. The main obstacle in these applications was, up to now, *memory requirements* for computer systems. By using the algorithms presented in this Dissertation this obstacle has been overcome.

The advantages of the approach presented were possible due to new insight and the formulation of spectral techniques. By investigating the links between spectral techniques and classical logic design methods, This interesting area of research is presented in a simple manner, by investigating the links between spectral techniques and classical logic design methods. The real meaning of spectral coefficients in classical logic terms (such as minterms and cubes) is shown. All the mathematical relationships between the number of true, false and don't care minterms, and spectral coefficients as well as between the size of disjoint cubes and spectral coefficients are stated.

One of the drawbacks of spectral techniques is that practically all the existing algorithms for calculating the spectral coefficients start from a Boolean function, represented either as a list of true minterms (alternatively - a list of false minterms) [15], [16], [21], [22], [25], [48], and [49] or as an already minimized sum-of-products Boolean expression [22] and [33]. The algorithm presented in Chapter V overcomes this weakness by representing a completely specified Boolean function as a set of disjoint cubes that completely covers the function. By such an approach, each spectral coefficient can be calculated separately or all the coefficients can be calculated in parallel. These advantages are absent in the existing methods. The possibility of calculating only some coefficients is very important since there are many spectral methods in digital logic design for which the values of only few selected coefficients are needed. Some of such cases where the entire spectrum need not to be computed are: Walsh and Reed-Muller spectral techniques for fault detection [20], [22], [23], [25], [30]-[32], [36], [105] and [106], spectral translation techniques for extracting core functions [21], [22], [27] and [28], designing of multiplexer based universal-logic-modules [29], prime implicants extraction [25] and [27], estimation of logic complexity [15], [50] and [52], and approximate implementation of logic functions [33].

Most of the current methods in the spectral domain deal only with completely

specified Boolean functions. On the other hand, all the algorithms introduced in this Dissertation are valid not only for completely specified Boolean functions but also for functions with don't cares, since don't care minterms can be simply represented in the form of disjoint cubes as well.

In order to use Boolean functions that are represented as arrays of non-disjoint cubes, an additional fast algorithm to generate disjoint cubes is presented. Use of the disjoint cubes representation of Boolean functions has been found advantageous in many algorithms used in digital logic design [3], [18], [37]-[39] and [54]. A detailed description of cube calculus operations can be found in [3], [43] and [54].

The theory of the calculation of spectral coefficients for Boolean functions is new for both of the methods introduced. The *second method* also allows for the calculation of spectra of a system of Boolean functions. When the system of incompletely specified Boolean functions is processed there is a restriction that all the functions in the system are assumed to be undefined at exactly the same points (minterms of a Karnaugh map). Optimal completion of don't care minterms for such a system of Boolean functions, from the point of view of a minimal number of spectral coefficients different from 0 (and by that obtaining simpler implementation of such a system of functions), was presented in [15]. However, this is a large restriction. The *second method* can process not only such functions but any system of completely and incompletely specified Boolean functions. Each function in the system of functions processed by the *second method* can have a don't care minterm anywhere in function's domain.

A short description of previously known properties of the classical Rademacher-Walsh transform when such a transform is applied to Boolean functions should make this Dissertation self-sufficient. All properties regarding incompletely specified Boolean functions are new. The formal proofs of these new properties by mathematical induction are trivial and therefore omitted.

The choice of a suitable transform for a given application is an important task for a designer who uses spectral techniques to design digital circuits. This Dissertation shows that by using the two concepts of "ordering" and "polarity" and by applying some new mathematical matrix operations, previously known transforms can be generalized and some new transforms can be created. The generalized transforms allow decomposition of logic functions described in terms of corresponding spectral coefficients calculated for orthogonal transforms with any ordering and polarity.

A systematic analysis of the concepts of ordering and polarity allowed the author to introduce the following transforms and to generalize the concept of some existing ones: Generalized Walsh Transforms for any ordering, two Walsh Transforms in Gray Code Ordering, Generalized Arithmetic and Adding Transforms for any ordering, and Generalized Reed-Muller Transforms for any ordering.

Spectra of Boolean functions of all of the above transforms can be calculated by either of the two methods introduced in this Dissertation. For all but one of the above transforms there exists an appropriate Fast Transform, and therefore the Fast Transform methods based on Good's concept of fast Fourier transform [64], [82] and [88] can be used also for the calculation of spectral coefficients. While introducing the concept of Generalized Arithmetic and Adding transforms, the author confined himself to the mathematical consideration of only elementary radix-2 matrices composed of only elements 0, 1 and -1. It is however obvious, that the elementary radix- 2^n matrices, where n is an arbitrary integer number can be a basis for the generation of other Generalized transforms. Which of the transforms generated in such a way have potential usefulness should be a topic of further investigations. During the research on new transforms, the author introduced some elementary operations on matrices: the vertical, horizontal and joint vertical - horizontal transformations and the bi-symmetrical Pseudo Kronecker product that is different from the Pseudo Kronecker product in the reference [55]. The same

name is a coincidence because the author was not aware of the name used in the reference at the time of the introduction of new operator [111]. It is interesting to note that it is possible to apply the new operators to other families of orthogonal elementary radix-2 matrices by what the alternative orthogonal and some not-orthogonal transformations can be created.

Here is a list of the chapters and a brief description of their contents, to establish a perspective on the Dissertation's presentation:

Chapter II: An introductory survey of discrete Walsh transforms. Examples of Walsh transforms with different orderings are shown.

Chapter III: Gives basic properties of Walsh transforms when they are applied to Boolean functions. Describes the important notion of standard trivial functions. Shows R and S coding of Boolean functions. Provides all of the relationships for Walsh spectra in both codings. Expresses values of spectral coefficients in terms of numbers of different types of minterms. Describes the *first method* of calculation of Walsh transforms. Builds bridges between spectral and classical approaches to logic design.

Chapter IV: Introduces Walsh transforms for multiple-valued input binary functions. Describes definitions and properties of two-dimensional maps that represent multiple-valued input binary functions. Gives all the algorithms necessary to calculate the spectral representation that is presented for multiple-valued input binary functions.

Chapter V: Describes the algorithm that generates the disjoint cube representation of Boolean functions. Introduces the *second method* of calculating Walsh spectra from the disjoint cube representation of a Boolean function. Expresses values of spectral coefficients in terms of disjoint cubes. Shows the calculation of Walsh spectra for systems of Boolean functions without any restriction on the domain of such functions.

Chapter VI: Introduces generalized Adding and Arithmetic transforms for both R and S codings. Describes new operations on matrices. Gives the properties of the new

transforms and expresses the values of spectral coefficients in terms of a number of different types of minterms.

Chapter VII: Describes the algorithm that calculates a generalized Reed-Muller transform from a disjoint cube representation of a Boolean functions. Provides all the basic definitions and properties of Reed-Muller spectral coefficients.

Chapter VIII: Introduces the generalized Walsh transform. Shows Adding, Arithmetic, and Reed-Muller transforms for different orderings. Presents circuit interpretation of spectral coefficients for different transforms.

Chapter IX: Introduces two Gray code orderings for Walsh transforms. Describes new operators on matrices: vertical mirror transformation, horizontal mirror transformation, vertical - horizontal mirror transformation and bi-symmetrical pseudo Kronecker product. Gives Fast Walsh transform for one of the introduced orderings. Presents constant geometry iterative architecture for Gray code ordered transform.

Chapter X: Conclusions reached.

CHAPTER II

PROPERTIES OF DISCRETE WALSH TRANSFORMS

II.1 RELATIONS BETWEEN DISCRETE FOURIER AND WALSH TRANSFORMS

The decomposition of continuous time signals into sums of sinusoids (The Fourier Transform) has been shown to be very important for studying of linear time-invariant systems [88]. This is a significant process because the response of a linear time-invariant system to a sinusoidal input is also a sine wave of the same frequency, but generally, with a different amplitude and phase.

Suppose that the signals considered are discrete time periodic or non-periodic functions. Such functions are of special interest to us because Boolean functions and their extensions to multiple-valued logic can be regarded as non-periodic functions. It has been shown that the Discrete Fourier Transform (DFT) is particularly suitable for describing phenomena related to a discrete time series [88]. Since the DFT is directly related to the Fourier Transform, many of the properties of the DFT are parallel to the properties of the Fourier Transform. The properties of the DFT and relationships between the DFT and the Fourier Transform are presented in [88]. The Fast Fourier Transform (FFT), a computational algorithm that reduces the number of multiplications and additions required for determining the coefficients of the DFT is also described in the same reference.

A number of linear transformations exist that have the form of the one-dimensional discrete Fourier transform relations. Such transformations can be written in the form [88]

$$F(i) = \sum_{k=0}^{N-1} f(k) a(k, i) \quad (\text{II.1})$$

$$f(k) = \sum_{i=0}^{N-1} F(i) b(k, i) \quad (\text{II.2})$$

where $a(k, i)$ is the *forward transformation kernel* and $b(k, i)$ is the *inverse transformation kernel*. In the case of the DFT, the functions $a(k, i)$ and $b(k, i)$ are $\exp(-2\pi k i j / N)$ and $\exp(2\pi k i j / N)$, respectively. In all the equations, $j^2 = -1$, N is the number of samples (an even number), and $f(k)$ is a sequence of N samples. The interval of i from 0 to $N - 1$ describes a finite time window for which the data sequence is available.

Another transformation pair of interest that is of this general type is the Walsh transformation. In the case of Boolean functions, i.e., functions on a two element field 0,1, the transformation kernel can be defined as

$$a_w(k, i) = \exp(2\pi k i j / 2) = (-1)^{k i} = \pm 1 \quad (\text{II.3})$$

when $k = 0$ or $k = 1$. The subscript w denotes a kernel that describes Walsh functions.

The above kernel can be regarded as a *generic kernel* (i.e., a kernel that *generates* orthogonal functions for a given ordering) of the Walsh transformation when it is compared to the kernels of Walsh transformations in Hadamard or Kaczmarz orderings [88]. Each ordering of the Walsh transform differs from the others by the order in which spectral coefficients corresponding to a given ordering are generated. The kernels for chosen orderings of Walsh functions will be presented later. When multiple-valued logical functions are considered then the generic kernel of orthogonal Chrestenson functions is obtained from Equations (II.2 and II.3) [15]. Since the interest of the author is only in the orthogonal transformations that can be used for Boolean and multiple-valued input logical functions the Chrestenson functions used for multiple-valued input multiple-valued output functions are not discussed. The interested reader can find more information on these other functions in reference [15].

II.2 WALSH FUNCTIONS

Walsh functions form a complete set of orthogonal functions in the interval $[0, 1]$. The advantage of a Walsh function expansion over other orthogonal expansions is that Walsh functions have only ± 1 values and multiplication by a Walsh function involves only algebraic sign assignment. Since the functions have only two values ± 1 then they can be easily generated by digital circuits.

Due to the binary nature of Walsh (and related) transforms, these transforms have an advantage over the Fourier transform because of their computational simplicity. That is, the transform matrices are made of only ± 1 values and the required computations in any digital signal processing application are only additions and subtractions. The resulting hardware implementation of such transforms is relatively simple [57], [65] and [80]. An example of such a hardware realization for Walsh transforms in Gray code ordering, newly introduced by the author, are presented in Chapter IX. These general properties of Walsh and related orthogonal transforms have made them of interest to scientists in many fields: image processing [56], [77], [82], [117] and [118], signal coding [24], [64], [69] and [88], filter design [120], imaging in spectroscopy and astronomy [116] and [117]-[119], telecommunication [24] and [69], statistical analysis [76], digital control systems [115], associative memory systems [121], and many other areas.

The basic properties of the spectral Walsh coefficients of Boolean functions are given in Chapter III. The Walsh functions may be presented in different orderings. The mutual relationships between different orderings of discrete Walsh functions are shown in Chapter IX. The Walsh functions are not sinusoidal nor do they possess the notion of frequency in any conventional sense. However, the functions cross the zero axis, and as an analog to frequency, the term "sequency" has been introduced by Hartmuth [69]. The comparison of sequencies for different Walsh transform ordering is done in Chapter IX.

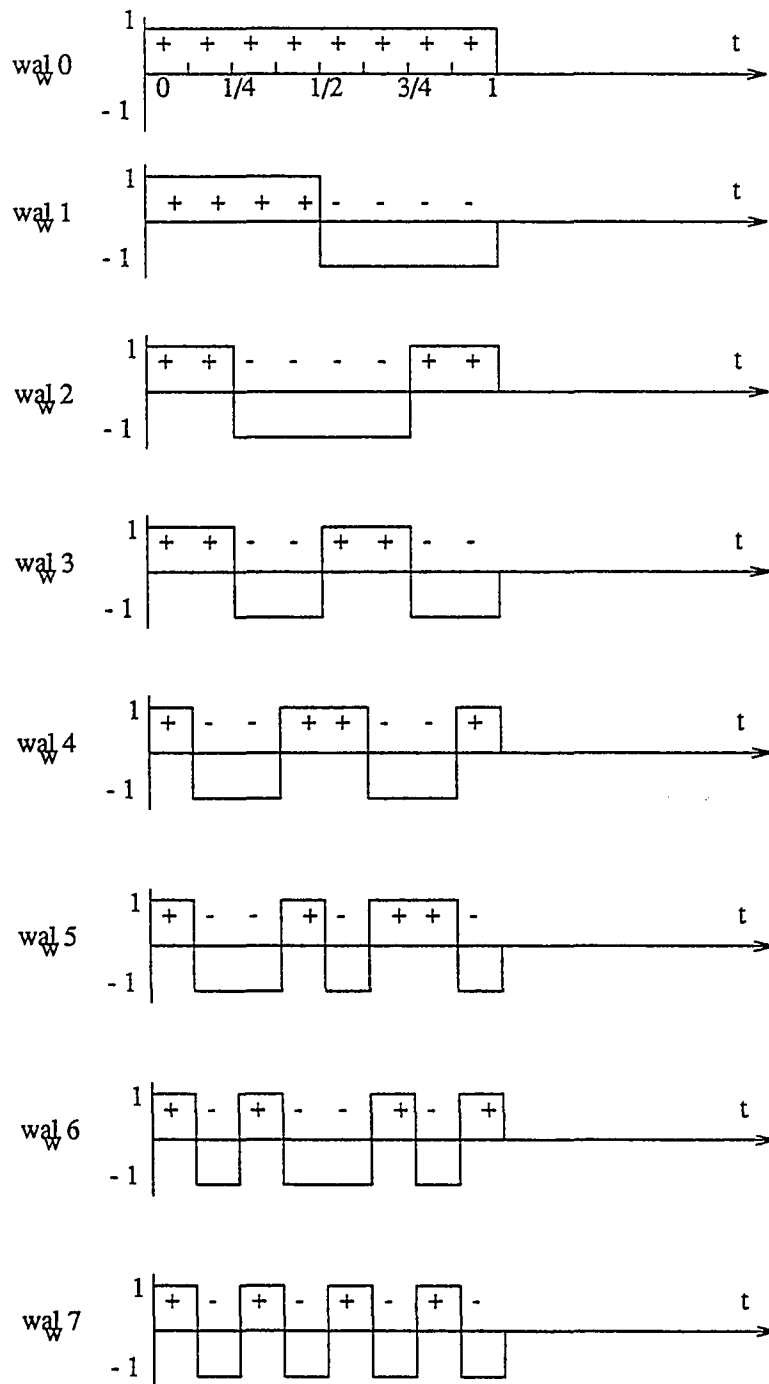


Figure 1. Kaczmarz ordered Walsh functions.

It is possible to apply different approaches to the definitions of Walsh functions [64] and [88]. For example, Walsh functions can be defined recursively or in the terms of Rademacher functions [15], [64] and [88]. However, the Walsh functions in any ordering comprise block waves (changing between +1 to -1) over the interval [0, 1]. Because Walsh functions form an orthogonal set, they are useful in creating additional transforms in the same way that sine and cosine functions are useful in a number of transforms.

Let us consider the Kaczmarz ordered Walsh functions shown in Figure 1. If these functions are sampled at the designated $8 = 2^3$ points (more generally at $N = 2^r$ points) then the 8×8 matrix is obtained. This matrix is shown in Figure 2, which shows only the signs (all of the amplitudes are equal to 1).

k \longrightarrow										
	i \downarrow		0	1	2	3	4	5	6	7
		0	+	+	+	+	+	+	+	+
		1	+	+	+	+	-	-	-	-
		2	+	+	-	-	-	-	+	+
		3	+	+	-	-	+	+	-	-
		4	+	-	-	+	+	-	-	+
		5	+	-	-	+	-	+	+	-
		6	+	-	+	-	-	+	-	+
		7	+	-	+	-	+	-	+	-

Figure 2. Values of the Kaczmarz ordered Walsh transform.

All rows (or columns) in this matrix are mathematically orthogonal to each other. The matrix represents discrete Walsh-ordered functions. The one-dimensional kernel for this function is as follows [15], [64], and [88]:

$$a_w(k, i) = (-1)^{\sum_{r=0}^{r-1} p_r(k) q_r(i)} \quad (\text{II.4})$$

where

$$\begin{aligned} q_0(i) &= p_{r-1}(i) \\ q_1(i) &= p_{r-1}(i) + p_{r-2}(i) \\ q_2(i) &= p_{r-2}(i) + p_{r-3}(i) \\ &\vdots \\ q_{r-1}(i) &= p_1(i) + p_0(i) \end{aligned}$$

and where the arguments of $p_r(k)$ and $q_r(i)$ are written in straight binary code, with the value of $p_r(i)$ equal to the r -th bit in the binary code of i . In particular, the Walsh transformation in Kaczmarz ordering is written explicitly as:

$$F(i) = \sum_{k=0}^{N-1} f(k) (-1)^{\sum_{r=0}^{r-1} p_r(k) q_r(i)} \quad (\text{II.5})$$

$$f(k) = \frac{1}{N} \sum_{i=0}^{N-1} F(i) (-1)^{\sum_{r=0}^{r-1} p_r(k) q_r(i)} \quad (\text{II.6})$$

Example II.1:

The element $a_w(4, 3)$ of the matrix shown in Figure 2 can be calculated using Equation (II.4) (notice that $r = 3$) as follows:

$$\begin{aligned} a_{w43}(r=3) &= (-1)^{\sum_{r=0}^2 p_r(4) q_r(3)} = (-1)^{[p_0(4) q_0(3) + p_1(4) q_1(3) + p_2(4) q_2(3)]} = \\ &= (-1)^{[p_0(4) p_2(3) + p_1(4) [p_2(3) + p_1(3)] + p_2(4) [p_1(3) + p_0(3)]]} \end{aligned}$$

The values of $p_r(k)$ are as follows:

$$p_0(4) = p_0(100) = 0; p_1(4) = p_1(100) = 0; p_2(4) = p_2(100) = 1;$$

$$p_0(3) = p_0(011) = 1; p_1(3) = p_1(011) = 1; p_2(3) = p_2(011) = 0$$

Therefore, the value of the element considered is:

$$a_{w43}(r=3) = (-1)^{[0 \times 0 + 0 \times [0 + 1] + 1 \times [1 + 1]]} = (-1)^2 = +1 = +.$$

Figure 2 also shows that the value of the considered element is +.

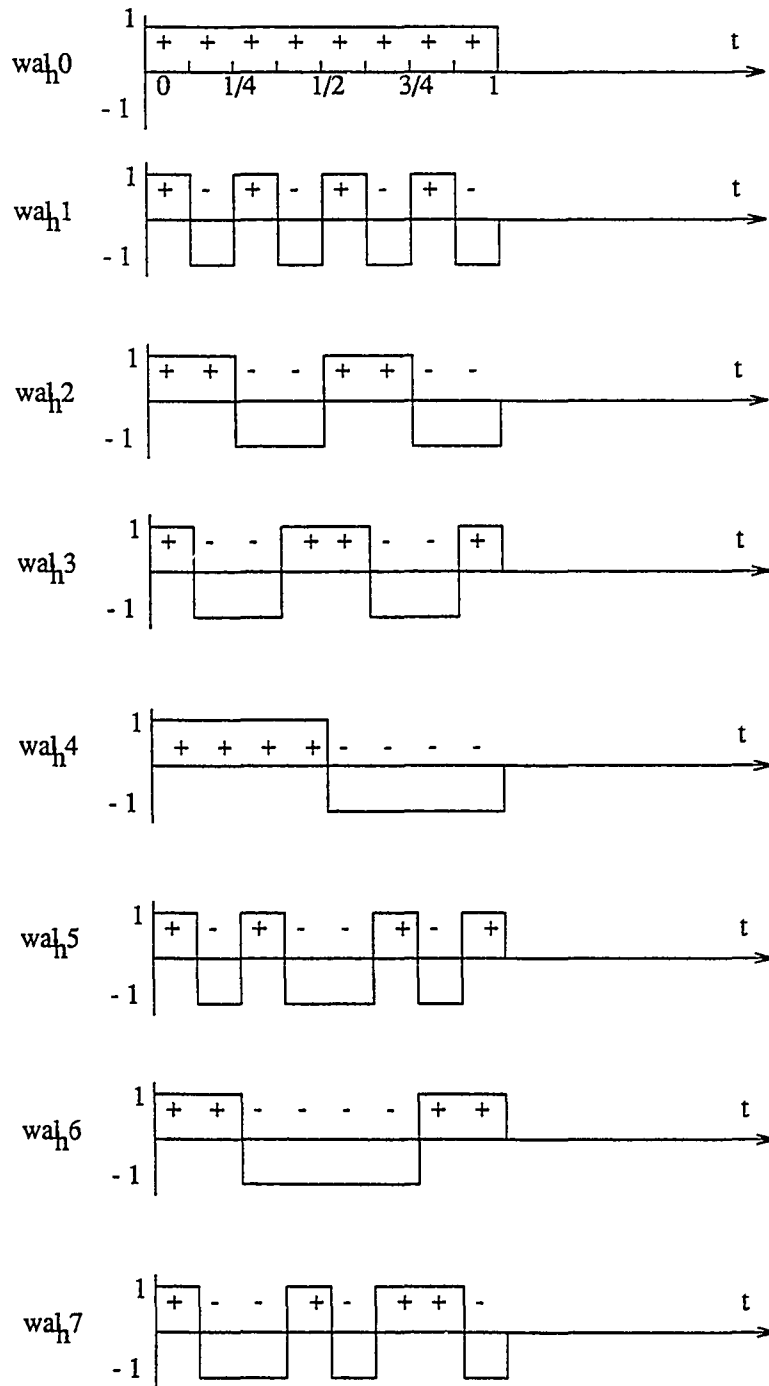


Figure 3. Hadamard ordered Walsh functions.

Consider now Walsh functions with Hadamard ordering as shown in Figure 3. In fact, the Hadamard-ordered Walsh function transformations are precisely the same as the Walsh function transformation described by the Equations (II.5) and (II.6). The only difference is the way in which the values for k and i are written. For each k and i , the straight binary code is written first, as it is with the Walsh-ordered functions. Next the binary forms are bit reversed and then the Gray code conversion of the bit reversed number is used.

When the Hadamard-ordered Walsh functions are sampled in 8 equal samples, an 8×8 matrix is obtained as shown in Figure 4.

		k →							
		0 1 2 3 4 5 6 7							
i ↓									
	0	+	+	+	+	+	+	+	+
	1	+	-	+	-	+	-	+	-
	2	+	+	-	-	+	+	-	-
	3	+	-	-	+	+	-	-	+
	4	+	+	+	+	-	-	-	-
	5	+	-	+	-	-	+	-	+
	6	+	+	-	-	-	-	+	+
	7	+	-	-	+	-	+	+	-

Figure 4. Hadamard ordered discrete Walsh transform.

Notice the presence in this matrix of an array of second order matrices (one of the sub-matrices has a simple change of the sign). The second order Hadamard matrix H_2 can be used to construct recursively any higher order matrices (of order $2 \times N$). This is done by taking the Kronecker product of the elementary second order H_2 matrix with the same matrix. The details of the Kronecker product operation are explained in Chapter IX.

The basic properties of other commonly used Walsh orderings are discussed in

Chapter IX. The Walsh functions can also be defined by the sets of other related orthogonal functions. S. Kaczmarz developed the definition of Walsh functions by using only Haar functions [15]. R. Paley defined the Walsh functions using Rademacher functions [64]. By investigating different orderings and definitions of Walsh functions, the author introduced two new orderings of Walsh functions that are presented in Chapter IX.

Walsh transform calculations can be presented in forms analogous to FFT calculations. All the Fast Walsh Transform algorithms use a Cooley-Tukey type data flow graph [64]. The reader can find these fast flow graphs in [24], [56], [64] and [88]. The fast flow graph for the transform introduced by the author is presented in Chapter IX.

CHAPTER III

ESSENTIAL RELATIONS BETWEEN CLASSICAL AND SPECTRAL APPROACHES TO ANALYSIS, SYNTHESIS, AND TESTING OF COMPLETELY AND INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS

III.1 DESCRIPTION

Some necessary background information on Rademacher-Walsh transforms is presented in this Chapter. This particular ordering of Walsh transforms is frequently used for logic design. However, the new method presented by the author in this Chapter can calculate the Walsh spectrum for any ordering since the method is independent of the ordering of the spectrum. An investigation of the links between spectral techniques and classical logic design methods yields a simple presentation of this interesting area of research. The real meaning of the spectral coefficients from R and S Walsh spectra in classical logic terms is shown. Moreover, two algorithms are shown for easy calculation of spectral coefficients, for completely and incompletely specified Boolean functions, by handwritten manipulations directly from Karnaugh maps. All mathematical relationships between the number of true, false and don't care minterms and the Walsh spectral coefficients are stated. A number of examples are given which should help to introduce these ideas to engineers working in the areas of test generation and logic design automation. The use of complicated mathematical formulas, so typical in articles on the subject, is minimized in this presentation. This is important, since unfortunately up to now unfamiliarity with the mathematical side of the spectral approach seems to be too great a hurdle to overcome for finding a fruitful place for practical applications of these ideas.

III.2 BASIC PROPERTIES OF RADEMACHER-WALSH SPECTRUM

The Rademacher-Walsh spectra S and R of an n -variable Boolean function are alternative canonical representations of the Boolean function. Other related canonical descriptions of Boolean functions are: the Adding and Arithmetic forms (Chapter VI), and the Reed-Muller form (Chapter VII). The latter form is, however, a canonical representation of only completely specified Boolean functions.

The Rademacher-Walsh spectrum S is formed as the product of a $2^n \times 2^n$ Rademacher-Walsh matrix T and a 2^n vector representation, M of a Boolean function F . When the Boolean function is represented as a truth table of all minterms, M will be either the $\langle +1, -1 \rangle$ vector representation of the truth table for a completely specified Boolean function, or the $\langle +1, 0, -1 \rangle$ vector representation of the truth table for an incompletely specified Boolean function [15], [21], [22], [24], [25] and [32]. In the coding scheme the conventional $\langle 0, 1, - \rangle$ values (false, true and don't care minterms) correspond to $\langle +1, -1, 0 \rangle$ codings, respectively ('-' stands for 'don't care'). This type of coding of the truth vector is called the S coding, and the coded truth vector is denoted by M . The values of the minterms in the original truth vector and the coded vector M are ordered according to the straight binary code of variables describing the minterms. For example, the first entry in the vector is the logical value of the minterm that is described by all negated variables (minterm 0), the second entry in the truth vector is the logical value of the second minterm that is described by all but x_1 negated variables, etc. The Rademacher-Walsh matrix T represents the Walsh functions in Rademacher ordering. The rules for recursive generation of such matrices are described in [15].

The Rademacher-Walsh spectrum R results from another coding of the truth table of a Boolean function with the conventional $\langle 0, 1, - \rangle$ values corresponding to $\langle 0, 1, 0.5 \rangle$ respectively. This Rademacher-Walsh spectrum is called the R spectrum and this type of coding is called R coding. The relationships between the spectral coefficients of the S

and R spectra are given later in this Chapter.

Besides the matrix method presented in this Chapter, recursive algorithms, data flow-graph methods and parallel calculations, similar to the Fast Fourier Transform, have also been used to calculate the Rademacher-Walsh and other related transforms [22], [24], [25] and [51]. All of the methods that have been mentioned reduce the necessary number of calculations from $n \times n$ to $n \times \log_2 n$, but they still require excessive computer memory. They also have some undesirable properties that are overcome by using the second spectrum calculation method which will be introduced in Chapter V.

The principal *properties* of the coefficients of the S spectrum for completely specified Boolean functions are shown below. The Properties derived from the four well known Walsh type transform matrices are taken from references [15], [21], [22], [24] and [25]. Properties III.4, III.7, III.9-III.13, III.16 and III.17 are new.

- III.1 The transform matrix of each ordering of the Walsh functions is *complete* and *orthogonal*, therefore, there is no information lost in the S and R spectra, concerning the minterms of the Boolean function F .
- III.2 Only the Hadamard-Walsh matrix describing the Hadamard-Walsh transform has the *recursive Kronecker product structure*. Other possible variants of the Walsh transforms, described by the corresponding matrices, are known in the literature as the Walsh-Kaczmarz, Rademacher-Walsh, and Walsh-Paley transforms.
- III.3 Only the Rademacher-Walsh matrix is not *symmetric*; all other variants of Walsh matrices are symmetric, so that, disregarding a scaling factor the same matrix can be used for both the forward and inverse transform operations.
- III.4 Each spectral coefficient s_I or r_I is described by its *order*, *subindices* and *magnitude*. The order of the spectral coefficient is equal to the number of subindices, and the subindices are the subscripts of all variables of a standard trivial function corresponding to the coefficient. The magnitude of a spectral coefficient is its

value. In the following material i, j, k denote subindices, and the order is denoted by o .

- III.5 When the classical matrix multiplication method is used to generate spectral coefficients for different Walsh transforms (different T matrices represent different Walsh functions with different orderings), the only difference in the result is the *ordering* of the spectral coefficients. The coded vector M is the same for all orderings of the Walsh functions. The values of the coefficients s_I and r_I with the same subindices are the same for every ordering of the Walsh transforms.
- III.6 Each spectral coefficient (either s_I or r_I) gives a *correlation value* between the Boolean function F and a *standard trivial function* u_I corresponding to the coefficient. The standard trivial functions for the spectral coefficients are, respectively: for the dc coefficients (direct current coefficients) s_I or r_I ($I = 0$) - the universe of the Boolean function F denoted by u_0 ; for the first order coefficients s_I or r_I ($I = i, i \neq 0$) - the variable x_i of the Boolean function F denoted by u_i ; for the second order coefficients s_I ($I = ij, i \neq 0, j \neq 0, i \neq j$) - the Exclusive-OR function between variables x_i and x_j of the Boolean function F denoted by u_{ij} ; for the third order coefficients s_I or r_I ($I = ijk, i \neq 0, j \neq 0, k \neq 0, i \neq j, i \neq k, j \neq k$) - the Exclusive-OR function between variables $x_i, x_j,$ and x_k of the Boolean function F denoted by u_{ijk} ; etc.
- III.7 The number of spectral coefficients of z -th order is equal to $C_n^z = \binom{n}{z}$, where n is the number of variables of a Boolean function.
- III.8 For a completely specified Boolean function the maximal value of any individual spectral coefficient s_I is $+2^n$ and the minimal value is -2^n . This happens when the Boolean function F is equal to either a standard trivial function u_I (sign +) for the maximal value or to its complement (sign -) for the minimal one. In either case, all the remaining spectral coefficients have zero values because of the

orthogonality of the transform matrix T .

- III.9 The maximal value of any spectral coefficient r_I except r_0 is $+2^{n-1}$ and will result when the Boolean function F is equal to complement of a standard trivial function u_I . The minimal value is -2^{n-1} and will result when F is equal to a standard trivial function u_i . In either case, all the remaining spectral coefficients have zero values because of orthogonality of the transform matrix T .
- III.10 For an incompletely specified Boolean function the maximal value of any individual spectral coefficient s_I is $+2^n - 1$ and the minimal value is $-2^n + 1$. This happens when the Boolean function F is equal to either the standard trivial function, for the maximal value of s_I , or to its complement for the minimal value, in all but one of the minterm.
- III.11 The maximal value of the r_0 spectral coefficient is $+2^n$ and will result when the Boolean function F is tautology. The minimal value is -2^n and will result when F is equal to the complement of the tautology. The tautology is the logical function for which all the minterms are true.
- III.12 When for more than half of spectral coefficients, of any completely specified Boolean function F the majority of the minterms have the same logical values as the minterms of standard trivial functions the sum of all of the coefficients of the S spectrum has a maximal value equal to $+2^n$. When for more than half of spectral coefficients the majority of minterms of F have the complemented logical values to the minterms of standard trivial functions the sum of all of the coefficients of the S spectrum has a minimal value equal to -2^n .
- III.13 For any incompletely specified Boolean function the sum of all of the spectral coefficients of the spectrum S has a maximal value of $+2^n - 1$ and a minimal value of $-2^n + 1$. The maximal or minimal value happens when the Boolean function has exactly one don't care minterm and all the spectral coefficients and

minterms follow the rule from Property III.12.

- III.14 Every standard trivial function u_I , except u_0 , corresponding to an n variable Boolean function F has the same number of true and false minterms, equal to 2^{n-1} .
- III.15 For each true minterm the coefficients from the spectrum S are: $s_0 = 2^n - 2$, and all remaining $2^n - 1$ spectral coefficients s_I are equal to ± 2 . The way of choosing the signs of spectral coefficients is defined in Chapter V (Properties V.6 - V.8).
- III.16 The spectrum S of each false minterm is given by $s_I = 0$.
- III.17 For each don't care minterm the coefficients from the spectrum S are: $s_0 = 2^{n-1} - 1$, and all of the remaining $2^n - 1$ spectral coefficients s_I are equal to ± 1 . The way of choosing the signs of spectral coefficients is defined in Chapter V (Properties V.6 - V.8).

Example III.1:

An example of the calculation of the Rademacher-Walsh spectrum S of a four variable completely specified Boolean function is shown in Figure 5. The matrix T describes discrete Walsh functions in Rademacher ordering. The coded truth vector M represents the values of minterms of a Boolean function in the S coding. For this example, the function from Figure 7 is used.

T	M	S	
1	1	2	s_0
1	-1	6	s_4
1	1	2	s_3
1	1	2	s_2
1	1	2	s_1
1	-1	6	s_{34}
1	1	-2	s_{24}
1	-1	-6	s_{23}
1	1	6	s_{14}
1	-1	-6	s_{13}
1	-1	2	s_{12}
1	1	6	s_{234}
1	1	-2	s_{134}
1	1	-2	s_{124}
1	-1	2	s_{123}
1	-1	-2	s_{1234}

Figure 5. Calculation of Rademacher-Walsh S spectrum of completely specified function.

Example III.2:

An example of the calculation of Rademacher-Walsh spectrum S of a four variable incompletely specified Boolean function is shown in Figure 6. In this example, the function from Figure 8 is used. One can notice easily, that the first entry in the vector M has value 0 since it corresponds to the logical value don't care of the minterm described by all variables negated on the Karnaugh maps of Figure 8. Since the Boolean function has only one don't care minterm, all other entries in the vector M are either + 1 or - 1.

method can be applied for finding spectral coefficients of only completely specified Boolean functions. However, the agreement - disagreement method, that is suitable for hand calculations, takes into the consideration all 2^n combinations of values of n variables of a Boolean function which number is equal to 2^n .

A similar method for hand calculation was proposed in [51]. The advantage of this approach is that only one half of all the combinations of n variables of a Boolean function has to be considered. Thus, the speed of this proposed technique is at least twice the speed of the agreement - disagreement method. However, this method has been presented only for completely specified Boolean functions and only for the spectrum S (which was erroneously denoted by the symbol R in [51]). Due to mutual relationships between both spectra this method can be easily extended for the calculation of the R spectrum. In this Chapter, all possible relationships between spectral coefficients from both spectra and classical logic terms are stated. Moreover, for the first time these relationships are presented not only for completely specified Boolean functions but for incompletely specified Boolean functions as well. The latter problem has been solved by the author for both S and R spectra. It has also been checked that the relationships that bind both these spectra together (equations III.4 and III.5) are still valid for the spectra calculated for incompletely specified Boolean functions. By expressing spectral coefficients through different formulas (what has never been done thoroughly even for completely specified Boolean functions) one is able to calculate the spectral coefficients from different available data what makes the methods more flexible.

Let us show more clearly the meaning of r_l and s_l spectral coefficients in classical logic terms. Moreover, let us expand our considerations for incompletely specified Boolean functions as well.

The following symbols will be used in the equations. Let a_l be the number of true minterms of a Boolean function F , where both the function F and the standard

trivial function u_I have the logical values 1; let b_I be the number of false minterms of the Boolean function F , where the function F has the logical value 0 and the standard trivial function u_I has the logical value 1; let c_I be the number of true minterms of the Boolean function F , where the function F has the logical value 1 and the standard trivial function u_I has the logical value 0; let d_I be the number of false minterms of the Boolean function F , where both the function F and the standard trivial function u_I have the logical values 0, let e_I be the number of don't care minterms of the Boolean function F , where the standard trivial function u_I has the logical value 1, and let f_I be the number of don't care minterms of the Boolean function F , where the standard trivial function u_I has the logical value 0.

Then, for completely specified Boolean functions having n variables, the following formulas hold for all but the s_0 and r_0 spectral coefficient (when $I \neq 0$):

$$a_I + b_I + c_I + d_I = 2^n \quad (\text{III.1})$$

and

$$a_I + b_I = c_I + d_I = 2^{n-1} \quad (\text{III.2})$$

For the s_0 and r_0 spectral coefficients (when $I = 0$):

$$a_I + b_I = 2^n \quad (\text{III.3})$$

as c_I and d_I are both zero.

The relationships between both spectra R and S are as follows [22]:

$$r_0 = \frac{1}{2} (2^n - s_0) \quad (\text{III.4})$$

and

$$r_I = -\frac{1}{2} s_I \quad (\text{III.5})$$

when $I \neq 0$.

Accordingly, for incompletely specified Boolean functions, having n variables, the following formulas hold for all but the s_0 and r_0 spectral coefficient (when $I \neq 0$):

$$a_I + b_I + c_I + d_I + e_I + f_I = 2^n \quad (\text{III.6})$$

and

$$a_I + b_I + e_I = c_I + d_I + f_I = 2^{n-1}. \quad (\text{III.7})$$

For the s_0 and r_0 spectral coefficients (when $I = 0$) :

$$a_I + b_I + e_I = 2^n \quad (\text{III.8})$$

as c_I , d_I , and f_I are all zero.

Subsequently, relations (III.1-III.8) are used, where necessary, to derive different alternative expressions for spectral coefficients and the final formulas are presented.

The s_I spectral coefficient (when $I = 0$) for completely specified Boolean functions can be defined in the following alternative ways:

$$s_I = 2^n - 2 a_I \quad (\text{III.9})$$

or

$$s_I = 2 b_I - 2^n \quad (\text{III.10})$$

or

$$s_I = b_I - a_I. \quad (\text{III.11})$$

The s_I spectral coefficients (when $I \neq 0$) for completely specified Boolean functions can be calculated according to any of the following formulas:

$$s_I = 2 (a_I + d_I) - 2^n \quad (\text{III.12})$$

or

$$s_I = 2^n - 2 (b_I + c_I) \quad (\text{III.13})$$

or

$$s_I = 2 (a_I - c_I) \quad (\text{III.14})$$

or

$$s_I = 2 (d_I - b_I) \quad (\text{III.15})$$

or

$$s_I = (a_I + d_I) - (b_I + c_I). \quad (\text{III.16})$$

Similarly, for completely specified Boolean functions having n variables, the r_I spectral coefficient (when $I = 0$) can be defined in three ways from equations (III.9-III.11) by using the relationship of equation (III.4). The r_I spectral coefficients (when $I \neq 0$) for completely specified Boolean functions having n variables can be defined in five ways from equations (III.12-III.16) by using the relationship of equation (III.5).

Let us now expand our considerations for incompletely specified Boolean functions having n variables. Then, the s_I spectral coefficient (when $I = 0$) can be defined in the following alternative ways:

$$s_I = 2^n - 2a_I - e_I \quad (\text{III.17})$$

or

$$s_I = 2b_I + e_I - 2^n \quad (\text{III.18})$$

or

$$s_I = b_I - a_I. \quad (\text{III.19})$$

The s_I spectral coefficients (when $I \neq 0$) for incompletely specified Boolean functions can be calculated according to the following formulas:

$$s_I = 2(a_I + d_I) + e_I + f_I - 2^n \quad (\text{III.20})$$

or

$$s_I = 2^n - 2(b_I + c_I) - (e_I + f_I) \quad (\text{III.21})$$

or

$$s_I = 2(a_I - c_I) + e_I - f_I \quad (\text{III.22})$$

or

$$s_I = 2(d_I - b_I) + f_I - e_I \quad (\text{III.23})$$

or

$$s_I = (a_I + d_I) - (b_I + c_I). \quad (\text{III.24})$$

Similarly, for incompletely specified Boolean functions having n variables, the r_I spectral coefficient (when $I = 0$) can be defined in three ways from equations (III.17-III.19) by using the relationship of equation (III.4). The r_I spectral coefficients (when $I \neq 0$) for completely specified Boolean functions having n variables can be defined in five ways from equations (III.20-III.24) by using the relationship of equation (III.5).

Here, it is interesting to note the following properties of the above equations. First, equations (III.4) and (III.5) are valid for spectra calculated for both completely and incompletely specified Boolean functions. Secondly, formulas (III.11) and (III.19) for the calculation of direct current (s_0) and corresponding formulas for the calculation of the r_0 coefficient are exactly the same for both, completely and incompletely specified Boolean functions. The same phenomenon occurs in formulas (III.16) and (III.24) and in the corresponding formulas for the calculation of the r_I (when $I \neq 0$) spectral coefficients. A number of don't care minterms e_I and f_I are eliminated from these formulas.

Of course, this does not mean that the spectral coefficients for incompletely specified Boolean functions do not depend on the number of don't care minterms. They do, but this dependence is included in the formulas (III.6-III.8).

Example III.3:

As a numerical example, consider a four variable completely specified Boolean function for which all standard trivial functions and the values of all corresponding a_I , d_I are given in Figure 7. This is the same Boolean function that was used in Example III.1.

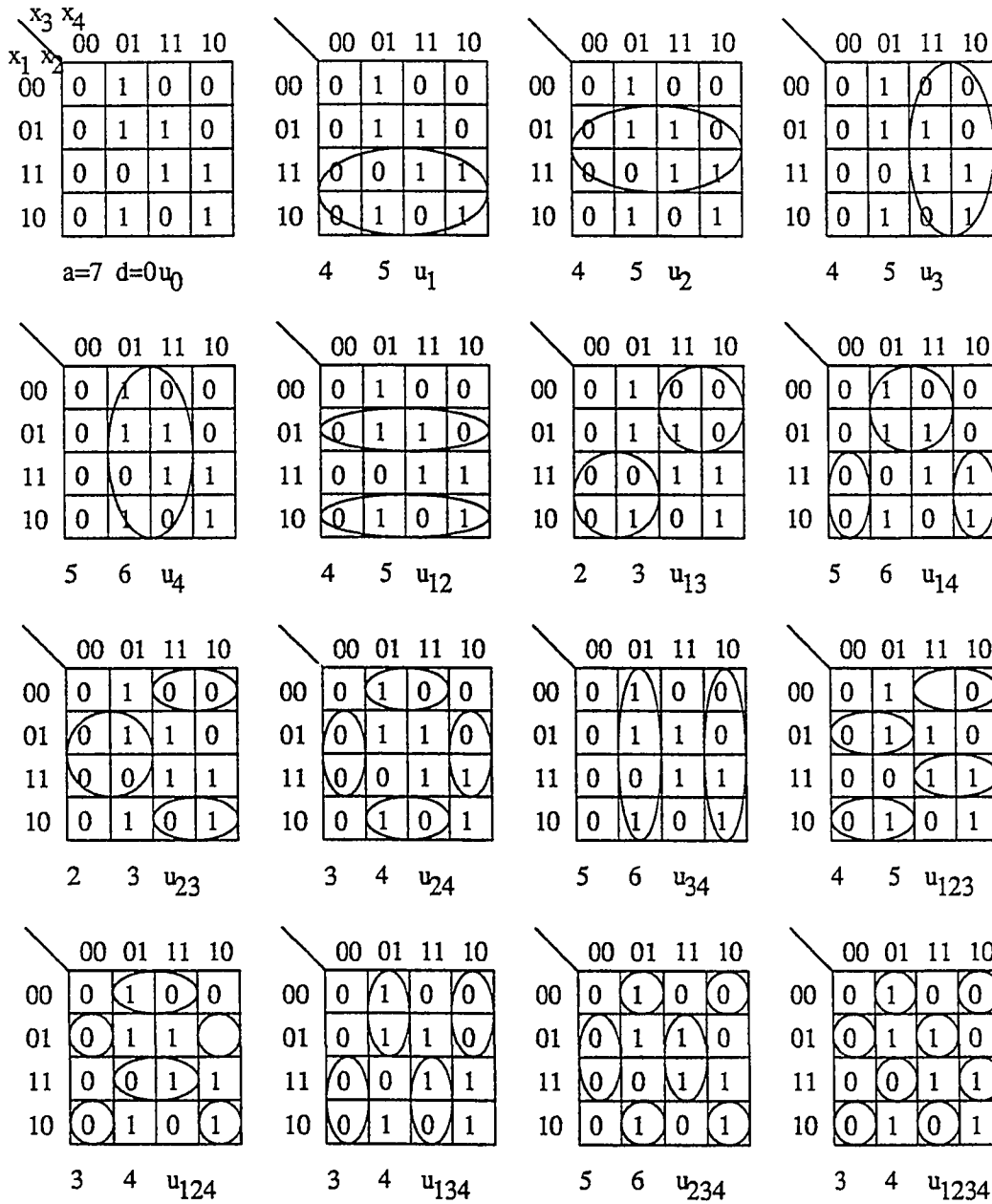


Figure 7. Standard trivial functions for completely specified Boolean function.

Then, according to equations (III.9) and (III.12) the spectral coefficients s_I for this function are as follows:

$$\begin{aligned}
 s_0 &= 16 - 14 = 2, s_1 = 18 - 16 = 2, \\
 s_2 &= 18 - 16 = 2, s_3 = 18 - 16 = 2, \\
 s_4 &= 22 - 16 = 6, s_{12} = 18 - 16 = 2, \\
 s_{13} &= 10 - 16 = -6, s_{14} = 22 - 16 = 6, \\
 s_{23} &= 10 - 16 = -6, s_{24} = 14 - 16 = -2, \\
 s_{34} &= 22 - 16 = 6, s_{123} = 18 - 16 = 2, \\
 s_{124} &= 14 - 16 = -2, s_{134} = 14 - 16 = -2, \\
 s_{234} &= 22 - 16 = 6, s_{1234} = 14 - 16 = -2.
 \end{aligned}$$

As can be seen from the above example the graphical method can generate spectral coefficients of Walsh functions in any ordering, or only some subset of them. The spectral coefficients obtained have exactly the same values as the ones obtained for this Boolean function by using the classical matrix multiplication method (Figure 5).

Example III.4:

As a numerical example, consider a four variable incompletely specified Boolean function for which all standard trivial functions and the values of all corresponding a_I , d_I , e_I , and f_I are shown in Figure 8. This Boolean function was used previously in Example III.2.

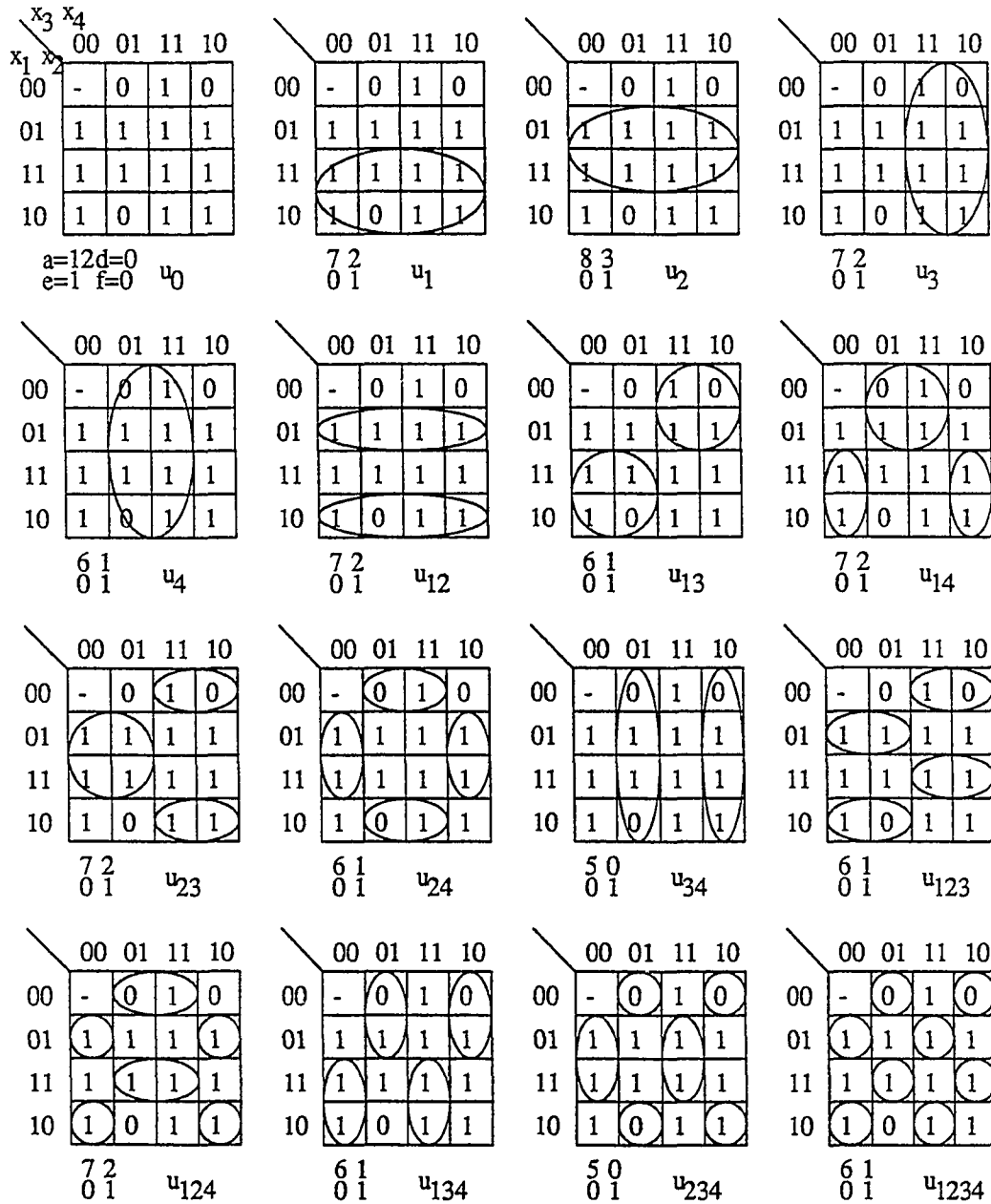


Figure 8. Standard trivial functions for incompletely specified Boolean function.

According to equations (III.17) and (III.20) the s_I spectral coefficients for this function are as follows:

$$s_0 = 16 - 24 - 1 = -9, s_1 = 18 + 1 - 16 = 3,$$

$$s_2 = 22 + 1 - 16 = 7, s_3 = 18 + 1 - 16 = 3,$$

$$s_4 = 14 + 1 - 16 = -1, s_{12} = 18 + 1 - 16 = 3,$$

$$s_{13} = 14 + 1 - 16 = -1, s_{14} = 18 + 1 - 16 = 3,$$

$$s_{23} = 18 + 1 - 16 = 3, s_{24} = 14 + 1 - 16 = -1,$$

$$s_{34} = 10 + 1 - 16 = -5, s_{123} = 14 + 1 - 16 = -1,$$

$$s_{124} = 18 + 1 - 16 = 3, s_{134} = 14 + 1 - 16 = -1,$$

$$s_{234} = 10 + 1 - 16 = -5, s_{1234} = 14 + 1 - 16 = -1.$$

The spectral coefficients obtained have exactly the same values as the ones obtained for the same Boolean function by using the classical matrix multiplication method (Figure 6).

The relations that have been determined in this Chapter allow the derivation of many properties of spectral coefficients that are important in their applications in the analysis and synthesis of logic circuits.

CHAPTER IV

WALSH TYPE TRANSFORMS FOR COMPLETELY AND INCOMPLETELY SPECIFIED MULTIPLE-VALUED INPUT BINARY FUNCTIONS

IV.1 DESCRIPTION

A multiple-valued input binary function is an extension of a Boolean function. Multiple-valued input binary functions find several applications in logic design, pattern recognition, and other areas [38], [39] and [99]-[102]. In logic design, they are primarily used for the minimization of PLAs that have 2-bit decoders on the inputs [5], and [100]. A PLA with r -bit decoders implements directly a sum-of-products expression (SOPE) of a 2^r -valued input binary function. As shown in [5], every set of m Boolean functions, where each of them has n binary variables, can be represented as a $n + 1$ multiple-valued input binary function with n binary inputs and one m -valued input. Thus, the standard logic minimization problem for multiple-output functions can be seen as a multiple-valued minimization problem.

The choice of an orthogonal transform for a given problem is very important since it affects the complexity of the calculations in the spectral domain as well as the calculation of the forward and inverse transforms. Currently, the main tools that are used for Boolean and multiple-valued (input as well as output) functions are the following transforms : Walsh, Chrestenson, Haar, Watari, polynomial Fourier transform, number-theoretic transform and the generalized discrete Fourier transform [15], [24] and [25]. In general, the interpretation of the spectral coefficients, except for Walsh, of the above

transforms is cumbersome or impossible. Only the Walsh spectrum, that can be applied for Boolean functions, has an interpretation in classical logic terms. The Walsh spectral coefficients can be calculated either from the minterms (Chapter III) or the disjoint cube representation (Chapter V) of the function. Special transforms for multiple-valued input binary functions have never been proposed.

In [39] the Mixed-Radix Exclusive Sum of Products concept was introduced for multiple-valued input binary functions. This points out the usefulness of multiple-valued generalizations of Reed-Muller transforms. Similarly, in this Chapter, Walsh-type spectra for multiple-valued input binary functions are proposed. There are two Walsh spectra for Boolean functions, S and R (Chapter III). Conventionally, the first spectrum, S , is used for analysis and synthesis of Boolean functions while the second one, R , is used in the design for testability of digital circuits [22]. It should be noticed, however, that each of these spectra can be used interchangeably since they are linearly related by Equations (III.4) and (III.5). One can expect similar applications of the transforms S and R transforms introduced here for multiple-valued input binary functions.

The author's main reason for the introduction of such type of new transforms was for the author the requirement of developing the transform which has a minimal number of coefficients and an easily understood interpretation. This Chapter introduces a new approach to spectral methods. First, the transform for a binary function of n multiple-valued variables is a vector of $\binom{n}{2}$ *partial transforms* of all pairs of these variables what minimizes the necessary spectral information to be kept about the function (partial coefficients). Secondly, the partial coefficients describe some global properties of the function and can be used by themselves, as well as for generating the final coefficients. Thirdly, the interpretation of each partial transform is given in classical logic terms (minterms). The values of partial spectral coefficients for completely and incompletely specified multiple-valued input binary functions are expressed by the number of true,

false and don't care minterms in the standard trivial functions corresponding to these coefficients.

All of the investigations presented can be applied to any multiple-valued input binary function. In the case that such a function has more than two multiple-valued variables then either the presented below approach can be applied to them or the multidimensional Walsh transform can be used [21]. The latter approach is similar to pattern analysis and image processing and is not considered in this Dissertation.

IV.2 MULTIPLE-VALUED INPUT BINARY FUNCTIONS

A multiple-valued input binary function called a (*binary function* for short) is a mapping $f(X_1, X_2, \dots, X_n): P_1 \times P_2 \times \dots \times P_n \rightarrow B$, where X_i is a multiple-valued variable that takes the values from the set $P_i = \{0, 1, \dots, p_i - 1\}$ and $B = \{0, 1, -\}$ (where $-$ denotes a don't care value). This is then the generalization of an ordinary n -input incompletely specified Boolean function $f: B^n \rightarrow B$.

A *literal* of multiple-valued input variable X_i , denoted by $X_i^{S_i}$, is defined as follows:

$$X_i^{S_i} = \begin{cases} 1 & \text{if } X_i \in S_i \\ 0 & \text{if } X_i \notin S_i \end{cases} \quad (\text{IV.1})$$

where $S_i \subseteq P_i$.

A *product of literals*, $X_1^{S_1}, X_2^{S_2}, \dots, X_k^{S_k}$, ($k \leq n$) is referred to as a *product term* (also called as *term* for short). A product term that includes literals for all function variables $X_1, X_2, X_3, \dots, X_n$ is called a *full term*. A *minterm* of a multiple-valued input binary function is a full term in which every set S_i reduces to a single logical value. The logical function has value 1 for a *true minterm*, value 0 for a *false minterm* and is not specified for a *don't care minterm*. A sum of products is denoted as a *sum-of-products expression (SOPE)* while a product of sums is called as a *product-of-sums expression*

(POSE).

IV.3 DEFINITIONS AND BASIC PROPERTIES OF TWO-DIMENSIONAL MAPS FOR MULTIPLE-VALUED INPUT BINARY FUNCTIONS

Two spectral representations, S and R , are introduced in the next section for multiple-valued input binary functions. The analogous definitions exist for Boolean functions (see Chapter III). In order to apply new spectral representations to binary functions, they are first represented by the set of two-dimensional maps on which the Walsh type transforms are performed. The auxiliary algorithm allowing any multiple-valued input binary function to be represented by a set of two-dimensional maps will be introduced.

The approach presented in this Chapter is not the only one possible for finding new spectral representations of the multiple-valued input binary functions. For example, instead of presenting each multiple-valued input binary function in the form of two-dimensional maps it is possible to use high-dimensional Hadamard matrices and transforms [56]. The approach presented however, enables the application of two-dimensional transforms to two-dimensional maps and, as a result, each spectral coefficient has an easy interpretation in logic terms. Moreover, the spectra can be calculated by known fast Walsh algorithms [22], [24], [25] and [56], and by the method presented in Chapter V.

The following symbols will be used in the presentation. Let n denote the number of different variables of a completely or incompletely specified multiple-valued input binary function. Let p_m denote the number of different logical values that can be assigned to any of the variables. It is obvious, that there exists only one such p_m that is maximal for the entire set of input variables, and that each input variable can take a different number of logical values.

The following properties are valid for any multiple-valued input binary function

and can be proven by mathematical induction.

Property IV.1: Any multiple-valued input binary function can be represented by $\binom{n}{2}$ two-dimensional maps where each map has, as the coordinates, two different variables from the set of all variables and the dimension of $p_i \times p_j$ where p_i and p_j are the maximal number of logical values that can be assumed by two different variables X_i and X_j , accordingly.

Property IV.2: A full term of a multiple-valued input binary function of n variables is represented by $\binom{n}{2}$ two-dimensional maps. The number of cells (areas on these map) that correspond to the full term can be found according to the formula $\sum_{j=1}^{n-1} \left[m_j \sum_{i=j+1}^n m_i \right]$ where m_i is the number of logical values of the i -th literal. This property is valid for any term. However, for variables that are not included in this term, one has to take the value of p_i instead of m_i in the above formula for each variable X_i .

Property IV.3: A minterm of a multiple-valued input binary function of n variables is represented by $\binom{n}{2}$ two-dimensional maps and on each of these maps the minterm is represented by one cell. Hence the number of cells that correspond to such a minterm is equal to the number of two-dimensional maps.

Property IV.4: The two-dimensional $p_m \times p_n$ maps described above can be transformed to partial spectral coefficients by an orthogonal Walsh-type transform without loss of any information if both numbers p_m and p_n are some powers of 2, possibly different.

Let us observe, that while in classical tables for multiple-valued logic [39] and [99] each minterm corresponds to a cell, in the proposed representation each minterm is represented by a set of cells, one cell from each map. The last requirement (Property IV.4) for the dimension of the map representing the function is due to the known ways of generating Hadamard matrices and to the requirements for the orthogonality of such matrices [81]. Since such a requirement would limit the possible number of different

logical values for input variables, the more general approach is presented below.

Proposition IV.1: The two-dimensional $p_m \times p_n$ map is expanded to the two-dimensional $p_m^* \times p_n^*$ map where p_m and p_n are any integer values describing the multiplicity of logical values of input variables, $p_m^* = p_m + 4 - (p_m \bmod 4)$ and $p_n^* = p_n + 4 - (p_n \bmod 4)$ accordingly. When the map is expanded all of the introduced cells have don't care values.

Proposition IV.2: The number of additional don't care cells that have to be added during the expansion process of the two-dimensional $p_m \times p_n$ map is equal to

- a) $p_n [4 - (p_m \bmod 4)]$ when $(p_n \bmod 4) = 0$ and $(p_m \bmod 4) \neq 0$
- b) $p_m [4 - (p_n \bmod 4)]$ when $(p_m \bmod 4) = 0$ and $(p_n \bmod 4) \neq 0$
- c) $p_n [4 - (p_m \bmod 4)] + p_m [4 - (p_n \bmod 4)]$
 $+ [4 - (p_m \bmod 4)] [4 - (p_n \bmod 4)]$
 when both $(p_m \bmod 4) \neq 0$ and $(p_n \bmod 4) \neq 0$.

The following algorithm describes how an arbitrary multiple-valued input binary function can be represented by the set of two-dimensional maps. It is assumed in the following description that the function is represented in the *SOPE* form. The dual algorithm can be derived for the function represented in the *PGSE* form. It is obvious, that the algorithm can be applied to the binary function represented in the form of cubes as well.

Algorithm IV.1: Transformation of multiple-valued input binary function in SOPE form to the set of two-dimensional matrices of any dimensions.

1. Set nl (number of literals in the binary function) and nt (number of terms in *SOPE* form).
2. For each pair of literals of the function create a two-dimensional $p_{m_i} \times p_{m_j}$ map where p_{m_i} and p_{m_j} are the maximal values of the i -th and j -th literal. The number of such maps is equal to $\binom{nl}{2}$ (Property IV.1).

3. For each term from the *SOPE* expression enter the true values into some cells of the two-dimensional maps. The total number of such cells in the maps having true values can be found for each term using Property IV.2. If step 3 has been performed for each term (i.e., nt times) then stop.

Hence, by using Algorithm IV.1, one can represent any multiple-valued input binary function in the form of the set of two-dimensional maps. In general, both dimensions of such maps are equal to any integer numbers. Due to Property IV.4, the additional algorithm converts a two-dimensional map to its equivalent (from the point of view of the logic function this map is representing) that has the dimensions equal to 2^j where j is any integer number.

Algorithm IV.2: Conversion of any two-dimensional map having one or two dimensions different from a power of 2 to its logical equivalent having dimensions that are powers of 2.

1. Set the value k - the number of two-dimensional maps for a given multiple-valued input binary function.
2. For each map do:
 - $k = k - 1$.
 - If either p_m or p_n or both these values are not some powers of 2 then modify either p_m or p_n or both to either p_m^* or p_n^* or both using Proposition IV.1.
 - If there was any modification of the dimensions of the map then fill the expanded map's cells with don't cares, and the number of such cells can be calculated using Proposition IV.2.
3. If $k = 0$ then stop otherwise go to step 2.

The following proposition deals with the dimension of the transform matrix that is necessary to calculate the spectrum of a two-dimensional map. In the case when

Algorithm IV.2 has not expanded the original map's dimension (dimensions) then instead of p_m^* or p_n^* the original values p_m and p_n should be used in the following formula.

Proposition IV.3: The Hadamard-Walsh matrix T of the dimension $2^n \times 2^n$ can transform the two-dimensional map having the dimension $p_m^* \times p_n^*$ iff $n = \log_2 \left[p_m^* p_n^* \right]$.

The application of both algorithms will be shown in the following example.

Example IV.1:

Consider the following three-variable multiple-valued input binary function:

$$f_1 = X^{(0)} Y^{(1)} + X^{(1,2)} Y^{(0,3)} Z^{(1,2,3)} \quad (\text{IV.2})$$

Then, according to Algorithm IV.1 $nl = 3$ and $nt = 2$. The three two-dimensional maps for this binary function that correspond to each pair of the variables are shown in Figure 9. (the areas on the maps filled with ones and zeros only). For instance, the term $X^{(0)} Y^{(1)}$ corresponds on the map $X Y$ to the cell $X^{(0)} Y^{(1)}$, on the map $Y Z$ to the cells $Y^{(1)} Z^{(0,1,2,3)}$, and on the map $X Z$ to the cells $X^{(0)} Z^{(0,1,2,3)}$. Then all these areas are filled with ones. Originally, according to Algorithm IV.1 the dimensions of these maps are 4×3 , 4×4 and 3×4 . After the application of Algorithm IV.2 the dimension of the first and third maps have increased to 4×4 and the expanded areas are filled with don't cares. The final result is shown in Figure 9.

	X				
		0	1	2	3
Y		0	1	1	-
0					
1		1	0	0	-
2		0	0	0	-
3		0	1	1	-

	Z				
		0	1	2	3
Y		0	1	1	1
0					
1		1	1	1	1
2		0	0	0	0
3		0	1	1	1

	Z				
		0	1	2	3
X		1	1	1	1
0					
1		0	1	1	1
2		0	1	1	1
3		-	-	-	-

$$f = X^{1,2} Y^{0,3} Z^{1,2,3} \oplus X^0 Y^1$$

Figure 9. Set of two-dimensional maps for 3-variable binary function.

Hence, by using Algorithm IV.1 and Algorithm IV.2 one can always represent a multiple-valued input binary function in the form of a set of two-dimensional maps having the dimensions equal to powers of 2. In the next section the application of Walsh-type transforms to these two-dimensional maps will be shown.

IV.4 BASIC PROPERTIES OF HADAMARD-WALSH SPECTRA S AND R FOR TWO-DIMENSIONAL MAPS REPRESENTING MULTIPLE-VALUED INPUT BINARY FUNCTIONS

In order to shorten the notation and make it similar to that of other authors, it is assumed that the symbol n that is used in what follows conforms to the requirements of Proposition IV.3. The properties of Walsh spectral coefficients for Boolean functions presented in Section III.2 will be rewritten in what follows to conform to the properties that are valid for the spectral representation of the multiple-valued input binary functions presented in this Chapter. Since possible alternative spectra of such binary functions can be based on high-dimensional Hadamard matrices [56] the Hadamard order of Walsh functions is used in this presentation. Such an approach allows the alternative spectrum of binary functions to be derived from the presented spectra. Since both spectra are based on the same ordering there is no need to do additional conversion operations between orderings. Since the spectrum based on high-dimensional Hadamard matrices has a non-minimum number of spectral coefficients only the spectrum based on the concept of two-dimensional maps is presented in this Chapter. The properties of Hadamard-Walsh spectra of such maps follow.

The Hadamard-Walsh S spectrum of a two-dimensional map is an alternative representation of the map. When the map is represented as a vector V formed of consecutive rows the Hadamard-Walsh S spectrum is formed by the multiplication of the $\langle +1, 0, -1 \rangle$ vector representation V^S (corresponding to the original vector V for an incompletely specified map) by a $2^n \times 2^n$ Hadamard-Walsh matrix (Chapter II). In the coding scheme, the conventional $\langle 0, 1, - \rangle$ values correspond to $\langle +1, -1, 0 \rangle$ coding, respectively ($-$ stands for a don't care). In the case of a completely specified two-dimensional map the conventional $\langle 0, 1 \rangle$ values correspond to $\langle +1, -1 \rangle$ coding only.

If one keeps the original coding scheme then the alternative R spectrum can be defined. The Hadamard-Walsh R spectrum of a two-dimensional map is an alternative representation of the map. When the map is represented as a vector V formed of consecutive rows, then the Hadamard-Walsh spectrum R is formed from the multiplication of the $\langle 0, 1, 0.5 \rangle$ vector representation V^R (corresponding to the original vector V for an incompletely specified map) by a $2^n \times 2^n$ Hadamard-Walsh matrix T . In the coding scheme, the conventional $\langle 0, 1, - \rangle$ values correspond to $\langle 0, 1, 0.5 \rangle$ coding, respectively.

The principal properties of the R and S spectra for two-dimensional maps are described below. It will be assumed without loss of generality that each map has two 4-valued variables as the coordinates, denoted in this description as X , - the horizontal variable and Y , - the vertical variable, accordingly. Also, for simplicity, instead of using the full set notation for the description of the multiple-valued literals, only the members of the set will be denoted. For example, the literal $X^{(1,3)}$ will be described as $X^{1,3}$ and the same abbreviation in the notation for spectral coefficients will be used as well. When the properties of the spectral coefficients of both the S and R spectra are the same, such properties will be given for the coefficients of the S spectrum only and this fact will be noted in the description of the property. When these properties differ then both spectra will be described separately.

The new properties of the spectra for two-dimensional map representation of the multiple-valued input binary functions given below can be derived from the properties of the spectral coefficients describing Boolean functions that were described in Chapter III. The names of transforms used below, refer, of course, to the *multiple-valued* counterparts of the respective transforms known in the literature as Walsh-Kaczmarz, Walsh-Paley, Rademacher-Walsh, and Hadamard-Walsh. Only these four basic orderings are compared. The transform matrices for each of these four basic orderings are the same for multiple-valued input binary functions and Boolean functions. The former are described

by the vector of spectra of each two-dimensional map (where each of such maps is treated as a separate two-variable binary function), while the latter is described by only one spectrum (a Boolean function can be treated as only one two-variable binary function and be represented by only one two-dimensional map).

$$\begin{array}{c}
 \text{T} \\
 \left[\begin{array}{cccccccccccccccc}
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\
 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\
 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\
 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\
 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\
 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 \\
 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1
 \end{array} \right]
 \end{array}$$

$$\begin{array}{c}
 \text{v}^R \\
 \left[\begin{array}{c}
 0 \\
 1 \\
 1 \\
 1 \\
 1 \\
 1 \\
 1 \\
 1 \\
 1 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 1 \\
 1 \\
 1 \\
 1
 \end{array} \right]
 \end{array}
 =
 \begin{array}{c}
 \text{R} \\
 \left[\begin{array}{c}
 10 \quad r_0 \\
 -2 \quad r_{z^2,3} \\
 -2 \quad r_{z^1,3} \\
 -2 \quad r_{z^1,2} \\
 -4 \quad r_{y^1,3} \\
 0 \quad r_{z^2,3} \oplus y^1,3 \\
 0 \quad r_{z^1,3} \oplus y^1,3 \\
 0 \quad r_{z^1,2} \oplus y^1,3 \\
 4 \quad r_{y^2,3} \\
 0 \quad r_{z^2,3} \oplus y^2,3 \\
 0 \quad r_{z^1,3} \oplus y^2,3 \\
 0 \quad r_{z^1,2} \oplus y^2,3 \\
 2 \quad r_{y^1,2} \\
 -2 \quad r_{z^2,3} \oplus y^1,2 \\
 -2 \quad r_{z^1,3} \oplus y^1,2 \\
 -2 \quad r_{z^1,2} \oplus y^1,2
 \end{array} \right]
 \end{array}$$

Figure 10. Spectrum R for completely specified map of Z and Y variables.

Properties describing spectra of Boolean functions are mainly based on the orthogonality of the transform matrices (Chapter III). When properties describing multiple-valued input binary functions are derived then the restrictions on the dimensions of the transform matrices have to be considered as well.

IV.5 The transform matrix is complete and orthogonal, and therefore, there is no information lost in the S and R spectra, concerning the cells of the map.

- IV.6 Only the Hadamard-Walsh matrix has the *recursive Kronecker product structure* [22], [24], [56], [68] and [82] and for this reason is preferred over other possible variants of the Walsh transform, known in the literature as Walsh-Kaczmarz, Rademacher-Walsh, and Walsh-Paley transforms.
- IV.7 Out of the four considered orderings of Walsh functions, only the Rademacher-Walsh transform is not *symmetric* ; all other variants of Walsh transforms are symmetric, so that, disregarding a scaling factor, the same matrix can be used for both the forward and inverse transform operations.
- IV.8 Each spectral coefficient s_I (as well as r_I) gives a *correlation value* between the two-variable input binary function F corresponding to a given two-dimensional map and a *standard trivial function* u_I corresponding to this coefficient. The standard trivial functions for the spectral coefficients are, respectively, for the *dc coefficient* (direct current coefficient) - the universe of the function (where all cells on the map have true value) denoted by u_0 ; for the coefficients $s_{X^{1,2}}, s_{X^{2,3}}, s_{Y^{1,2}}, s_{Y^{2,3}}$ etc. (first order coefficients) - the literals $X^{1,2}, X^{2,3}, Y^{1,2}, Y^{2,3}$ of the binary function shown on the map and denoted by $u_{X^{i,j}}, u_{X^{j,k}}, u_{Y^{i,j}}, u_{Y^{j,k}}$; for the coefficients $s_{X^{1,2} \oplus Y^{2,3}}, s_{X^{2,3} \oplus Y^{1,2}}, s_{X^{2,3} \oplus Y^{2,3}}, s_{X^{1,2} \oplus Y^{1,2}}$ etc. (second order coefficients) - the exclusive-or function between literals $X^{1,2} \oplus Y^{2,3}, X^{2,3} \oplus Y^{1,2}, X^{2,3} \oplus Y^{2,3}, X^{1,2} \oplus Y^{1,2}$ of the binary function shown on the map and denoted by $u_{X^{i,j} \oplus Y^{j,k}}$ or by $u_{X^{i,j} \oplus Y^{i,j}}$. In all the formulas, i, j , and k are different integer numbers, $i = 1, j = 2, k = 3$. In short, the dc coefficient can be denoted by s_I ($I = 0$), first order coefficients by s_{L^i} ($I = i, j, i \neq 0, j \neq 0, i \neq j$, and L is a literal), second order coefficients by $s_{L_1^i \oplus L_2^j}$ ($I = i, j, i \neq 0, j \neq 0, i \neq j$, and L_1, L_2 are two different literals).
- IV.9 The sum of all spectral coefficients of the S spectrum for any completely specified two-dimensional map is $\pm 2^n$.

- IV.10 The sum of all spectral coefficients of the S spectrum for any incompletely specified two-dimensional map is not $\pm 2^n$.
- IV.11 The maximal (minimal) value of any individual spectral coefficient of spectrum S is $\pm 2^n$. This happens when the binary function represented in a given two-dimensional map is equal to either a standard trivial function u_I (sign +) or to its complement (sign -). In either case, all the remaining spectral coefficients have zero values because of the orthogonality of the transform matrix T .
- IV.12 The maximal (minimal) value of any but r_0 individual spectral coefficient r_I is $\pm 2^{n-1}$. This happens when the binary function represented on a given two-dimensional map is equal to either a standard trivial function u_I (sign -) or to its complement (sign +). In either case, all but r_0 remaining spectral coefficients have zero values because of the orthogonality of the transform matrix T .
- IV.13 The maximal value of the r_0 spectral coefficient is 2^n . It happens when all the cells of the two-dimensional map have the logical value 1.
- IV.14 Each standard trivial function u_I , except u_0 , corresponding to a two-dimensional map has the same number of true and false minterms equal to 2^{n-1} .
- IV.15 The S spectrum of each true cell of a two-dimensional map is given by $s_0 = 2^n - 2$, and all remaining $2^n - 1$ spectral coefficients s_I are equal to ± 2 .
- IV.16 The S spectrum of each don't care cell of a two-dimensional map is given by $s_0 = 2^n - 1$, and all remaining $2^n - 1$ spectral coefficients s_I are equal to ± 1 .
- IV.17 The S spectrum of each false minterm of a two-dimensional map is given by $s_I = 0$.

Example IV.2:

An example of the calculation of the R spectrum of a completely specified two-dimensional map is shown in Figure 10. The calculation of the S spectrum for the same

$$\begin{array}{c}
 \text{T} \\
 \left[\begin{array}{cccccccccccccccc}
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\
 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\
 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\
 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\
 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\
 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 \\
 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1
 \end{array} \right]
 \begin{array}{c}
 \text{v}^R \\
 \left[\begin{array}{c}
 0 \\
 1 \\
 1 \\
 0.5 \\
 1 \\
 0 \\
 0 \\
 0.5 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0.5 \\
 0 \\
 1 \\
 1 \\
 1 \\
 0.5
 \end{array} \right]
 =
 \begin{array}{c}
 \text{R} \\
 \left[\begin{array}{c}
 7 \\
 -1 \\
 -1 \\
 -1 \\
 -1 \\
 -1 \\
 -1 \\
 -1 \\
 1 \\
 1 \\
 1 \\
 1 \\
 1 \\
 3 \\
 -1 \\
 -1 \\
 -1 \\
 -5
 \end{array} \right]
 \begin{array}{c}
 r_0 \\
 r_{z^2,3} \\
 r_{z^1,3} \\
 r_{z^1,2} \\
 r_{y^1,3} \\
 r_{z^2,3} \oplus y^1,3 \\
 r_{z^1,3} \oplus y^1,3 \\
 r_{z^1,2} \oplus y^1,3 \\
 r_{y^2,3} \\
 r_{z^2,3} \oplus y^2,3 \\
 r_{z^1,3} \oplus y^2,3 \\
 r_{z^1,2} \oplus y^2,3 \\
 r_{y^1,2} \\
 r_{z^2,3} \oplus y^1,2 \\
 r_{z^1,3} \oplus y^1,2 \\
 r_{z^1,2} \oplus y^1,2
 \end{array}
 \end{array}
 \end{array}$$

Figure 12. Spectrum R for incompletely specified map of X and Y variables.

Recursive algorithms, data flow-graph methods and parallel calculations similar to the Fast Fourier Transform [22], [24], [25], [56], [82] and [88] can be used to calculate the transforms introduced above. The advantages of calculating Walsh spectra based on the cube representation of logic functions are discussed in Chapter V. The corresponding algorithms for Boolean functions are presented in Chapter V as well. The Walsh spectrum of multiple-valued input binary functions described in this Chapter can also be calculated in the most efficient way also by using the disjoint cube representation of these functions. The extension of the method presented in Chapter V for the spectral representation of multiple-valued input binary functions is described in [92] and [103].

$$\begin{array}{c}
 \text{T} \\
 \left[\begin{array}{cccccccccccc}
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\
 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\
 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 \\
 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \\
 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 \\
 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \\
 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & 1 \\
 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\
 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 \\
 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 \\
 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & -1 \\
 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1
 \end{array} \right]
 \begin{array}{c}
 \text{VS} \\
 \left[\begin{array}{c}
 1 \\
 -1 \\
 -1 \\
 0 \\
 -1 \\
 1 \\
 1 \\
 0 \\
 1 \\
 1 \\
 1 \\
 0 \\
 1 \\
 -1 \\
 -1 \\
 0
 \end{array} \right]
 =
 \begin{array}{c}
 \text{S} \\
 \left[\begin{array}{c}
 2 \\
 2 \\
 2 \\
 2 \\
 2 \\
 2 \\
 2 \\
 2 \\
 -2 \\
 -2 \\
 -2 \\
 -2 \\
 -6 \\
 2 \\
 2 \\
 10
 \end{array} \right]
 \begin{array}{c}
 s_0 \\
 s_{x^2,3} \\
 s_{x^1,3} \\
 s_{x^1,2} \\
 s_{y^1,3} \\
 s_{x^2,3} \oplus y^{1,3} \\
 s_{x^1,3} \oplus y^{1,3} \\
 s_{x^1,2} \oplus y^{1,3} \\
 s_{y^2,3} \\
 s_{x^2,3} \oplus y^{2,3} \\
 s_{x^1,3} \oplus y^{2,3} \\
 s_{x^1,2} \oplus y^{2,3} \\
 s_{y^1,2} \\
 s_{x^2,3} \oplus y^{1,2} \\
 s_{x^1,3} \oplus y^{1,2} \\
 s_{x^1,2} \oplus y^{1,2}
 \end{array}
 \end{array}
 \end{array}$$

Figure 13. Spectrum S for incompletely specified map of X and Y variables.

IV.5 LINKS BETWEEN SPECTRAL AND CLASSICAL LOGIC DESIGN

The material presented in this section is valid, not only for two-dimensional maps representing multiple-valued input binary functions, but for Boolean functions and their Karnaugh map representations as well. If the latter is the case then in all of the following formulas n corresponds to the number of variables of the Boolean function and the two-dimensional map corresponds to a Karnaugh map rewritten from Gray-code to straight binary code (in the case of a 4×4 dimensioned map, the second and third rows have to be mutually interchanged - the same applies to the second and third columns). For multiple-valued input binary functions n fulfills the requirements of Proposition IV.3 and

when both p_m^* and p_n^* from this proposition are equal then n represents the number of different logical values that can be assumed by each of the literals of the binary function. The meaning of all other symbols that are going to be introduced below is the same for both Boolean functions and two-dimensional maps.

Hence, let us show more clearly, in classical logic terms, what the real meaning of the spectral coefficients is for each map. The following symbols will be used. Let a_I be the number of true cells in the two-dimensional map, where both the map and the standard trivial function u_I have the logic values of 1; let b_I be the number of false cells in the two-dimensional map, where the map has the logic value 0 and the standard trivial function u_I has the logic value 1; let c_I be the number of true cells in the two-dimensional map, where the map has the logic value 1 and the standard trivial function u_I has the logical value 0; let d_I be the number of false cells in the two-dimensional map, where both the map and the standard trivial function u_I have the logic values 0, let e_I be the number of don't care cells in the two-dimensional map, where the standard trivial function u_I has the logic value 1, and let f_I be the number of don't care minterms of a two-dimensional map, where the standard trivial function u_I has the logic value 0. Then, for a completely specified $n \times n$ two-dimensional map, these formulas hold:

$$a_I + b_I + c_I + d_I = 2^n \quad (\text{IV.3})$$

and

$$a_I + b_I = c_I + d_I = 2^{n-1}. \quad (\text{IV.4})$$

Accordingly, for an incompletely specified $n \times n$ two-dimensional map, these formulas hold:

$$a_I + b_I + c_I + d_I + e_I + f_I = 2^n \quad (\text{IV.5})$$

and

$$a_I + b_I + e_I = c_I + d_I + f_I = 2^{n-1}. \quad (\text{IV.6})$$

The s_I spectral coefficients for a completely specified two-dimensional map can be

defined as follows:

$$s_I = 2^n - 2 \times a_I, \quad (\text{IV.7})$$

when $I = 0$,

$$s_I = 2 \times (a_I + d_I) - 2^n. \quad (\text{IV.8})$$

when $I \neq 0$.

The spectral coefficients for an incompletely specified two-dimensional map can be defined as follows:

$$s_I = 2^n - 2 \times a_I - e_I, \quad (\text{IV.9})$$

when $I = 0$

and

$$s_I = 2 \times (a_I + d_I) + e_I + f_I - 2^n. \quad (\text{IV.10})$$

when $I \neq 0$.

As one can see, for the case when both $e_I = 0$, and $f_I = 0$, i.e., for the completely specified two-dimensional maps, equations (IV.9) and (IV.10) reduce to equations (IV.7) and (IV.8). And again, by easy mathematical transformations, one can define all spectral coefficients, except s_0 , as follows:

$$s_I = (a_I + d_I) - (b_I + c_I), \quad (\text{IV.11})$$

when $I \neq 0$.

The s_0 spectral coefficient can be rewritten as follows:

$$s_I = b_I - a_I. \quad (\text{IV.12})$$

Thus, in the final formulas, describing all s_I spectral coefficients, the number of don't care minterms $e_I + f_I$ can be eliminated. Moreover, the final formulas are exactly the same as the ones for the completely specified two-dimensional map. This is due to the fact that equation (IV.6) for the numbers a_I , b_I , c_I , d_I , e_I , and f_I links all these values together.

Let us show now the meanings of the r_I spectral coefficients. The meanings of all the symbols $a_I, b_I, c_I, d_I, e_I,$ and f_I are exactly the same as described previously.

The r_I spectral coefficients for a completely specified two-dimensional map can be defined as follows:

$$r_I = a_I, \quad (\text{IV.13})$$

when $I = 0,$

$$r_I = c_I - a_I, \quad (\text{IV.14})$$

when $I \neq 0.$

The spectral coefficients for an incompletely specified two-dimensional map can be defined as follows:

$$r_I = a_I + \frac{e_I}{2}, \quad (\text{IV.15})$$

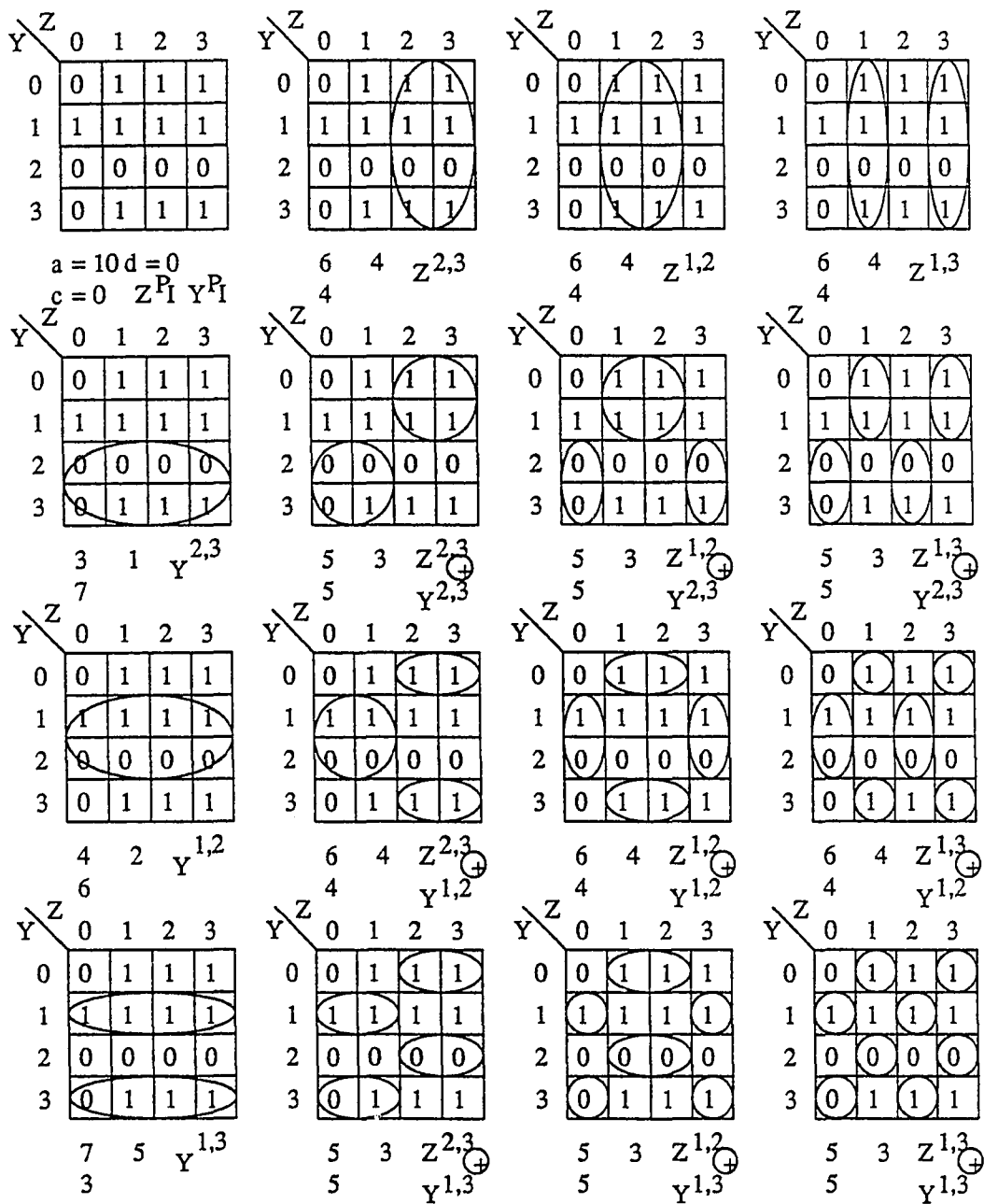
when $I = 0$

and

$$r_I = c_I - a_I + \frac{f_I - e_I}{2}, \quad (\text{IV.16})$$

when $I \neq 0.$

As one can see, for the case when $e_I = 0$ and $f_I = 0,$ i.e., for the completely specified two-dimensional map, equations (IV.15) and (IV.16) reduce to equations (IV.13) and (IV.14) presented previously for r_I spectral coefficients.



Note: $P_I = \{ 0, 1, 2, 3 \}$

Figure 14. Standard trivial functions for completely specified two-dimensional map.

The application of the above formulas will be shown in the following examples (all examples are for the spectra in Hadamard-Walsh order).

Example IV.3:

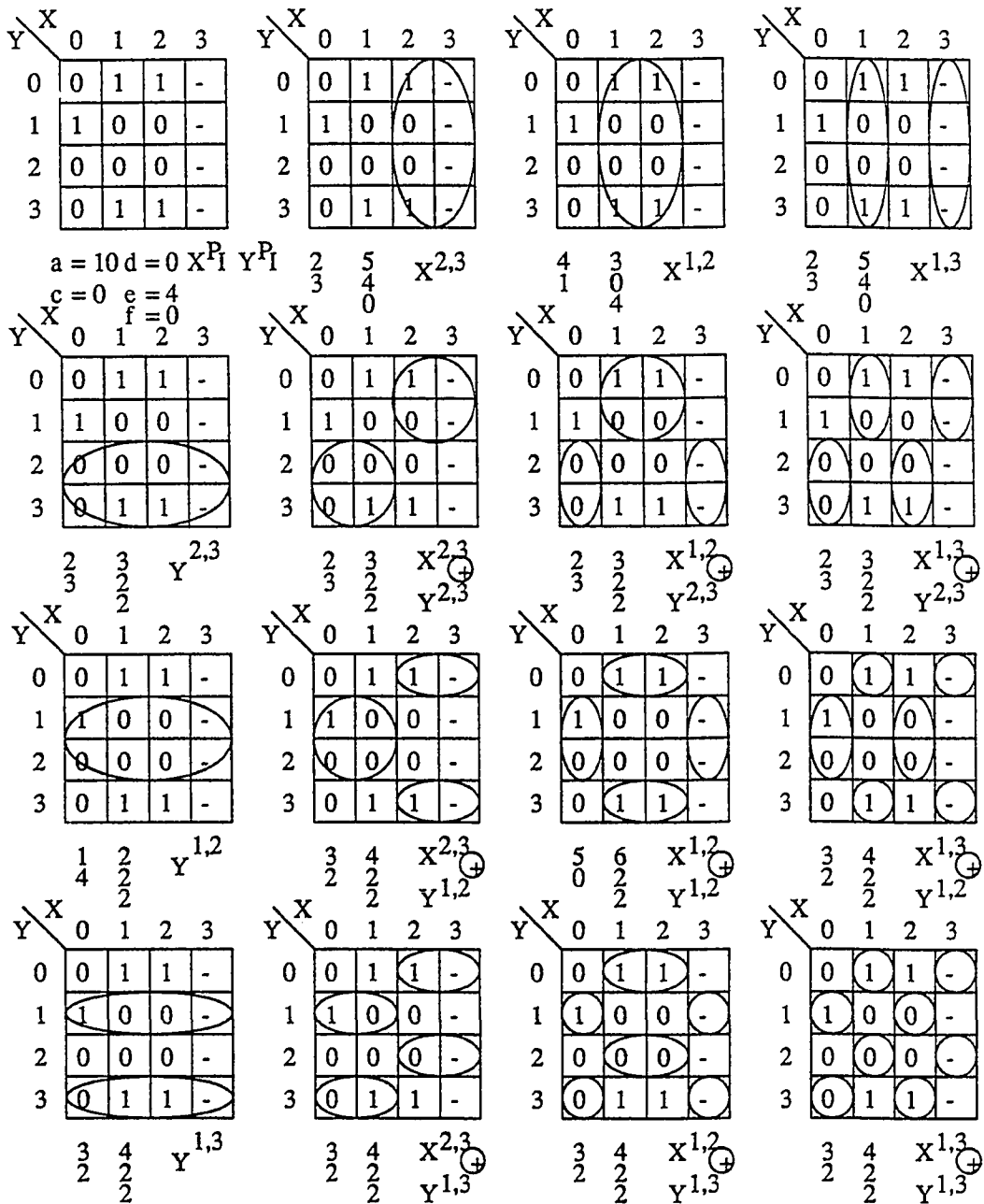
Consider the completely specified two-dimensional map describing the relationship between four-valued variables Y and Z for the binary function from Figure 9. All standard trivial functions and the corresponding values of a_I , c_I , and d_I for this map are shown in Figure 14.

The R spectrum for this map can be calculated using equations (IV.13) and (IV.14):

$$\begin{aligned}
 r_0 &= 10, r_{z^{2,3}} = 4 - 6 = -2, \\
 r_{z^{1,3}} &= 4 - 6 = -2, r_{z^{1,2}} = 4 - 6 = -2, \\
 r_{y^{1,3}} &= 3 - 7 = -4, r_{z^{2,3} \oplus y^{1,3}} = 5 - 5 = 0, \\
 r_{z^{1,3} \oplus y^{1,3}} &= 5 - 5 = 0, r_{z^{1,2} \oplus y^{1,3}} = 5 - 5 = 0, \\
 r_{y^{2,3}} &= 7 - 3 = 4, r_{z^{2,3} \oplus y^{2,3}} = 5 - 5 = 0, \\
 r_{z^{1,3} \oplus y^{2,3}} &= 5 - 5 = 0, r_{z^{1,2} \oplus y^{2,3}} = 5 - 5 = 0, \\
 r_{y^{1,2}} &= 6 - 4 = 2, r_{z^{2,3} \oplus y^{1,2}} = 4 - 6 = -2, \\
 r_{z^{1,3} \oplus y^{1,2}} &= 4 - 6 = -2, r_{z^{1,2} \oplus y^{1,2}} = 4 - 6 = -2.
 \end{aligned}$$

The S spectrum for this map can be calculated using equations (IV.7) and (IV.8):

$$\begin{aligned}
 s_0 &= 16 - 20 = -4, s_{z^{2,3}} = 20 - 16 = 4, \\
 s_{z^{1,3}} &= 20 - 16 = 4, s_{z^{1,2}} = 20 - 16 = 4, \\
 s_{y^{1,3}} &= 24 - 16 = 8, s_{z^{2,3} \oplus y^{1,3}} = 16 - 16 = 0, \\
 s_{z^{1,3} \oplus y^{1,3}} &= 16 - 16 = 0, s_{z^{1,2} \oplus y^{1,3}} = 16 - 16 = 0, \\
 s_{y^{2,3}} &= 8 - 16 = -8, s_{z^{2,3} \oplus y^{2,3}} = 16 - 16 = 0, \\
 s_{z^{1,3} \oplus y^{2,3}} &= 16 - 16 = 0, s_{z^{1,2} \oplus y^{2,3}} = 16 - 16 = 0, \\
 s_{y^{1,2}} &= 12 - 16 = -4, s_{z^{2,3} \oplus y^{1,2}} = 20 - 16 = 4, \\
 s_{z^{1,3} \oplus y^{1,2}} &= 20 - 16 = 4, s_{z^{1,2} \oplus y^{1,2}} = 20 - 16 = 4.
 \end{aligned}$$



Note: $P_I = \{ 0, 1, 2, 3 \}$

Figure 15. Standard trivial functions for incompletely specified two-dimensional map.

As one can find, the obtained R and S spectra are exactly the same as the ones calculated by the classical method shown in Figure 10 and Figure 11.

Example IV.4:

Consider the incompletely specified two-dimensional map describing the relationship between four-valued variables Y and X for the binary function from Figure 9. All standard trivial functions and the corresponding values of a_I , c_I , d_I , e_I , and f_I for this map are shown in Figure 15.

The R spectrum for this map can be calculated using equations (IV.15) and (IV.16):

$$r_0 = 5 + 2 = 7, r_{x^{2,3}} = 3 - 2 + \frac{0 - 4}{2} = 1 - 2 = -1,$$

$$r_{x^{1,3}} = 1 - 2 = -1, r_{x^{1,2}} = -3 + 2 = -1,$$

$$r_{y^{1,3}} = -1, r_{x^{2,3} \oplus y^{1,3}} = -1,$$

$$r_{x^{1,3} \oplus y^{1,3}} = -1, r_{x^{1,2} \oplus y^{1,3}} = -1,$$

$$r_{y^{2,3}} = 1, r_{x^{2,3} \oplus y^{2,3}} = 1,$$

$$r_{x^{1,3} \oplus y^{2,3}} = 1, r_{x^{1,2} \oplus y^{2,3}} = 1,$$

$$r_{y^{1,2}} = 3, r_{x^{2,3} \oplus y^{1,2}} = -1,$$

$$r_{x^{1,3} \oplus y^{1,2}} = -1, r_{x^{1,2} \oplus y^{1,2}} = -5.$$

The S spectrum for this map can be calculated using equations (IV.9) and (IV.10):

$$s_0 = 16 - 10 - 4 = 2, s_{x^{2,3}} = 14 + 4 - 16 = 2,$$

$$s_{x^{1,3}} = 14 + 4 - 16 = 2, s_{x^{1,2}} = 14 + 4 - 16 = 2,$$

$$s_{y^{1,3}} = 14 + 4 - 16 = 2, s_{x^{2,3} \oplus y^{1,3}} = 14 + 4 - 16 = 2,$$

$$s_{x^{1,3} \oplus y^{1,3}} = 14 + 4 - 16 = 2, s_{x^{1,2} \oplus y^{1,3}} = 14 + 4 - 16 = 2,$$

$$s_{y^{2,3}} = 10 + 4 - 16 = -2, s_{x^{2,3} \oplus y^{2,3}} = 10 + 4 - 16 = -2,$$

$$s_{x^{1,3} \oplus y^{2,3}} = 10 + 4 - 16 = -2, s_{x^{1,2} \oplus y^{2,3}} = 10 + 4 - 16 = -2,$$

$$s_{y^{1,2}} = 6 + 4 - 16 = -6, s_{x^{2,3} \oplus y^{1,2}} = 14 + 4 - 16 = 2,$$

$$s_{x^{1,3} \oplus y^{1,2}} = 14 + 4 - 16 = 2, s_{x^{1,2} \oplus y^{1,2}} = 22 + 4 - 16 = 10.$$

As one can find, the R and S spectra obtained are exactly the same as the ones calculated by the classical method shown in Figure 12 and Figure 13.

As the final example the S and R spectra for the third two-dimensional map representing the binary function f will be shown. This time, the calculations are not shown but can be performed by any of the methods already presented.

Example IV.5:

Consider the incompletely specified two-dimensional map describing the relationship between four-valued variables X and Z for the binary function from Figure 9.

The R spectrum for this map is as follows:

$$\begin{aligned}
 r_0 &= 12, r_{z^{2,3}} = -2, \\
 r_{z^{1,3}} &= -2, r_{z^{1,2}} = -2, \\
 r_{x^{1,3}} &= 2, r_{z^{2,3} \oplus x^{1,3}} = 0, \\
 r_{z^{1,3} \oplus x^{1,3}} &= 0, r_{z^{1,2} \oplus x^{1,3}} = 0, \\
 r_{x^{2,3}} &= 2, r_{z^{2,3} \oplus x^{2,3}} = 0, \\
 r_{z^{1,3} \oplus x^{2,3}} &= 0, r_{z^{1,2} \oplus x^{2,3}} = 0, \\
 r_{x^{1,2}} &= 0, r_{z^{2,3} \oplus x^{1,2}} = 2, \\
 r_{z^{1,3} \oplus x^{1,2}} &= 2, r_{z^{1,2} \oplus x^{1,2}} = 2.
 \end{aligned}$$

The S spectrum for this map is as follows:

$$\begin{aligned}
 s_0 &= -8, s_{z^{2,3}} = 4, \\
 s_{z^{1,3}} &= 4, s_{z^{1,2}} = 4, \\
 s_{x^{1,3}} &= -4, s_{z^{2,3} \oplus x^{1,3}} = 0, \\
 s_{z^{1,3} \oplus x^{1,3}} &= 0, s_{z^{1,2} \oplus x^{1,3}} = 0, \\
 s_{x^{2,3}} &= -4, s_{z^{2,3} \oplus x^{2,3}} = 0, \\
 s_{z^{1,3} \oplus x^{2,3}} &= 0, s_{z^{1,2} \oplus x^{2,3}} = 0, \\
 s_{x^{1,2}} &= 0, s_{z^{2,3} \oplus x^{1,2}} = -4,
 \end{aligned}$$

$$s_{z^{1,3} \oplus x^{1,2}} = -4, s_{z^{1,2} \oplus x^{1,2}} = -4.$$

Then the multiple-valued input binary function from the Figure 9 is represented by a vector which is composed of three sets of spectral coefficients. The values of all three spectra are given in Example IV.3, Example IV.4, and Example IV.5.

CHAPTER V

EFFECTIVE COMPUTER METHODS FOR THE CALCULATION OF THE RADEMACHER-WALSH SPECTRUM FOR COMPLETELY AND INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS

V.1 DESCRIPTION

In this Chapter a *new computer method* for the calculation of the Rademacher-Walsh spectrum of incompletely specified Boolean functions is introduced. The method can calculate a Walsh spectrum of any ordering since the algorithm is independent of the ordering of the spectrum. Since a direct linear relationship exists between the R and S spectra described by equations (III.4) and (III.5) then this Chapter uses mainly the S spectrum. The *computer method* has been implemented in the DIADES automation system [14], [89], [90] and [92].

The method presented in this Chapter can be regarded as representative of a whole family of two kinds of methods and approach that is presented here can be easily adapted to other transforms used in digital logic design. For example, the adaptation of the *method* for Adding and Arithmetic Transforms was described in [104], while the adaptation of the same *method* for the Reed-Muller Transform is presented in Chapter VII. The *method* is also universal for multiple-valued binary functions and its extension to Walsh spectra of such functions was presented in [103]. The *method* was also applied to the Reed-Muller transform of multiple-valued binary functions in [92].

The algorithm presented can be applied to both completely and incompletely specified Boolean functions. It operates on the disjoint cube representation of a Boolean

function. By this approach, each spectral coefficient can be calculated separately or all of the coefficients can be calculated in parallel. These advantages are absent in existing methods.

In order to use Boolean functions that are represented as arrays of nondisjoint cubes, an additional fast algorithm to generate disjoint cubes is presented. Use of the disjoint cubes representation of Boolean functions has been found advantageous in many algorithms used in digital logic design [3], [18], [37]-[39], [54] and [108].

The theory introduced for calculating the spectral coefficients is new. The method presented also allows for the calculation of the spectra of a system of Boolean functions. When the system of incompletely specified Boolean functions is processed there is a restriction that all the functions in the system are assumed to be undefined at exactly the same points (minterms of a Karnaugh map). Optimal completion of don't care minterms for such a system of Boolean functions from the point of view of a minimal number of spectral coefficients different from 0 (in order to obtain a simpler implementation of such a system of functions) was presented in [15]. However, this is a large restriction. The method presented here can not only process such functions but can also deal with any system of completely or incompletely specified Boolean functions. Each function in the system of functions processed by the method can have a don't care minterm anywhere in the function's domain.

V.2 ALGORITHM TO GENERATE DISJOINT CUBES

A new algorithm is shown that generates a representation of completely or incompletely specified Boolean functions in the form of arrays of disjoint ON- and DC- (if any) cubes or an array of disjoint OFF-cubes. A peculiar feature of the algorithm, which speeds up its execution, is the fact that in comparison to known algorithms it minimizes the number of cube calculus operations. The algorithm introduced here generates a dis-

joint cubes representation of a Boolean function. In such a representation each Boolean function is shown in the form of a set of arrays of disjoint cubes that completely covers the function.

The new algorithm can be applied to completely as well as to incompletely specified Boolean functions. As the input data, the Boolean functions are represented in the form of arrays of nondisjoint ON- cubes (in the case of a completely specified Boolean function) and ON- and DC- cubes or ON- and OFF- cubes (in the case of incompletely specified Boolean functions). An ON- array of the Boolean function F corresponds directly to a sum-of-products expression (SOPE) of F (the cubes are composed of true minterms). A DC- array (an array of don't care cubes) has cubes that include minterms where function F evaluates to either 0 or 1 (don't care minterms). An OFF- array includes cubes of all minterms, where the function F evaluates to 0 (false minterms). The computer algorithms that operate on incompletely specified Boolean functions represent them either as arrays of ON- and DC- cubes or as arrays of ON- and OFF- cubes [3], [54], [29] and [30].

The algorithm introduced here converts an array of nondisjoint cubes to an array of disjoint cubes. The main motivation for this part of the research was to create a very fast preprocessing method to generate input data for the new algorithm calculating the Rademacher-Walsh spectrum of Boolean functions directly from disjoint ON- and DC- cubes rather than minterms (Section V.3). Hence, the new algorithm for the generation of the disjoint cubes representation of Boolean functions is important for both classical [3], [43] and [54] and spectral [15], [21], [22], and [25] techniques of designing digital logic. Similar preprocessing algorithms were described and used previously in the PALMINI [37], UMINI [38], EXORCISM [18], and EXORCISM-MV [39] programs, but the new algorithm is more efficient since it does not use sorting and minimizes the number of cube calculus operations used.

Reference [38] discusses applications of minimally split product implicants in logic synthesis. Table 1 in [38] presents the ratio of the number of minimally split product implicants to the number of minterms for each function from the MCNC benchmark. This ratio varies from 0.008 for function t3 to 1 for function pla4. However, for large functions this ratio is significantly smaller than 1. This results from the method of generation of the minimally split product implicants [38] that their number is usually much larger than the number of disjoint cubes, which is close to the number of prime implicants in a minimal cover. Therefore, for most Boolean functions, the number of disjoint cubes is much smaller than the number of minterms in the classical algorithms used to generate spectra.

The same new algorithm for creating disjoint cubes can be applied separately to an array of ON- cubes, an array of DC- cubes, and an array of OFF- cubes when the last array is used in the description of Boolean functions. The only drawback of the new algorithm is the fact that the generated arrays of cubes do not always have the largest available disjoint cubes as the solutions. However, this feature is not important in the case of the programs mentioned above which can use this algorithm as a preprocessor for their data representation of Boolean functions.

It is assumed, that the data for the algorithm is a logical function represented in the array of cubes form. It is enough to have any two of the three possible arrays of cubes. It will now be demonstrated how, from the nondisjoint set of cubes, one is able to generate a disjoint set of cubes, by applying a sequence of well-known operations on arrays of cubes [3], [43] and [54]. Cube operations of disjoint sharp, absorption, and intersection are used in the algorithm.

The following notation will be used in the description of the new algorithm. During the execution of the main loop, every time two cubes c_a and c_b are compared. The cubes describing a given Boolean function are stored in an array A . The pointer a indi-

cates the position of the cube c_a in the array A which is the first in the pair of cubes being compared, the pointer b indicates the position of the second cube in the pair. Note, that the pointer a changes its values from 1 to $cu - 1$ (where cu is the current number of cubes in the array) and the pointer b changes its values from cu to 2 accordingly. At any given moment, the value of the pointer b is always greater by at least 1 than the value of the pointer a . During the execution of the algorithm the content of the array A changes dynamically and the final number of cubes in this array can be different from the original number of cubes. That is due to the fact that two cube operations used in the algorithm have the following properties : 1) the disjoint sharp operator (denoted by $\#_j$) can generate more than one cube as its result, 2) absorption can remove some cubes and decrease by that the total number of cubes.

Algorithm V.1: Generation of disjoint cubes.

Symbols:

a, b : pointers to two different cubes,

$cube_a, cube_b$: determine the cubes pointed to by a , and b in the cube list,

d : number of solution cubes generated by the disjoint sharp operation,

m : number of cubes in the cube list before entering a new loop,

cu : number of cubes during execution of the loop.

so : final number of disjoint cubes.

Algorithm:

step 1. set a and b to the following positions in the cube list :

$a := 1, b := nc, m := cu$

where cu is the initial number of cubes in the cube list.

step 2. Main loop :

For each pair of cubes $cube_a$ and $cube_b$ from an array A do:

if an intersection of cubes $cube_a$ and $cube_b$ is not an empty set

then

{ if $cube_a$ absorbs $cube_b$

then { substitute $cube_b$ by the last cube of the array A ;

$cu := cu - 1$;

if $b = a$ then go to step 3

else go to step 2}

else { calculate the d solution cubes from the disjoint sharp

operation $cube_b \#_j cube_a$;

replace $cube_b$ by one solution cube and add

the other ones to the end of the array A ;

$cu := cu + d - 1$;

if $b \leq a$ then go to step 3

else go to step 2}}

else { $b := b - 1$;

if $b \leq a$ then go to step 3

else go to step 2}

step 3.

$b := cu$;

$a := a + 1$;

if $a = m$ then $so = b$; stop

else $m := cu$;

go to step 2.

Example V.1:

An example of an application of this algorithm is shown below. The array $\text{cube}[cu]$ describes the array A , the array $\text{sol}[so]$ is the result of one disjoint sharp operation. The order of the cubes is chosen in such a way that all different branches of the algorithm are passed through. Figure 16 shows the states of the algorithm when the content of the array of cubes has been changed. The figures in Figure 16 are referred to in the description below. The symbol "•" denotes the beginning of a new loop in the algorithm, the indentation shows the inner and outer loop.

step 1 : Initialization

number of cubes $cu = 5$;

Figure 16 a

step 2 : Loop

• *$cu = b = 5 ; a = 1 ;$*

intersection : $\text{cube}[1] \cap \text{cube}[4] = 1100;$

absorption : $\text{cube}[4] \neq 1100;$

disjoint sharp : $\text{cube}[4] \#_j \text{cube}[1] :$

$\text{sol}[1] = 01XX$

$\text{sol}[2] = 111X$

$\text{sol}[3] = 1101$

substitute $\text{cube}[4]$ with $\text{sol}[1]$;

place the rest of solution cubes at the end of the array

Figure 16 b

$$cu = cu + d - 1 = 7$$

- $cu = 7 ; a = 1 ; b = 3 ;$

$$\text{intersection : } \quad \text{cube}[1] \cap \text{cube}[3] = \emptyset$$

- $cu = 7 ; a = 1 ; b = 2 ;$

$$\text{intersection : } \text{cube}[1] \cap \text{cube}[2] = \emptyset$$

- $cu = 7 ; a = 2 ; b = 7 ;$

$$\text{intersection : } \quad \text{cube}[2] \cap \text{cube}[7] = \emptyset$$

- $cu = 7 ; a = 2 ; b = 6 ;$

$$\text{intersection : } \quad \text{cube}[2] \cap \text{cube}[6] = 111X;$$

$$\text{absorption : } \quad \text{cube}[6] = 111X;$$

substitute cube[6] with cube[n] and remove cube[n]

Figure 16 c

- $cu = 6 ; a = 2 ; b = 5 ;$

cube[5] is DC-cube

- $cu = 6 ; a = 2 ; b = 4 ;$

$$\text{intersection : } \text{cube}[2] \cap \text{cube}[4] = \emptyset$$

- $cu = 6 ; a = 2 ; b = 3 ;$

$$\text{intersection : } \text{cube}[2] \cap \text{cube}[3] = 1X11;$$

$$\text{absorption : } \text{cube}[3] \neq 1X11;$$

disjoint sharp: cube[3] #j cube[2] :

$$\text{sol}[1] = 0X11;$$

substitute cube[3] with sol[1]

Figure 16 d

- $cu = 6 ; a = 3 ; b = 6 ;$

intersection : cube[3] \cap cube[6] = \emptyset

- $cu = 6 ; a = 3 ; b = 5 ;$

cube[5] is DC-cube

- $cu = 6 ; a = 3 ; b = 4 ;$

intersection : cube[3] \cap cube[4] = 0111;

absorption : cube[4] \neq 0111

disjoint sharp : cube[4] #; cube[3] :

sol[1] = 010X

sol[2] = 0110;

substitute cube[4] with sol[1];

place sol[2] at the end of the array

Figure 16 d

- $cu = 7 ; a = 4 ; b = 7 ;$

intersection : cube[4] \cap cube[7] = \emptyset

- $cu = 7 ; a = 4 ; b = 6 ;$

intersection : cube[4] \cap cube[6] = \emptyset

- $cu = 7 ; a = 4 ; b = 5 ;$

cube[5] is DC-cube

- $cu = 7 ; a = 5 ; b = 7 ;$

cube[7] is DC-cube

- $cu = 7 ; a = 6 ; b = 7 ;$

intersection : cube[6] \cap cube[7] = \emptyset

solution array :

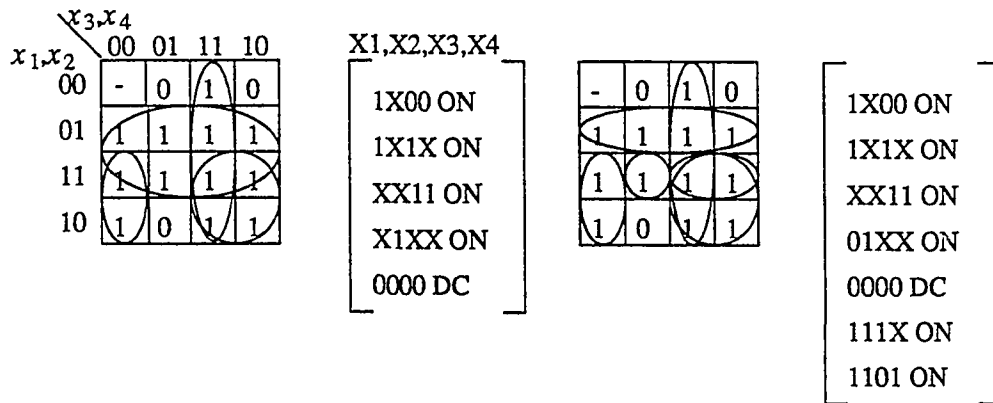
Figure 16 e

In order to generate disjoint DC-cubes the same algorithm has to be performed on the DC-cubes. In the above example, it is obvious, that the single DC-cube is a disjoint one and the algorithm does not have to be called.

The next two sections describe the properties used to develop the computer method to generate the Rademacher-Walsh spectra for completely and incompletely specified Boolean functions (Section V.3) and for systems of completely and incompletely specified Boolean functions (Section V.4).

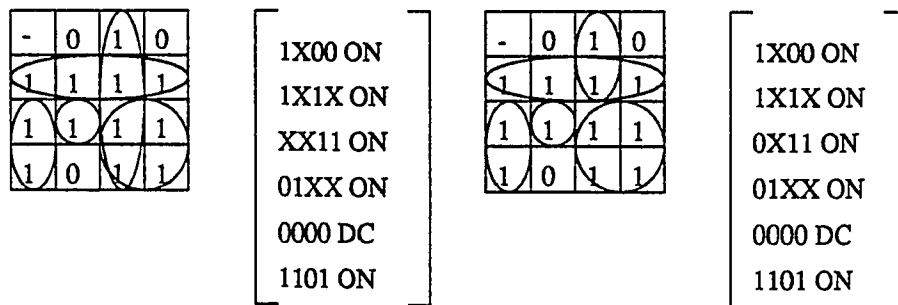
V.3 APPLICATION OF AN ARRAY METHOD FOR THE CALCULATION OF SPECTRAL COEFFICIENTS OF COMPLETELY AND INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS

An algorithm already exists for calculating spectral coefficients for completely specified Boolean functions directly from a sum-of-products Boolean expression [22] and [33]. In case the implicants are not mutually disjoint this algorithm requires additional correction to calculate the exact values of spectral coefficients for minterms of Boolean function F that are included more than once in some implicants. By using a representation of Boolean functions in the form of an array of disjoint cubes one can apply the existing algorithm without having to perform any additional correction operations, because, for an array of disjoint cubes as input data the exact values of spectral coefficients can be calculated immediately. Here the extension of the algorithm to incompletely specified Boolean functions is proposed.



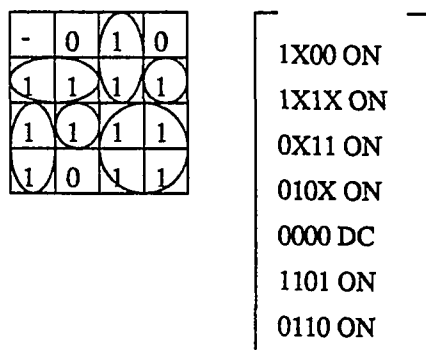
a. input array

b. cube[4] #; cube[1]



c. absorption of cube[6]

d. cube[3] #; cube[2]



e. cube[4] #; cube[3]

Figure 16. Algorithm generating disjoint cube representation in stages.

In what follows the properties of the existing algorithm are rewritten in notation corresponding to our representation of Boolean functions with n variables in the form of arrays of disjoint cubes. This is the first time all the properties describing incompletely specified Boolean functions have been presented.

Definition V.1: The *cube of degree m* is a cube that has m literals that can be either in affirmation or negation (i.e., m is equal to the sum of the number of zeros and ones in the description of a cube).

Let symbol p denote the number of X's in the cube. Then, $n = m + p$.

Example V.1:

Consider the cube $1X00$. It is a cube of degree 3 since three of the literals describing this cube are either in affirmation (x_1) or negation (x_3 and x_4). The cube does not depend on literal x_2 .

Definition V.2: The *partial spectral coefficient of an ON- or DC- cube with degree m of a Boolean function F* is equal to the value of the spectral coefficient that corresponds to the contribution of this cube to the full n -space spectrum of the Boolean function F .

The number of partial spectral coefficients $np\text{sc}$ describing the Boolean function F is equal to the number of ON- and DC- cubes describing the function. When the Boolean function is described by its truth table (a set of true, false, and don't care minterms) then the number of partial spectral coefficients of the Boolean function in this representation is equal to the sum of the number of ON- and DC- minterms. Let symbol fm denote the number of false minterms. Then, $np\text{sc} = 2^n - fm$.

Example V.2:

Consider Table I representing the array method of calculating spectral coefficients. Each row in this Table shows the partial spectral coefficients of either ON- or DC- cubes of a Boolean function. The function in the example has seven partial spectra. The number of partial spectra is equal to the number of disjoint ON- and DC- cubes describing the

function ($npsc = 7$).

Suppose arrays of disjoint ON- and DC- cubes that fully define Boolean function F are given. Then each cube of degree m can be treated as a minterm within its particular reduced m -space of function F . Let us recall that the spectrum of each true minterm is given by $s_0 = 2^n - 2$, and all remaining $2^n - 1$ coefficients are equal to ± 2 (Property III.15). Similarly the spectrum of each don't care minterm is given by $s_{DC\ 0} = 2^n - 1$, and all the remaining $2^n - 1 - 1$ coefficients are equal to ± 1 (Property III.17). The symbols $s_{DC\ I}$ denote spectral coefficients corresponding to DC- cubes (when $I = 0$, the symbol $s_{DC\ 0}$ denotes a direct current spectral coefficient corresponding to a DC- cube).

Cubes of degree m have the following properties.

V.1 The contribution of an ON- cube of degree m to the full n -space spectrum of function F (where n is the number of variables in the function F) is:

$$s_0 \text{ in full } n\text{-space} = 2^n - 2 \times 2^p \quad (\text{V.1})$$

and

$$s_I \text{ in full } n\text{-space} = s_I \text{ in } m\text{-space} \times 2^p \quad (\text{V.2})$$

where $I \neq 0$.

V.2 The contribution of a DC- cube of degree m to the full n -space spectrum of function F is:

$$s_{DC\ 0} \text{ in full } n\text{-space} = 2^n - 1 - 2^p \quad (\text{V.3})$$

and

$$s_{DC\ I} \text{ in full } n\text{-space} = s_{DC\ I} \text{ in } m\text{-space} \times 2^p \quad (\text{V.4})$$

where $I \neq 0$.

Notice that when the above formulae are applied to minterms (i.e., for $m = n$, and $p = 0$) they reduce to Properties III.15 and III.17. The contribution of a DC- cube of degree m is equal to one half of the contribution of an ON- cube that has the same degree m . More-

over, the contribution of ON- or DC- cubes of degree m to the full n -space spectrum of function F can be expressed for s_0 as the absolute value of the sum of all negative spectral coefficients corresponding to these cubes.

Equations (V.2) and (V.4) determine the absolute values of those partial spectral coefficient s_I that are not equal to zero for a given cube. Properties V.3 - V.5 determine the signs of the partial spectral coefficients, and if some of them are equal to zero.

Example V.3:

Consider again Table I. The value of partial spectral coefficient s_0 , corresponding to the ON- cube 1X00 ($n = 4, p = 1$), is equal to $2^4 - 2 \times 2^1 = 12$ according to (V.1). The absolute values of those partial spectral coefficients s_I that are not equal to zero are calculated according to (V.2) and are equal to $2 \times 2^1 = 4$.

The value of partial spectral coefficient s_0 corresponding to the DC- cube 0000 ($n = 4, p = 0$) is equal to $2^3 - 2^0 = 7$ according to Equation (V.3). The absolute values of those partial spectral coefficients s_I that are not equal to zero are calculated according to equation (V.4) and are equal to $1 \times 2^0 = 1$.

The following properties determine which partial spectral coefficients have values zero for an ON- or DC- cube of the degree m .

- V.3 If in a given cube the x_i variable of a Boolean function is denoted by the symbol "X", then all of the partial spectral coefficients s_I whose indexes I contain the subindex i are equal to 0.
- V.4 If in a given cube each of the variables of a Boolean function x_i, x_j, x_k , etc. from the complete set of all variables of the function is denoted by the symbol "X", then every partial spectral coefficient s_I whose index I contains the subindices i, j, k , etc. is equal to 0.
- V.5 For an ON- or DC- cube of degree m the number of nonzero partial spectral

coefficients is equal to 2^{n-p} .

Example V.4:

Consider again Table I. The variable x_2 is denoted by symbol X in the cube $1X00$. Then, by Property V.3 the values of all partial spectral coefficients with a subindex of 2 are equal to zero. Therefore, $s_2 = s_{12} = s_{23} = s_{24} = s_{123} = s_{124} = s_{234} = s_{1234} = 0$. For this cube, by Property V.5, the number of partial spectral coefficients different from zero is equal to $2^{4-1} = 8$.

The cube $1X1X$ has two variables denoted by the X symbols: x_2 , and x_4 . Therefore, by Property V.4, only the partial spectral coefficients s_0 , s_1 , s_3 , and s_{13} are different from zero (by Property V.5 the number of these coefficients is equal to $2^{4-2} = 4$).

The following properties describe the signs of each partial spectral coefficient s_I , where $I \neq 0$, and are valid for ON- and DC- cubes of any degree:

- V.6 If in a given cube the x_i variable of a Boolean function is in affirmation, then the sign of the corresponding first order coefficient is positive; otherwise for a variable that is in negation, the sign of the corresponding first order coefficient is negative. If in a given cube the x_i variable of a Boolean function is in affirmation, then the sign of the corresponding first order coefficient is positive; otherwise for a variable that is in negation, the sign of the corresponding first order coefficient is negative.
- V.7 The signs of all even order coefficients are given by multiplying the signs of the related first order coefficients by -1 .
- V.8 The signs of all odd order coefficients are given by multiplying the signs of the related first order coefficients.

Example V.5:

Consider again Table I. In the ON- cube 1X00 the variable x_1 is in affirmation, while the variables x_3 and x_4 are in negation. Therefore, by Property V.4 the sign of the partial spectral coefficient s_1 is positive and the signs of partial spectral coefficients s_3 and s_4 are negative.

The signs of second order coefficients are determined by Property V.7. The sign of the even order partial spectral coefficient s_{13} of cube 1X00 is positive, since this sign is determined by the product of the related first order coefficients, s_1 and s_3 , times -1 , i.e., $(-1) \times 1 \times (-1) = 1$.

The signs of the third order coefficients are determined by Property V.8. The signs of the partial spectral coefficient s_{134} of the same cube is positive since it is determined according to Property V.8 as the product of the related first order coefficients, s_1 , s_3 , and s_4 and the result is $1 \times (-1) \times (-1) = 1$.

The algorithm is as follows:

Algorithm V.1: Calculation of spectral coefficients for completely and incompletely specified Boolean functions.

1. For each ON- and DC- cube of degree m , calculate the value and the sign of the contribution of the cube to the full n -space spectrum according to the properties described previously.
2. The values of all of the spectral coefficients s_l , except s_0 are equal to the sum of all of the contributions to the spectral coefficients from all ON- and DC- disjoint cubes from an array of cubes.
3. For a completely specified Boolean function the value of the dc spectral coefficient s_0 is equal to the sum of all of the partial spectral coefficients corresponding to all of the disjoint ON- cubes describing the function, times the

correction factor $-(k - 1) \times 2^n$, where k is the number of disjoint cubes in the array of ON- cubes.

4. For an incompletely specified Boolean function the value of the dc spectral coefficient s_0 is equal to the sum of all of the partial spectral coefficients corresponding to all of the disjoint DC- cubes describing the function, times the correction factor $-(w - 1) \times 2^n$, where w is the number of disjoint cubes in the array of DC- cubes.
5. For an incompletely specified Boolean function the value of the dc spectral coefficient s_0 is equal to the sum of all of the partial spectral coefficients corresponding to all of the disjoint ON- and DC- disjoint cubes describing the function, multiplied by the correction factor $-(k - 1) \times 2^n - w \times 2^{n-1}$, where k is the number of disjoint ON- cubes, and w is the number of disjoint DC- cubes.

The correction factor $-(k - 1) \times 2^n$ is due to the fact that the cubes over the complete n -space have been added k times during the calculation of the k partial spectral coefficients. A similar explanation applies to DC- cubes as well.

Of course, the algorithm can calculate each coefficient separately or in parallel. If some of the 2^n spectral coefficients are not needed for a particular application, then a reduced number of operations can be performed.

Example V.6:

An example of the calculation of the S spectrum for a four variable incompletely specified Boolean function is shown in Table I. The function in this example is the same as the one used in Example III.2, III.4 and V.1. Figure 16 showed the stages of the execution of the algorithm generating the disjoint cube representation for the same function. The input data for the algorithm for this section was presented in Figure 16.e. The array of disjoint cubes representing the function is repeated from Figure 16.e as the first column in Table I. The values and signs of all of the partial spectral coefficients for this

function are determined by Properties V.1 - V.8. The results of the application of these properties are shown for two cubes.

The spectral coefficients of the first ON- cube in Table I (cube 1X00 of degree $m = 3$) are as follows:

1. within its own m -space, treated as a single minterm:

$$s_0 = 6, s_1 = 2, s_2 = 0, s_3 = -2, s_4 = -2, s_{12} = 0, s_{13} = 2, s_{14} = 2,$$

$$s_{23} = 0, s_{24} = 0, s_{34} = -2, s_{123} = 0, s_{124} = 0, s_{134} = 2, s_{234} = 0, s_{1234} = 0.$$

2. within the full n -space of Boolean function F (partial spectral coefficients):

$$s_0 = 12, s_1 = 4, s_3 = -4, s_4 = -4, s_{13} = 4, s_{14} = 4, s_{34} = -4, s_{134} = 4.$$

On the other hand, the spectral coefficients of the DC- cube in Table I (cube 0000 of degree $m = 4$ i.e., single minterm) are as follows:

1. within its own m -space, treated as a single minterm:

$$s_0 = 7, s_1 = -1, s_2 = -1, s_3 = -1, s_4 = -1, s_{12} = -1, s_{13} = -1,$$

$$s_{14} = -1,$$

$$s_{23} = -1, s_{24} = -1, s_{34} = -1, s_{123} = -1, s_{124} = -1, s_{134} = -1,$$

$$s_{234} = -1, s_{1234} = -1.$$

2. within the full n -space of Boolean function F (partial spectral coefficients) - the same as within its own m -space since it is a single minterm.

In order to obtain the values of all of the spectral coefficients of the whole function, except s_0 , the columns of partial spectral coefficients corresponding to all cubes describing the function are added (step 2 of the algorithm). The value of s_0 is obtained by the addition of all partial spectral coefficients with the correction factor (step 5 of the algorithm).

The resulting spectrum is shown at the bottom row of Table I, and, as it can be easily checked, is exactly the same as the one obtained by matrix multiplication in Example

III.2 and by using the standard trivial functions in Example III.4.

TABLE I
COMPLETE RADEMACHER-WALSH SPECTRUM S

$x_1 x_2 x_3 x_4$	s_0	s_1	s_2	s_3	s_4	s_{12}	s_{13}	s_{14}	s_{23}	s_{24}	s_{34}
1X00 ON	12	4	0	-4	-4	0	4	4	0	0	-4
1X1X ON	8	8	0	8	0	0	-8	0	0	0	0
0X11 ON	12	-4	0	4	4	0	4	4	0	0	-4
010X ON	12	-4	4	-4	0	4	-4	0	4	0	0
0000 DC	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1101 ON	14	2	2	-2	2	-2	2	-2	2	-2	2
0110 ON	14	-2	2	2	-2	2	2	-2	-2	2	2
	-9	3	7	3	-1	3	-1	3	3	-1	-5

$x_1 x_2 x_3 x_4$	s_{123}	s_{124}	s_{134}	s_{234}	s_{1234}
1X00 ON	0	0	4	0	0
1X1X ON	0	0	0	0	0
0X11 ON	-4	0	-4	0	0
010X ON	4	0	0	0	0
0000 DC	-1	-1	-1	-1	-1
1101 ON	-2	2	-2	-2	2
0110 ON	-2	2	2	-2	-2
	-1	3	-1	-5	-1

V.4 CALCULATION OF SPECTRAL COEFFICIENTS FOR SYSTEMS OF COMPLETELY AND INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS

The algorithm from the previous Section can be modified easily to calculate Walsh spectra of systems of Boolean functions. The calculation of a Walsh spectrum for a system of completely specified Boolean functions was presented in [15] for the R coding. There, the calculation of the R spectrum of a system of incompletely specified Boolean functions, with the following restriction is considered.

Restriction V.1: When a system of incompletely specified Boolean functions is considered then all of the functions of the system have the same don't care minterms (i.e., the same cells of Karnaugh maps are not specified in every function of the system).

The method presented in [15] has, however, all the drawbacks of the classical approach of spectral methods since it uses matrix calculation methods.

In this section the representation of systems of Boolean functions with the above restriction on a system of incompletely specified Boolean functions is presented for the first time for S coding. Moreover, the representation of systems of incompletely specified Boolean functions that can have any don't care minterms is introduced. When applied to a system of Boolean functions, the method still has all of the advantages described in the previous section.

Let us assume that the functions in the system are in the order: $F [1], F [2], F [3], \dots F [\mu]$, where μ is the number of the functions in the system and the function $F [\mu]$ is on the rightmost position in the system. Then, for the system of completely specified Boolean functions and for the system of incompletely specified Boolean functions, with Restriction V.1, the following properties hold:

- V.9 The contribution of the spectrum of the function $F [i], i = 1, 2, \dots, \mu$ to the total spectrum of a system of Boolean functions S_{TOT} is equal to the spectrum $S_{[i]}$ of the function $F [i]$ calculated by Algorithm V.1, which in turn has to be modified by Equation (V.5).
- V.10 The total spectrum of a system of Boolean functions S_{TOT} is equal to the sum of all the modified spectra of all the Boolean functions in the system.

The contribution of the spectrum of the "i-th" function $F [i]$ to the total spectrum of a system of μ Boolean functions is denoted in Equation (V.5) by $S_{[i]}^*$ and the spectrum of the "i-th" function calculated by Algorithm V.1 is denoted by $S_{I[i]}$.

$$S_{[i]}^* = 2^{\mu - i} \times S_{I[i]} \quad (V.5)$$

Recall that S denotes the spectrum and s a spectral coefficient.

When the more general case of a system of incompletely specified Boolean functions having arbitrary don't care minterms is considered, the system has to be represented

by two spectra - one corresponding to the don't care minterms of the system and the second corresponding to the true minterms. The requirement of having two separate spectra for a system of arbitrary incompletely specified Boolean functions is caused by the properties of the Rademacher-Walsh matrix T . If Properties V.9 and V.10 were applied to don't care and true minterms of an arbitrary system of incompletely specified Boolean functions, then the original system of functions would not be retrieved when the inverse transform is applied. For example, the contribution of the don't care minterm for the function $F [u - 1]$ to the total spectrum of the system of Boolean functions would be, in a case of using both Properties V.9 and V.10, the same as the contribution of the true minterm of the function $F [u]$. Therefore, Properties V.9 and V.10 can be applied to an arbitrary system of incompletely specified Boolean functions only after representing each of the functions in this system by two arrays of cubes: one of only don't cares minterms and the other of only true minterms. The total spectrum has to be calculated for each of these arrays separately. Then, the system of incompletely specified Boolean functions should be processed by the following algorithm.

Algorithm V.3: Spectral coefficients calculation for a system of arbitrary incompletely specified Boolean functions.

1. Represent each function in the system of Boolean functions by arrays of disjoint ON- and DC- cubes according to Algorithm V.1.
2. Calculate the spectrum of an array of ON- cubes for each separate function $F [i]$ according to Algorithm V.2.
3. Calculate the total spectrum $S_{TOT\ ON}$ of the system by using Properties V.9 and V.10.
4. Calculate the spectrum of an array of DC- cubes of each separate function $F [i]$ according to Algorithm V.2.

5. Calculate the total spectrum $S_{TOT DC}$ of the system by using Properties V.9 and V.10.

Example V.7:

An example of the calculation of spectra $S_{TOT ON}$ and $S_{TOT DC}$ of a system of two incompletely specified Boolean functions ($u = 2$), having four variables, is shown in Table II and Table III. The function F [2] in this example is the same as the one used in Example III.2, III.4, V.1, V.6. The function F [1] is taken from [12]. Both functions have no restriction in the choice of don't care minterms, therefore Algorithm V.3 has to be performed. The original functions are presented in Figure 17a - function F [1], and Figure 17d - function F [2]. The sets of ON- minterms that describe ON- cubes is presented in Figure 17b, and Figure 17e. The sets of DC- minterms are shown in Figure 17c and Figure 17f. The corresponding arrays of ON- and DC- cubes are generated by Algorithm V.1 (step 1).

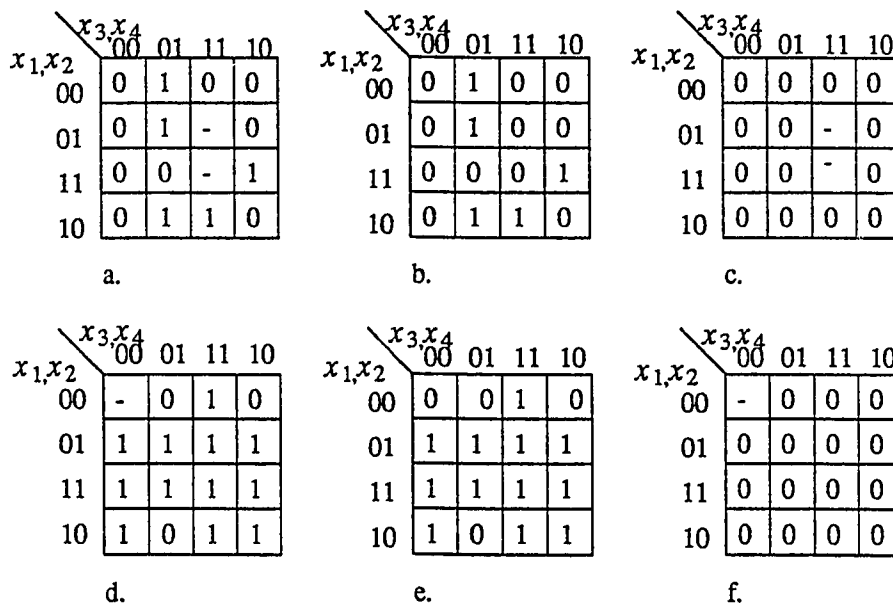


Figure 17. Incompletely specified Boolean functions F [1] and F [2].

The execution of the second step of Algorithm V.3 for ON- cubes is shown in the first two rows of Table II. The modified value of the spectrum of function F [1] is shown in the third row of Table II (step 3). Since for the function F [2] the modified value of the spectrum is equal to the original one this value is not repeated in the table. The total spectrum $S_{TOT ON}$ of this system of functions is the sum of rows one and three. The result of this addition is shown in the row 4 of Table II.

TABLE II
SPECTRUM S TOT-ON OF A SYSTEM OF BOOLEAN FUNCTIONS

	s_0	s_1	s_2	s_3	s_4	s_{12}	s_{13}	s_{14}	s_{23}	s_{24}	s_{34}
$S_{[2]}$	-10	2	6	2	-2	2	-2	2	2	-2	-6
$S_{[1]}$	6	2	-2	-2	2	2	-6	6	-2	2	10
$S_{[1]}^*$	12	4	-4	-4	4	4	-12	12	-4	4	20
$S_{TOT ON}$	2	6	2	-2	2	6	-14	14	-2	2	14

	s_{123}	s_{124}	s_{134}	s_{234}	s_{1234}
$S_{[2]}$	-2	2	-2	-6	-2
$S_{[1]}$	2	-2	-2	2	-2
$S_{[1]}^*$	4	-4	-4	4	-4
$S_{TOT ON}$	2	-2	-6	-2	-6

The execution of the fourth step of Algorithm V.3 is shown in the first two rows of Table III. The modified value of the spectrum of function F [1] is shown in the third row of Table III (step 5). Since for the function F [2] the modified value of the spectrum is equal to the original one, this value is not repeated in the table. The total spectrum $S_{TOT DC}$ of this system of functions is the sum of rows one and three and is shown in row 4.

The system of Boolean functions considered is represented by two spectra, $S_{TOT ON}$ and $S_{TOT DC}$, shown in Table II and Table III.

TABLE III
SPECTRUM S TOT-DC OF A SYSTEM OF BOOLEAN FUNCTIONS

	s_0	s_1	s_2	s_3	s_4	s_{12}	s_{13}	s_{14}	s_{23}	s_{24}	s_{34}
$S_{[2]}$	7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
$S_{[1]}$	6	0	2	2	2	0	0	0	-2	-2	-2
$S_{[1]}^*$	12	0	4	4	4	0	0	0	-4	-4	-4
$S_{TOT DC}$	19	-1	3	3	3	-1	-1	-1	-5	-5	-5

	s_{123}	s_{124}	s_{134}	s_{234}	s_{1234}
$S_{[2]}$	-1	-1	-1	-1	-1
$S_{[1]}$	0	0	0	2	0
$S_{[1]}^*$	0	0	0	4	0
$S_{TOT DC}$	-1	-1	-1	3	-1

A system of completely specified Boolean functions or incompletely specified Boolean functions with Restriction V.1 can be represented by one spectrum. For this Algorithm V.3 can be simplified to V.4.

Algorithm V.4: Spectral coefficients calculation for a system of an incompletely specified Boolean functions (with Restriction V.1) or a system of completely specified Boolean functions.

1. Represent each function in the system of Boolean functions by arrays of disjoint ON- and DC- cubes according to Algorithm V.1.
2. Calculate the spectrum of each separate function $F [i]$ according to Algorithm V.2.
3. Calculate the total spectrum S_{TOT} of the system by using Properties V.9 and V.10.

CHAPTER VI

A FAMILY OF ALL ESSENTIAL RADIX-2 ADDITION AND SUBTRACTION MULTI-POLARITY TRANSFORMS WITH ALGORITHMS AND INTERPRETATIONS IN THE BOOLEAN DOMAIN

VI.1 DESCRIPTION

Encouraged by a multiplicity of applications of Fourier, Walsh and Reed-Muller transforms the author is investigating new orthogonal transforms that can find applications in Boolean minimization, testing, image coding, cryptography and communication. With respect to the simplicity of the implementation the author assumes that the operations used in the transformations are ordinary addition and subtraction. One of these transforms is the well-known *Hadamard-Walsh transform* (Chapter II) that is applied here to binary and ternary vectors. One of the other transforms considered, when applied to binary vectors, is called the *Arithmetic transform* [96]. However, this transform has never been applied to ternary vectors. The third transform, which is *completely new*, is called the *Adding transform*, and is applied to binary and ternary vectors.

The transforms mentioned above are obtained by introducing some operations on matrices and considering a family of first order matrices. Two new operations on matrices: the row-wise and column-wise joins (concatenations) of two matrices are used to create transforms of radix-2. The elementary second order matrices are expanded by using the standard *tensor product of matrices* known also as the *direct* or *Kronecker product* [15], [22], [24], [56], [64], [68] and [82].

In this Chapter it will be shown that when elementary second order matrices are

composed of only 0, 1 and - 1 there are only four essential types of radix-2 transforms (one of them is the identity matrix). All other permutations of elements 0, 1, and - 1 create second order matrices that can be obtained from the essential types by multiplication with some permutation matrices. Since the identity matrix is a trivial case from the point of view of transformation there are only three essential matrices of second order considered. After expansion of the basic types by using the Kronecker product the derived higher radix transforms are used to create spectra of binary and ternary vectors.

For each of the three transforms, the interpretation of each particular spectral coefficient on a Karnaugh map is presented. All mathematical relationships between the number of true, false, and don't care minterms in the Karnaugh map regions which correspond to *standard trivial functions* (where a standard trivial function is the region of a Karnaugh map corresponding to the given spectral coefficient) are stated for both codings and all three types of transforms.

In this presentation only ordinary subtraction and addition operations are used. Since the generalized Reed-Muller transforms [19], [55], [93], [94] and [95] (with all possible 2^n fixed polarities for n variable Boolean functions) have been found useful in Boolean minimization, design for testability, and image processing, the author proposes here to apply the same idea of fixed polarities for all the three transforms. The detailed description of the calculation of the generalized Reed-Muller transform using a disjoint cube description of Boolean functions is presented in Chapter VII. The concept of different polarities of the new transforms is important from the point of view of the analysis and synthesis of digital networks. It is already well known, for example, that the fixed-polarity Reed-Muller form can be better implemented for many Boolean functions than the standard sum-of-products expression [94]. The same savings, from the point of view of the computer memory used for storing the spectra, are valid for the new transforms as well.

A very important property of the new transforms should also be noted. With Reed-Muller transforms there is more than one expression for an incompletely specified Boolean function [95]. In the case of the new transforms this property is no longer valid - on the contrary, each incompletely specified Boolean function has only one spectrum. Hence, there is an exact relationship between incompletely specified Boolean functions and their spectra. So, it is always possible to use the new transforms to calculate inverse transforms for incompletely specified Boolean functions. In the case of completely specified Boolean functions none of the new transforms, nor the Reed-Muller transform lose any information and it is always possible to calculate the inverse transforms.

VI.2 DEFINITIONS OF ESSENTIAL RADIX-2 MATRICES

Some families of matrices will be defined. The building blocks for the definitions are three fundamental elements (matrices of orders 1×1): 0, -1 , and $+1$. The following operations on these matrices are introduced here.

Definition VI.1: A row-wise join or concatenation of a matrix A of order $n \times m$ and a matrix B of order $n \times m$ is the partitioned matrix C of order $n \times 2m$ such that its first m rows are exactly the same as the rows of matrix A and the rows from $m + 1$ to $2m$ are exactly the same as the rows of matrix B . This operator is denoted by the symbol "RWJ".

Definition VI.2: A column-wise join or concatenation of a matrix A of order $n \times m$ and a matrix B of order $n \times m$ is the partitioned matrix C of order $2n \times m$ such that its first n columns are exactly the same as the columns of matrix A and the columns from $n + 1$ to $2n$ are exactly the same as the columns of matrix B . This operator is denoted by the symbol "CWJ".

Let us apply the *CWJ* operator to three elementary matrices, of orders 1×1 , for all possible combinations of these matrices. The nine different matrices of order 2×1 that result from the application of *CWJ* to all these permutations are shown in Figure 18.

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Figure 18. Nine different matrices of order 2×1 .

Let us now apply the *RWJ* operator to all possible combinations of the nine matrices shown in Figure 18. The result will be 81 different matrices of order 2×2 . Some of them are not-orthogonal and are of no interest to us. All orthogonal matrices can be classified into four basic types. The first 45 matrices of the 81 matrices are shown in Figure 19, with the 4 basic types marked. The symbol "NO" on this Figure denotes a non orthogonal matrix. The way that the remaining 36 matrices are generated can be deduced from this figure. In each row of matrices shown in Figure 19, a different one of the nine matrices from Figure 18 is *joined* with all nine of the 2×1 matrices, one at a time, using *RJW* operation to form all nine of the 2×2 matrices in the row. The same 81 matrices could be generated by applying the *RWJ* operator to the three basic elements to obtain 9 matrices of order 1×2 , then joining them by using the *CWJ* operator to the elementary row matrices obtained in the previous step.

All basic types have been found by observing the following properties of these matrices: any matrix (of order 2×2) from a basic type can be obtained from another matrix of the same type by applying some of the following operations on matrices: transposition of rows, transposition of columns, change of all signs in a whole row or change of all signs in a whole column. So there are only four elementary types of matrices of order 2×2 composed from the elements 0, + 1, and - 1. One of these types, type I, is the identity matrix, and therefore is not interesting from the point of view of the transformations. There are three types of orthogonal, radix-2 matrices remaining and their application to the transformation of binary and ternary vectors is presented later. From each of

the three types one particular representative has to be chosen. In our case, in order to get some transforms which are already known, the matrices denoted by * in Figure 19 have been chosen. These three elementary matrices of 2×2 order will be denoted by the symbols H_2 (the *Hadamard transform* [24], [56], [64], [96] and [82], Chapter II), AR_2 (the *Arithmetic transform* [96]), and AD_2 (the *Adding transform*).

$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$
NO	IV*	IV	NO	II*	II	III	III	NO
$\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}$
IV	NO	NO	IV	III*	II	II	III	NO
$\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 \\ 1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}$
IV	NO	NO	IV	II	III	III	II	NO
$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 \\ -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ -1 & 0 \end{bmatrix}$
NO	IV	IV	NO	III	III	II	II	NO
$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$
II	II	III	III	I*	NO	I	NO	NO

Figure 19. First 45 possible matrices with the four basic types marked.

The Walsh functions in Hadamard order are generated when the standard

Kronecker product of the elementary Hadamard matrix H_2 is performed with itself. Similarly, the Arithmetic transform of higher orders is obtained by successive application of the Kronecker product to the core matrix AR_2 . The same is true for the Adding transform as well where the core matrix is AD_2 . When all these three elementary matrices are denoted by the same symbol TR_2 , then

$$TR_N = (TR_2)^{[n]}, \quad (\text{VI.1})$$

where $[n]$ in the exponent means the application of the Kronecker product n times, N is the order of the transform matrix, and $n = \log_2 N$.

Later it will be shown, how the derived transforms are used to create the spectra of ternary and binary vectors. Since the detailed description of the properties of the Hadamard-Walsh spectrum of Boolean functions has been presented in Chapter III, only the application and properties of Arithmetic and Adding transforms will be considered here.

VI.3 GENERALIZED ARITHMETIC AND ADDING TRANSFORMS

The Arithmetic transform AR_N has been used for the generation of a *canonic arithmetic expansion* of Boolean functions [55], [96] and [98]. In the literature, this expansion has only been used for completely specified Boolean functions. The author proposes *three extensions* of the currently used Arithmetic transform. First, it is proposed to use this transform not only for completely specified Boolean functions but for incompletely specified ones as well. Hence, the *Arithmetic transform can be applied not only to binary but also to ternary vectors*. Secondly, *two types of codings of Boolean functions are used*. In the first type, in the case of the completely specified Boolean function, the true minterms of the function are represented by 1 and false minterms by 0. When the second coding is used, the true minterms are represented by -1 and the false minterms by 1. In the case of the incompletely specified Boolean functions, in the first coding

scheme the don't care minterms are represented by 0.5, and in the second coding scheme by 0. The coding of the true and false minterms for the functions with don't cares is the same as the one for the completely specified Boolean functions. The same types of coding schemes have been used for Hadamard-Walsh spectrum of Boolean functions (Chapter III) and the corresponding Walsh spectra are known as the R spectrum (for the first type of coding later called the R coding) and the S spectrum (for the second type of coding later called the S coding). Thirdly, the *notion of the polarity of the Arithmetic transform is introduced*. Since for the Boolean function having n variables there exist 2^n possible substitutions of a given i -th variable by its complement it is possible to have an equal number (2^n) of possible expansions in which each variable is in either complemented or not-complemented form. All of these possible expansions are called the *generalized Arithmetic transforms* and are classified by their *polarities*. The latter notion is similar to the one used for Reed-Muller transforms [19], [95] and will be rewritten for our needs.

Definition VI.3: A *polarity* number is calculated by taking the decimal equivalent of the n -bit straight binary code formed by writing a 0 or a 1 for each variable dependently whether this variable is in a positive or complemented form respectively.

Let us illustrate the notions introduced by the following examples.

Example VI.1:

An example of the calculation of the Arithmetic transform of a fourth-order completely specified Boolean function in R coding is shown in Figure 20. The transform is in zero polarity, and all the variables describing the coefficients of the arithmetic canonical expansion are positive.

AR	X_R	C			
1	0	0	=	0	c_0
-1	1	0		0	c_1
-1	0	1		0	c_2
1	-1	-1		0	c_{12}
-1	0	0		0	c_3
1	-1	0		1	c_{13}
1	0	-1		0	c_{23}
-1	1	1		1	c_{123}
-1	0	0		1	c_4
1	-1	0		0	c_{14}
1	0	-1		1	c_{24}
-1	1	1		0	c_{124}
1	0	0		0	c_{34}
-1	1	0		0	c_{134}
-1	0	1		1	c_{234}
1	-1	-1		1	c_{1234}

Figure 20. Calculation of Arithmetic transform for a completely specified function.

The canonical arithmetic expansion for this function is as follows:

$$f(X) = x_3 x_1 + x_4 - x_4 x_2 x_1 - x_4 x_3 - x_4 x_3 x_1 + x_4 x_3 x_2 + x_4 x_3 x_2 x_1 \quad (\text{VI.2})$$

The addition symbol in the canonic arithmetic expansion "+" is an arithmetic addition and not Boolean "or". The value of a given minterm can be obtained from the arithmetic expansion of any polarity. When the binary equivalent of a minterm is substituted into the expansion, the value of each term in the expansion is calculated logically and the ones that correspond to the terms that are true after the first substitution are arithmetically added or subtracted. This rule is valid for both codings of completely and incompletely specified Boolean functions.

As can be easily checked, the values of all the minterms of this function can be generated from its canonical arithmetic expansion by replacing the literals $x_4, x_3, x_2,$ and x_1 with the binary code of a given minterm. For instance, the minterm 0000 has the value 0, and the minterm 1111 has the value $1 + 1 - 1 - 1 - 1 + 1 + 1 = 1$.

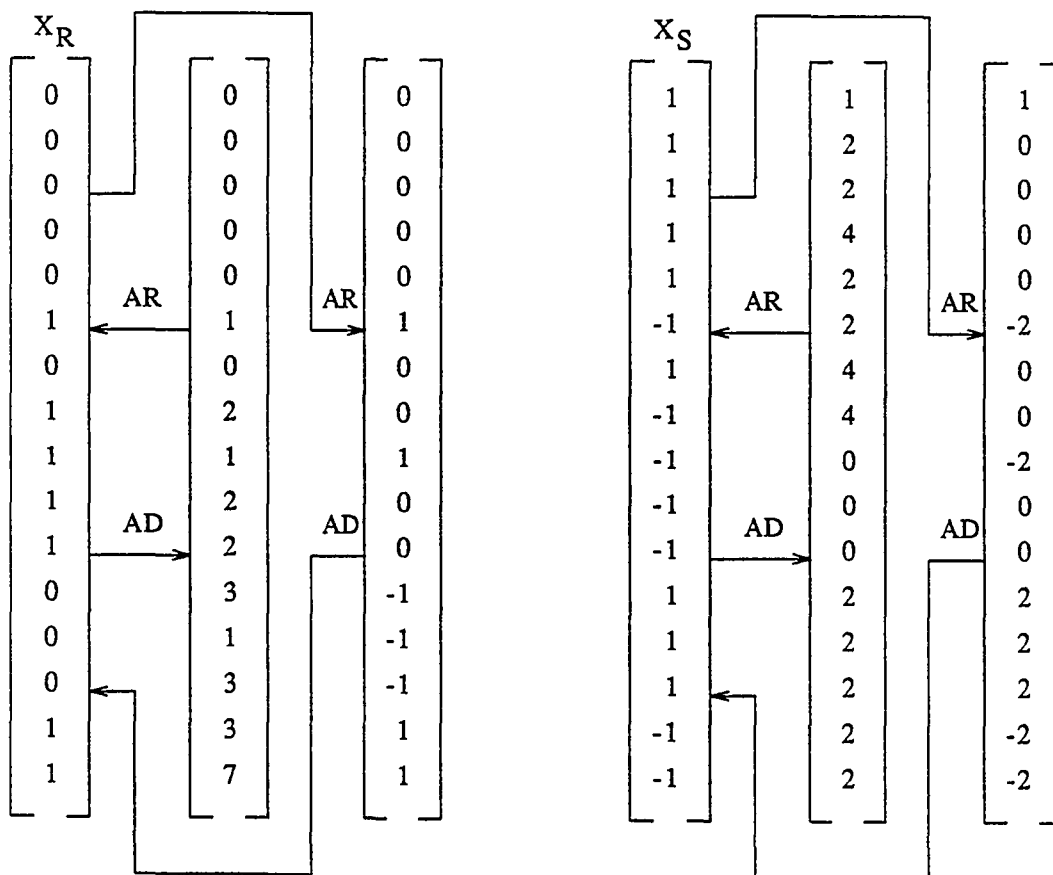


Figure 21. Calculation of Arithmetic and Adding transforms.

The other arithmetic canonical expansion can be obtained for this function from the second S coding. The coefficients for the second expansion are shown in Figure 21 (the vector on the right side of this figure with the arrow AR pointing to it). Since the polarity number is zero again the variables of the Boolean function occurring in the terms of this arithmetic canonical expansion are exactly the same as in the previous case. And again, it

can be easily checked that the values of the minterms of the function in the S coding can be obtained from the second canonical form by substituting the literal $x_4, x_3, x_2,$ and x_1 with the binary code of a given minterm.

The Arithmetic transform can be applied to both completely and incompletely specified Boolean functions in both S and R codings. Similarly to the Arithmetic transform, the Adding transform can be applied to both completely and incompletely specified Boolean functions. Two types of codings can be used for this transform as well. Moreover, the Adding transform can have the same polarities as the Arithmetic transform. Before showing the examples of other polarities, and applications of the transforms to incompletely specified Boolean functions let us state the fundamental relationship between these two transforms. *For the zero polarity, both matrices AR_N and AD_N are inverses of each other, i.e.,*

$$(AR_N)^{-1} = AD_n \quad (VI.3)$$

and

$$(AD_N)^{-1} = AR_n \quad (VI.4)$$

The transform matrix for the Adding transform looks similar to the transform matrix for the Arithmetic transform shown in Figure 20 - the only difference being the fact that all the entries in the matrix are + 1 i.e., all - 1 in the matrix for the Arithmetic transform should be replaced by + 1 for the Adding transform and all + 1 remain not changed. The next two examples are only for zero polarity only.

Example VI.2:

An example of both Arithmetic and Adding spectra for the same fourth order completely specified Boolean function X_R (in R coding) and X_S (in S coding) is shown in Figure 21. This is the same function that was used in Example VI.1. The arrows in this figure show the applications of the Arithmetic AR and Adding AD transforms. It is also shown that the transforms' matrices are inverses of each other.

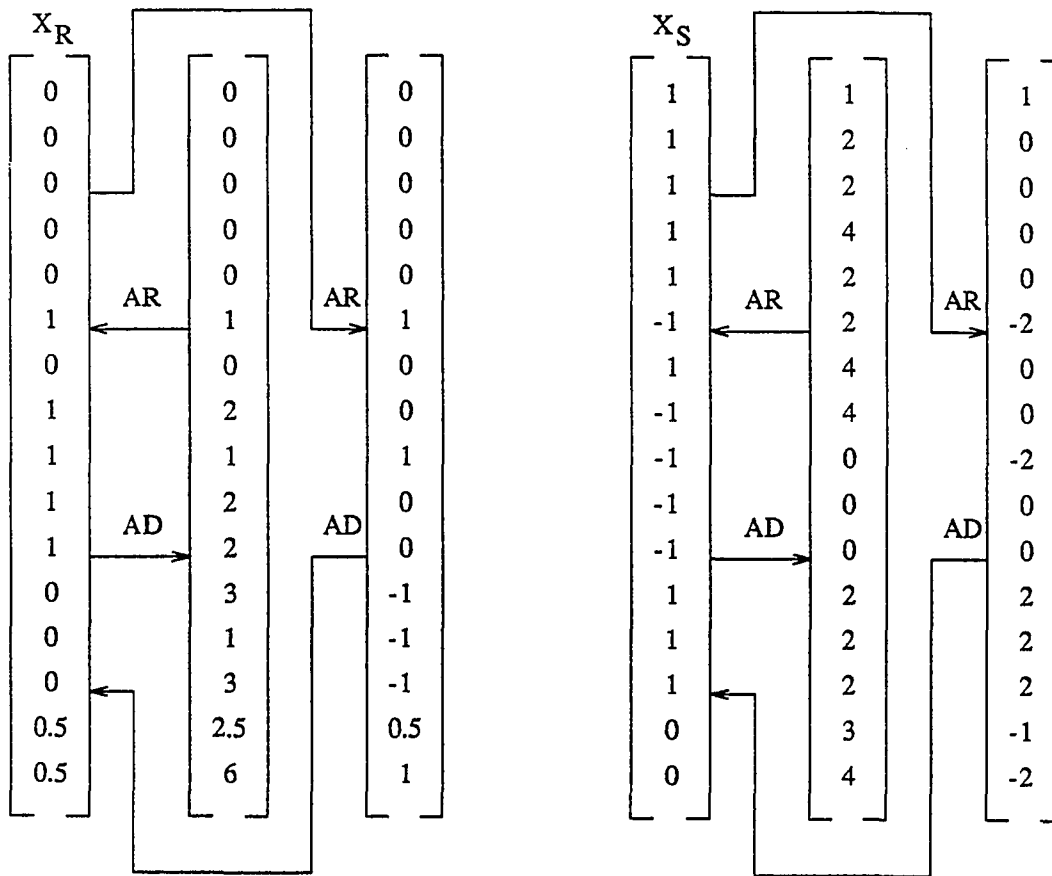


Figure 22. Calculation of Arithmetic and Adding transforms for an incompletely specified function.

Example VI.3:

Transformations of the same fourth order incompletely specified Boolean functions in two codings by means of the Arithmetic and Adding transforms are shown in Figure 22. Also for incompletely specified Boolean functions both transforms' matrices are inverses of each other.

It is very important to notice that both Arithmetic and Adding spectra are *canonical representations* of completely and incompletely specified Boolean functions for any polarity. This property of both transforms makes them especially distinct from other related transforms. For example, the Reed-Muller transform that has the same

transformation matrix as the Adding transform (for any given polarity this correspondence exists) and only the operations of addition are executed "modulo-2" instead of normal arithmetic addition as in the case of the Adding transform, does not have a canonical form for the transformation of incompletely specified Boolean function [95].

An important relationship exists between the Arithmetic spectral coefficients calculated according to R and S codings for both completely and incompletely specified Boolean functions and for all polarities. When arr_I (where the I are different natural numbers) denotes the coefficients calculated for R coding, and ars_I denotes the coefficients calculated for S coding then for all ar_I except ar_0 the following formula holds:

$$arr_I = -\frac{1}{2} ars_I \quad (\text{VI.5})$$

For arr_0 and ars_0 equation (VI.5) is not valid. Instead, the following formula holds:

$$arr_0 = \frac{1}{2} (1 - ars_0). \quad (\text{VI.6})$$

Let us note, that the same relationship as equation (VI.5) is valid for Hadamard-Walsh spectral coefficients (equation III.5). However, equation (VI.5) does not hold for the coefficients from the Adding spectrum as can be easily checked in Figure 21 and Figure 22.

Let us now show examples of the generalized Arithmetic transform for the same completely specified Boolean function. Only one example of the generalized Arithmetic transform for the polarity 0011 is shown and only for the completely specified Boolean function. The examples for an incompletely specified Boolean function and for other coding can be derived by an interested reader in a manner similar to the one presented. The Adding transform can be calculated for this polarity by replacing all -1 by $+1$, and rewriting all $+1$ from the matrix describing the Arithmetic transform. Only one R coding is shown. The methods used to calculate the generalized Arithmetic and Adding

transforms for any coding and any Boolean function should be obvious from the previous examples.

Example VI.4:

The calculation of the Arithmetic transform with 0011 polarity for a fourth-order completely specified Boolean function is shown in Figure 23.

The standard trivial functions corresponding to the coefficients of the canonical arithmetic expansion for this polarity have positive and complemented forms. The symbols c_l have exactly the same meaning here as s_l in the case of Walsh transforms. See Property III.6 for more details.

$$\begin{array}{c}
 \text{AR} \\
 \left[\begin{array}{cccccccccccccccc}
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\
 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\
 -1 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\
 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & 1 & -1 \\
 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & -1 \\
 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & 1 & 1
 \end{array} \right]
 \begin{array}{c}
 \text{X}_R \\
 \left[\begin{array}{c}
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 1 \\
 0 \\
 1 \\
 1 \\
 1 \\
 0 \\
 1 \\
 0 \\
 0 \\
 1 \\
 1
 \end{array} \right]
 \end{array}
 =
 \begin{array}{c}
 \text{C} \\
 \left[\begin{array}{c}
 0 \\
 0 \\
 0 \\
 0 \\
 1 \\
 -1 \\
 0 \\
 0 \\
 1 \\
 1 \\
 -1 \\
 0 \\
 0 \\
 1 \\
 -2 \\
 1
 \end{array} \right]
 \begin{array}{c}
 c_0 \\
 c_1 \\
 c_2 \\
 c_{12} \\
 c_3 \\
 c_{13} \\
 c_{23} \\
 c_{123} \\
 c_4 \\
 c_{14} \\
 c_{24} \\
 c_{124} \\
 c_{34} \\
 c_{134} \\
 c_{234} \\
 c_{1234}
 \end{array}
 \end{array}$$

Figure 23. Calculation of Arithmetic transform for a function for polarity 0011.

Example VI.5:

The calculation of the inverse Arithmetic transform with 0011 polarity for the function from the previous example is shown in Figure 24.

$$\begin{array}{c}
 \text{AR}^{-1} \\
 \left[\begin{array}{cccccccccccccccc}
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0
 \end{array} \right]
 \begin{array}{c}
 \text{C} \\
 \left[\begin{array}{c}
 0 \\
 0 \\
 0 \\
 0 \\
 1 \\
 -1 \\
 0 \\
 0 \\
 0 \\
 1 \\
 1 \\
 -1 \\
 0 \\
 0 \\
 -2 \\
 1
 \end{array} \right]
 =
 \begin{array}{c}
 \text{X}_R \\
 \left[\begin{array}{c}
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 1 \\
 0 \\
 1 \\
 1 \\
 1 \\
 1 \\
 0 \\
 0 \\
 0 \\
 1 \\
 1
 \end{array} \right]
 \end{array}
 \end{array}$$

Figure 24. Calculation of an inverse Arithmetic transform for a function for polarity 0011.

VI.4 LINKS OF ARITHMETIC AND ADDING TRANSFORMS WITH CLASSICAL LOGIC DESIGN

Let us show the real meaning of the Arithmetic and Adding spectral coefficients in classical logic terms. Let the symbol a_i denote the spectral coefficient from either an Arithmetic or an Adding transform in any coding. The definition of *standard trivial functions* and their relationships to the spectral coefficients (from both Arithmetic and

Adding spectra) follows.

Definition VI.4: Each spectral coefficient a_I gives a *correlation value* between the Boolean function F and a *standard trivial function* u_I corresponding to the coefficient. The standard trivial functions for the spectral coefficients are, respectively, for the coefficients a_I (where $I = 0$) - the minterm of the Boolean function corresponding to a given polarity denoted by u_0 , for the first order coefficients a_I (where $I = i, i \neq 0$) - the minterm of the Boolean function u_0 and one of its neighbors, in turn, denoted by u_i , for the second order coefficients a_I (where $I = ij, i \neq 0, j \neq 0$) - the minterm of the Boolean function u_0 and three of its neighbors, in turn, denoted by u_{ij} , for the third order coefficients a_I and ($I = ijk, i \neq 0, j \neq 0, k \neq 0$) - the minterm of the Boolean function u_0 and seven of its neighbors, in turn, denoted by u_{ijk} , etc.

Since the formulas for the calculation of spectral coefficients are derived for both spectra the necessary symbols are introduced together. Moreover, let us expand our considerations for incompletely specified Boolean functions as well. The following symbols will be used. Let a_I be the number of true minterms of the Boolean function F , where both the function F and the standard trivial function u_I have the logic values of 1; let b_I be the number of true minterms of Boolean function F , where the function F has the logic value 1 and the standard trivial function u_I has the logic value 0; let c_I be the number of don't care minterms of the Boolean function F , where the standard trivial function u_I has the logic value 1; let d_I be the number of don't care minterms of Boolean function F , where the standard trivial function u_I has the logic value 0.

The arr_I Arithmetic spectral coefficients for completely specified Boolean functions with R coding having n variables, can be defined in the following way :

$$arr_0 = a_0, \quad (VI.7)$$

and

$$arr_I = a_I - b_I, \quad (VI.8)$$

when $I \neq 0$.

The arr_I Arithmetic spectral coefficients for incompletely specified Boolean functions with R coding having n variables, can be defined in the following way :

$$arr_0 = a_0 + \frac{1}{2} c_0, \quad (VI.9)$$

and

$$arr_I = (a_I - b_I) + \frac{1}{2} (c_I - d_I), \quad (VI.10)$$

when $I \neq 0$.

The formulas for the calculation of the Arithmetic spectral coefficients with S coding can be found from equations (VI.4, VI.5, VI.7-VI.9).

When the Adding spectrum with R coding is to be calculated the formulas for its coefficients are the same as equations (VI.7-VI.9) - the only difference being the replacement in all these formulas of the sign $-$ with $+$.

Example VI.6:

The standard trivial functions for the completely specified Boolean function of Example VI.1 for the 0000 polarity are shown in Figure 25. The circles denote the areas where the standard trivial functions have the logic values 1 while triangles denote the areas where the standard trivial functions have the logic values 0.

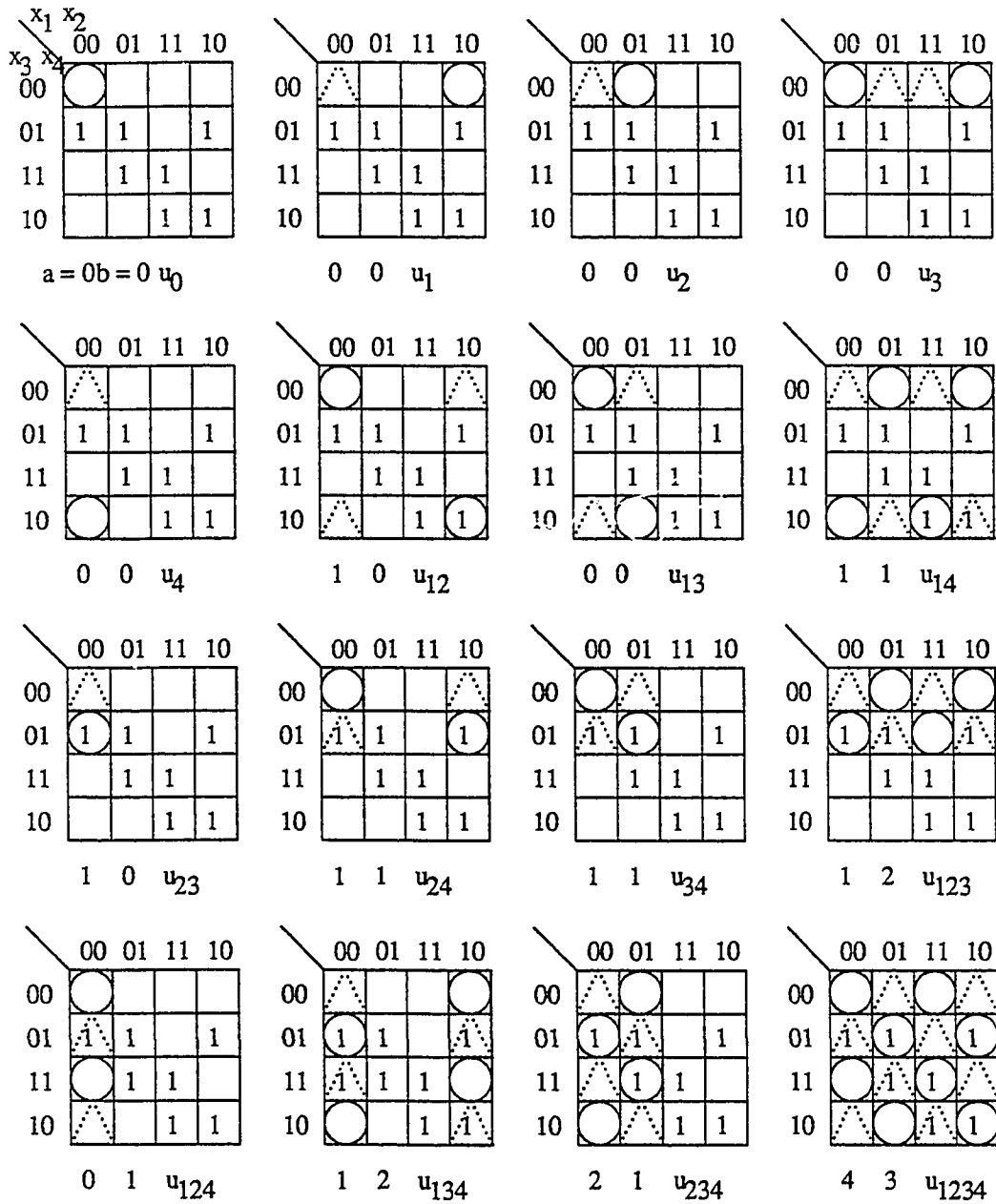


Figure 25. Standard trivial functions for polarity 0000.

Then, according to equations (VI.7) and (VI.8) the arithmetic spectral coefficients for this function are as follows:

$$\begin{aligned}
c_0 &= 0, c_1 = 0 - 0 = 0, \\
c_2 &= 0 - 0 = 0, c_3 = 0 - 0 = 0, \\
c_4 &= 1 - 0 = 1, c_{12} = 0 - 0 = 0, \\
c_{13} &= 1 - 0 = 1, c_{14} = 1 - 1 = 0, \\
c_{23} &= 0 - 0 = 0, c_{24} = 1 - 1 = 0, \\
c_{34} &= 0 - 1 = -1, c_{123} = 1 - 1 = 0, \\
c_{124} &= 1 - 2 = -1, c_{134} = 1 - 2 = -1, \\
c_{234} &= 2 - 1 = 1, c_{1234} = 4 - 3 = 1.
\end{aligned}$$

One can easily check that by calculating spectral coefficients according to the above formulas one obtains exactly the same results as previously when the classical matrix multiplication method was used (Figure 20).

Example VI.7:

The standard trivial functions for the function from the previous example for 0011 polarity are shown in Figure 26.

Then, according to equations (VI.7) and (VI.8) the arithmetic spectral coefficients for this function are as follows:

$$\begin{aligned}
c_0 &= 0, c_1 = 0 - 0 = 0, \\
c_2 &= 0 - 0 = 0, c_3 = 1 - 0 = 1, \\
c_4 &= 0 - 0 = 0, c_{12} = 0 - 0 = 0, \\
c_{13} &= 0 - 1 = -1, c_{14} = 1 - 0 = 1, \\
c_{23} &= 1 - 1 = 0, c_{24} = 1 - 0 = 1, \\
c_{34} &= 1 - 1 = 0, c_{123} = 1 - 1 = 0, \\
c_{124} &= 1 - 2 = -1, c_{134} = 2 - 2 = 0, \\
c_{234} &= 1 - 3 = -2, c_{1234} = 4 - 3 = 1.
\end{aligned}$$

One can easily check that by calculating spectral coefficients according to the above formulas one obtains exactly the same results as before when the classical matrix

multiplication method was used (Figure 23).

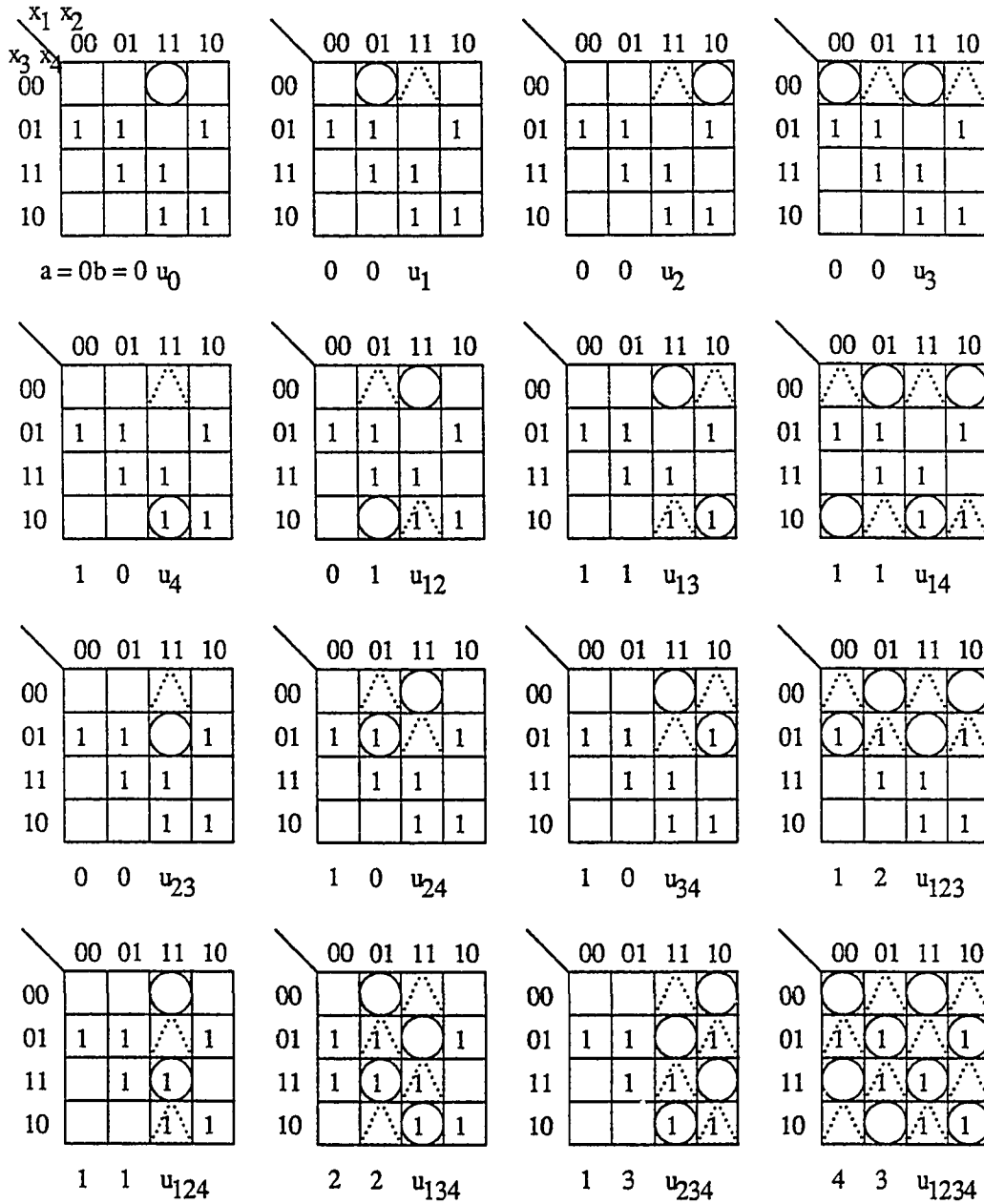


Figure 26. Standard trivial functions for polarity 0011.

CHAPTER VII

THE CALCULATION OF GENERALIZED REED-MULLER CANONICAL EXPANSIONS FROM DISJOINT CUBE REPRESENTATIONS OF BOOLEAN FUNCTIONS

VII.1 DESCRIPTION

The classical approach to analysis, synthesis and testing of digital circuits is based on the description by the operators of Boolean algebra. However, for many years, an alternate description based on the operations of modulo-2 arithmetic has been developed [19], [55], [99] and [100]. The algebra corresponding to this second approach, is an example of a finite field and supports such familiar digital signal processing operations as matrices, and fast transforms. Modulo-2 algebra is the simplest case of an algebra known in references [19] and [55] as a finite field or Galois field algebra and it is why it is frequently denoted by the symbol $GF(2)$.

Any Boolean function can be represented in modulo-2 algebra. The modulo-2 sum-of-products expression is known in the literature [19], [55] and [100] as the *complement-free ring-sum* or *Reed-Muller* expansion. In such an expression there are 2^n possible product terms selected from the n variables of a Boolean function. Each term is made up of un-complemented variables of a Boolean function only. By allowing the complementation of the input variables one can derive the *Generalized Reed-Muller* expansions (GRMEs) where each input variable x_i can appear either true throughout the expansion or complemented throughout it. It is apparent that there are 2^n GRMEs for a Boolean function with n variables, each with a different polarity, including the positive polarity form (i.e., Reed-Muller expansion). Since the input polarity of a GRME is con-

stant in these expansions they are termed fixed-polarity forms. It should be noted that for a given Boolean function each GRME is unique and is a *canonical form*.

The application of exclusive OR gates has some advantages over the gates commonly used in logic design. One of the reasons is that many useful functions have a high content of the so called linear part (the EXOR part) of the function. Some examples of such functions are: adders and parity checkers. What is more, circuits built with EXOR gates are easily testable. Reddy showed that when a circuit is designed using a Reed-Muller expansion and if only permanent stuck-at faults occur in either a single AND or only a single EXOR gate is faulty, then only $(n + 4)$ tests are required for fault-free primary inputs for an arbitrary n input Boolean function [41].

VII.2 BASIC DEFINITIONS AND PROPERTIES

The properties of the disjoint cube representation of a Boolean function, used in the description of the algorithm for calculating the GRME will be stated later. An algorithm that generates such a representation has been shown in Chapter V and its implementation has been described in [114]. In what follows, the definitions of the RME [19] and GRME are also presented.

Definition VII.1: The n -variable Reed-Muller expression takes the form:

$$F(x_n, x_{n-1}, \dots, x_1) = \sum_{i=0}^{2^n-1} c_i x_n^{e_{i,n}} x_{n-1}^{e_{i,n-1}} \dots x_1^{e_{i,1}} \quad (\text{VII.1})$$

In the above definition \sum means summation over $GF(2)$, and the $e_{i,j}$ are either 0 or 1 so that literal $x_k^0 = 1$ and $x_k^1 = x_k$.

Property VII.1: The n -variable Reed-Muller expression has 2^n possible product terms (piterms represented by symbol π) selected from the n variables.

Property VII.2: The subscript i in the piterm π_i represents the decimal equivalent of the straight binary code (SBC) formed from the join of symbols $e_{i,n}, e_{i,n-1}, \dots, e_{i,1}$,

where the most significant bit of the SBC is $e_{i,n}$ and the least significant bit of the SBC is $e_{i,1}$.

Definition VII.2: The n -variable Reed-Muller expression can be represented in short by:

$$F(x_n, x_{n-1}, \dots, x_1) = \sum_{i=1}^{2^n-1} c_i \pi_i, \quad (\text{VII.2})$$

where the meaning of the summation is exactly the same as in Definition VII.1.

The next property is the consequence of Definition VII.1 and Definition VII.2.

Property VII.3: The n -variable Reed-Muller expression is fully described by the set of all c_i Reed-Muller spectral coefficients.

Definition VII.3: The n -variable Generalized Reed-Muller expression takes the form:

$$F(x_n, x_{n-1}, \dots, x_1) = \sum_{i=1}^{2^n-1} c_i x_n^{e_i^\omega} x_{n-1}^{e_i^{\omega-1}}, \dots, x_1^{e_i^{\omega_1}} \quad (\text{VII.3})$$

In the above definition \sum means summation over $GF(2)$, and the e_i^ω are either 0, 1, or -1 so that literal $x_k^0 = 1$, $x_k^1 = x_k$, and $x_k^{-1} = \overline{x_k}$. The symbol ω denotes the polarity of the GRME.

Property VII.4: The n -variable Generalized Reed-Muller expression has 2^n possible product terms (generalized piterms represented by symbol π^ω) selected from the n variables.

Definition VII.4: The polarity number of the GRME expression denoted by ω is a binary string computed by taking the n bit straight binary code (SBC) formed by writing a 0 or a 1 for each variable according to whether it is used in affirmative or negative form respectively.

Property VII.5: For a given polarity ω each variable in all generalized piterms can be either in affirmative or negative form, or be absent from the piterm.

Definition VII.5: The n -variable Generalized Reed-Muller expression can be represented in short by:

$$F(x_n, x_{n-1}, \dots, x_1) = \sum_{i=1}^{2^n-1} c_i \pi_i^\omega \quad (\text{VII.4})$$

where the meaning of the summation is exactly the same as in Definition VII.3.

The next property is the consequence of Definition VII.3 and Definition VII.5.

Property VII.6: The n -variable Generalized Reed-Muller expression is fully described by the set of all c_i Reed-Muller spectral coefficients.

Definition VII.6: The *cube of degree m* is a cube that has m defined literals that can be either affirmative or negative (i.e., m is equal to the sum of the number of zeros and ones in the description of the cube).

Let symbol p denote the number of X 's in the cube (for explanation of the symbol X see Definition VII.9) and n denote the number of variables of a given Boolean function F . Then, $n = m + p$.

Definition VII.7: The cube and the piterm are of the *same degree (order)* when the number of defined literals is the same for both of them.

Property VII.7: The number of piterms of z -th order is equal to $C_n^z = \binom{n}{z}$, where n is the number of variables of the Boolean function.

Definition VII.8: The *partial Reed-Muller spectral coefficients* of an ON- cube cu_i of degree m from the disjoint cube representation of a Boolean function F are those parts of the final Reed-Muller spectral coefficients c_i that correspond to the contribution of the cube cu_i of degree m to the full n - space Reed-Muller spectrum of the Boolean function F described by the array of all the disjoint cubes.

Property VII.8: Each final Reed-Muller spectral coefficient is obtained by adding modulo 2 all partial coefficients of all the disjoint cubes describing the function.

Property VII.9: The number of partial Reed-Muller spectral coefficients $npsc$ describing the Boolean function F is equal to the number of ON- cubes describing the function times 2^n .

Property VII.10: When the Boolean function is described by its truth table (a set of true and false minterms) the number of partial spectral coefficients of the Boolean function in this representation is equal to the sum of the number of ON- minterms times 2^n . Let symbol f denote the number of false minterms. Then, $npsc = 2^n (2^n - f)$.

In Table VI and Table VII each row of symbols "0" and "1" next to the cube gives the values of the partial spectral coefficients of the given cube. The order of each of these coefficients is the same as the order of the piterm placed in the top of the column above the particular partial spectral coefficient.

Definition VII.9: A cube is represented by a *positional notation* where "0" corresponds to a negated literal, "1" to an affirmative literal, and "X" to the literal that is absent in a given cube.

For example, the cube $x_1 \bar{x}_3 x_4$ is represented by 1X01.

Definition VII.10: A piterm π_i^ω is represented by a *positional notation* that is the same as in Definition VII.9 for cubes.

For example, the piterm $\pi_{134}^{0101} = x_1 x_3 \bar{x}_4$ is represented by 1X10.

Definition VII.11: A cube *adjusted to a polarity* ω is a cube built of the literals of the original cube that are different from the bits of the polarity ω in the SBC. These literals are marked by the symbol "-".

TABLE IV
ADJUSTMENT OPERATOR BETWEEN CUBE AND POLARITY

Cube	Polarity	
	0	1
0	-	0
1	1	-
X	X	X

The adjustment operator is defined in Table IV. When the cube $cu_i = 1X01$ is adjusted to the polarity $\omega = 0000$ then it is equal to $cu_i^{0000} = 1X-1$, and when the same cube is adjusted to the polarity $\omega = 0101$ then it is equal to $cu_i^{0101} = 1X--$.

Definition VII.12: A cube and a piterm of the same degree *match exactly* when both of them have the same literals in the same polarity ω or the cube has some literals that have been created by the adjustment operation and are marked "-".

TABLE V
EXACT MATCHING OPERATOR BETWEEN CUBE AND PITERM

Cube	Piterm		
	0	1	X
0	1	0	0
1	0	1	0
X	0	0	1
-	1	1	1

The exact matching operator is defined for bits of a piterm and a cube in positional notation in Table V. When a cube is adjusted to some polarity the value of the exact matching operator for the bits of the adjusted cube marked "-" is always 1 regardless of the value of the piterm. This is shown in the fourth row of Table V. From this point on it will be understood that the exact matching operation is defined by Table V.

Definition VII.13: A cube and a piterm of the same degrees *match in all but one literal* when there exists one literal in the piterm that is negated in the cube and the rest of the literals in both the cube and piterm match exactly.

The above definition can be extended to a more general case.

Definition VII.14: A cube and a piterm of the same degrees *match in all but k literals* when there exist k literals in the piterm that are negated in the cube and the rest of the literals in both cube and piterm match exactly.

Definition VII.15: A cube and a piterm of the same degree *match opposing* when all the literals in the piterm are in negation to the literals of the cube, i.e., literals in the cube are in the polarity $\bar{\omega}$ ($\bar{\omega}$ means bit by bit negation).

Definition VII.16: A *set of expanded piterms*, for a given cube and a piterm that matches it in all but k literals, is composed of the highest order piterm that exactly matches plus all the higher order piterms that are formed from the expansion of the literals of the exactly matching piterm by the remaining literals of the cube in such a way that the resulting piterms match the given cube in either all but one, all but two, ..., all but k literals.

Property VII.11: A set of expanded piterms from Definition 16 has 2^k members.

Property VII.12: All members of the set of expanded piterms match exactly the adjusted cube for which they are generated.

Property VII.13: The order of the highest order member of the set of expanded piterm of a given cube is not higher than the order of the adjusted cube itself.

The next property is a consequence of Property VII.13.

Property VII.14: The number of the piterms that has to be checked for exact matching in order to generate the whole set of expanded piterms for an adjusted cube of order z is equal to $C_n^1 + C_n^2 + C_n^3 + \dots + C_n^z$, where n is the number of variables of a Boolean function. It can be seen that $n > z$.

VII.3 ALGORITHM TO GENERATE GRME FROM DISJOINT CUBES

This algorithm refers to properties and definitions from the previous section. It is assumed that a Boolean function is described by an array of disjoint ON- cubes that can be generated by the algorithm presented in Chapter V. During execution of the algorithm the values of the partial spectral coefficients corresponding to adjusted cubes are stored in a temporary array. The dimension of the array is $m \times 2^n$, where m is a number of

disjoint cubes that represent an n variable Boolean function.

Algorithm VII.1: Generation of GRME from disjoint cubes.

step 1.

Set a number of disjoint cubes and adjust all piterms to a given polarity ω .

step 2.

Adjust all disjoint cubes to a given polarity ω .

step 3.

For each adjusted cube generate a set of all piterms that match exactly.

The operation of exact matching starts with the piterms that have the same order as the cube and continues for piterms of all smaller orders.

If all cubes are matched go to step 4 else repeat step 3.

step 4.

Add, modulo 2, each column of the temporary array. The result describes Reed-Muller spectral coefficients of the Boolean function.

A detailed example of the execution of this algorithm is shown next.

Example VII.1:

An example of the calculation of a Reed-Muller canonical expansion for a four variable completely specified Boolean function described by a set of disjoint cubes in positional notation is shown in Table IX and Table X. In order to obtain the values of all Reed-Muller spectral coefficients the columns of the partial spectral coefficients from the temporary array corresponding to all cubes describing a given Boolean function are added modulo 2.

Let us show the execution of all of the steps of the algorithm simultaneously for both tables. First, the number of cubes is determined to be 6. The piterms for Table IX are in

the polarity $\omega = 0000$, and for Table X they have polarity $\omega = 0101$. The piterms from both tables have the same symbols. It should be noticed, however, that piterm π_{1234} from Table IX corresponds to group $x_1 x_2 x_3 x_4$ while piterm π_{1234} from Table X corresponds to group $x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$, etc.

Secondly, all the cubes are adjusted to the given polarities. The results of this operation for both tables are shown in Table VI.

TABLE VI

CUBE ADJUSTMENT TO POLARITIES 0000 AND 0101

Cube	Table 6	Table 7
	0000	0101
1X00 ON	1X--	1X-0
1X1X ON	1X1X	1X1X
0X11 ON	-X11	-X1-
010X ON	-1-X	---X
1101 ON	11-1	1---
0110 ON	-11-	--10

TABLE VII

GENERATION OF EXPANDED SET OF PITERMS FOR POLARITY 0000

Cube	Piterm			
1X--	1XXX	1X1X	1XX1	1X11
1X1X	1X1X			
-X11	XX11	1X11		
-1-X	X1XX	11XX	X11X	111X
11-1	11X1	1111		
-11-	X11X	111X	X111	1111

In the third step, the exact matching operation generates the expanded set of piterms. The result of this operation for the adjusted cubes from Table IX is shown in Table VII and from Table X in Table VIII.

TABLE VIII
GENERATION OF EXPANDED SET OF PITERMS FOR POLARITY 0101

Cube	Piterm							
1X-0	1XX0	1X10						
1X1X	1X1X							
-X1-	XX1X	1X1X	XX10	1X10				
---X	XXXX	1XXX	X0XX	XX1X	10XX	1X1X	X01X	101X
1---	1XXX	10XX	1X1X	1XX0	101X	10X0	1X10	1010
--10	XX10	1X10	X010	1010				

For the sake of explanation Table VII is rewritten to Table IX, and Table VIII to Table X.

TABLE IX
REED-MULLER SPECTRAL COEFFICIENTS FOR POLARITY 0000

$x_1 x_2 x_3 x_4$	π_0	π_1	π_2	π_3	π_4	π_{12}	π_{13}	π_{14}	π_{23}	π_{24}	π_{34}
1X00 ON	0	1	0	0	0	0	1	1	0	0	0
1X1X ON	0	0	0	0	0	0	1	0	0	0	0
0X11 ON	0	0	0	0	0	0	0	0	0	0	1
010X ON	0	0	1	0	0	1	0	0	1	0	0
1101 ON	0	0	0	0	0	0	0	0	0	0	0
0110 ON	0	0	0	0	0	0	0	0	1	0	0
	0	1	1	0	0	1	0	1	0	0	1

$x_1 x_2 x_3 x_4$	π_{123}	π_{124}	π_{134}	π_{234}	π_{1234}
1X00 ON	0	0	1	0	0
1X1X ON	0	0	0	0	0
0X11 ON	0	0	1	0	0
010X ON	1	0	0	0	0
1101 ON	0	1	0	0	1
0110 ON	1	0	0	1	1
	0	1	0	1	0

Note that both pairs of tables contain the same information but with a different notation. First, the columns cube in Table IX and Table X correspond to the cubes before adjustment (see Table VI). Secondly, the members of the set of expanded piterms from Table VII and Table VIII are marked in Table IX and Table X by 1-es accordingly. In both

tables all the partial coefficients are given for each cube - not only those that have 1 values (that mark the piterms from the set of expanded piterms) but also the partial spectral coefficients that have 0 values.

The last step of the algorithm is the operation of addition modulo 2 of all the partial spectral coefficients. The result of this operation is shown in the bottom rows of Table IX and Table X.

TABLE X
REED-MULLER SPECTRAL COEFFICIENTS FOR POLARITY 0101

$x_1 x_2 x_3 x_4$	π_0	π_1	π_2	π_3	π_4	π_{12}	π_{13}	π_{14}	π_{23}	π_{24}	π_{34}
1X00 ON	0	0	0	0	0	0	0	1	0	0	0
1X1X ON	0	0	0	0	0	0	1	0	0	0	0
0X11 ON	0	0	0	1	0	0	1	0	0	0	1
010X ON	1	1	1	1	0	1	1	0	1	0	0
1101 ON	0	1	0	0	0	1	1	1	0	0	0
0110 ON	0	0	0	0	0	0	0	0	0	0	1
	1	0	1	0	0	0	0	0	1	0	0

$x_1 x_2 x_3 x_4$	π_{123}	π_{124}	π_{134}	π_{234}	π_{1234}
1X00 ON	0	0	1	0	0
1X1X ON	0	0	0	0	0
0X11 ON	0	0	1	0	0
010X ON	1	0	0	0	0
1101 ON	1	1	1	0	1
0110 ON	0	0	1	1	1
	0	1	0	1	0

The resulting 4-variable Reed-Muller expression for Table IX takes the form (the symbol " \oplus " stands for the sum modulo 2):

$$F(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_1 x_2 \oplus x_1 x_4 \oplus x_3 x_4 \oplus x_1 x_2 x_4 \oplus x_2 x_3 x_4.$$

The resulting 4-variable Generalized Reed-Muller expression for Table X is as follows (the symbol " \oplus " has the same meaning as above):

$$F(x_1, \bar{x}_2, x_3, \bar{x}_4) = 1 \oplus \bar{x}_2 \oplus \bar{x}_2 x_3 \oplus x_1 \bar{x}_2 \bar{x}_4 \oplus \bar{x}_2 x_3 \bar{x}_4.$$

Note that the original set of disjoint cubes can be taken directly as one possible mixed polarity Generalized Reed-Muller form [18], [39] and [40] of the considered Boolean function.

CHAPTER VIII

GENERALIZATIONS, ORDERINGS AND CIRCUIT INTERPRETATIONS OF SPECTRAL TRANSFORMS

VIII.1 DESCRIPTION

This Chapter presents a unified approach to different transforms used in logic design. It shows that all of the transforms that are discussed can be generalized. Each of the transforms can be defined for an arbitrary ordering. All the transforms can be applied to completely and incompletely specified logic functions that are coded in both R and S codings. Finally, each spectral coefficient has a meaning in terms of standard logic gates and this is shown for different transforms.

The extensions of previously known transforms presented here and those introduced by the author transforms are important from the point of view of the application of these transform extensions in logic design. It is, for example, well known that for some Boolean functions the best circuit realization (from the point of view of the number of gates and the number of inputs) is based not on the Reed-Muller canonical expression of zero polarity but on the expression for some other polarity [19]. The same applies to generalizations of other transforms: Adding, Arithmetic, and Walsh. Only a generalized Walsh transform is introduced here because generalized Arithmetic and Adding transforms were already presented in Chapter VI. The definition of generalized Walsh transforms also applies to two-dimensional map representations of multiple-valued input binary functions (Chapter IV).

The importance of different orderings of Walsh transforms when such transforms

are used in signal processing is discussed in Section IX.2. Either Rademacher or Hadamard orderings are used when Walsh transforms are applied to logic design. The choice of the ordering depends on the particular application. For example, the Rademacher ordering is used for the classification of logic functions [21] and [22], the Hadamard ordering is preferred for testing of circuits [22]. The Gray Code Ordered Walsh transform introduced by the author is presented in the next Chapter. It is advantageous from the point of view of the number of connections needed for its hardware implementation.

VIII.2 GENERALIZED WALSH TRANSFORM

When Walsh transforms are applied to logic functions the concept of standard trivial functions can be introduced (Property III.6). Note that for this Property all of the logic variables are in affirmation. Therefore, the Walsh transform discussed in Chapter II, Chapter III, Chapter IV, and Chapter V can be regarded as the zero polarity transform. Polarity numbers for Adding and Arithmetic transforms were given by Definition VI.3 and the same definition is valid for the case of the generalized Walsh transform.

Definition VIII.1: A *Generalized Walsh* transform of a logic function F for a given polarity is a Walsh transform that has standard trivial functions described by variables of the logic function that are either in affirmation or negation depending on the chosen polarity.

Example VIII.1:

An example of the calculation of a Generalized Walsh transform in the polarity 0011 for the four variable Boolean function of Example III.1 is shown in Figure 27.

T	M	=	S
1	1		2
1	-1		6
1	1		2
-1	1		-2
-1	1		-2
1	-1		6
-1	1		2
-1	-1		6
-1	1		-6
-1	-1		6
1	-1		2
-1	1		-6
-1	1		2
1	1		-6
-1	-1		2
1	-1		-2
1	-1		2
1	-1		-2

s_0
s_4
s_3
s_2
s_1
s_{34}
s_{24}
s_{23}
s_{14}
s_{13}
s_{12}
s_{234}
s_{134}
s_{124}
s_{123}
s_{1234}

Figure 27. Calculation of generalized Rademacher-Walsh spectrum.

It is obvious that a generalized Walsh transform can be defined not only for the Rademacher-Walsh ordering shown in Example VIII.1 but for an arbitrary ordering as well. When the resulting spectrum for the same Boolean function with 0000 polarity (Figure 5) and 0011 polarity are compared it is easy to see that the magnitudes of all the spectral coefficients are the same. The only differences are the signs of the following spectral coefficients: $s_1, s_2, s_{24}, s_{23}, s_{14}, s_{13}, s_{234}$ and s_{134} .

In order to modify Algorithm V.1 to calculate Generalized Walsh transforms, only one property that this algorithm is based on has to be changed. Property V.6 should be replaced by the following property. The algorithm itself remains the same.

Property VIII.1: If in a given cube the x_i variable of a Boolean function is in affirmation, and the bit of polarity number corresponding to this variable is 0, then the sign of the corresponding first order coefficient is positive; otherwise for the same polarity of this variable, when a variable in the cube is in negation, the sign of the corresponding first order coefficient is negative. If in a given cube the x_i variable of a Boolean function is in affirmation, and the bit of polarity number corresponding to this variable is 1, then the sign of the corresponding first order coefficient is negative; otherwise for the same polarity of this variable, when a variable in the cube is in negation, the sign of the corresponding first order coefficient is positive.

VIII.3 DIFFERENT ORDERINGS OF ARITHMETIC, ADDING AND REED-MULLER TRANSFORMS

The Arithmetic and Adding transforms introduced in Chapter VI and the Reed-Muller transform discussed in Chapter VII were presented only for one ordering, which, like the Walsh transforms, will be called the Hadamard ordering for Arithmetic, Adding, or Reed-Muller transforms. The Arithmetic transform in Hadamard ordering was presented in Figure 20. When all of the -1 symbols are replaced by 1 and all of the other symbols remain unchanged the transform matrix from Figure 20 represents either the Adding transform matrix or the Reed-Muller transform matrix in Hadamard ordering. It should be noted that both Reed-Muller and Adding transform matrices are the same - the only difference between both transforms is the way in which the multiplication of the transform matrix with a data vector is performed. The operations used for Adding transforms are standard multiplication and addition while for Reed-Muller transforms they are multiplication modulo 2 and addition modulo 2.

Since the relations discussed above between Arithmetic, Adding, and Reed-Muller transforms are valid in any ordering, only some chosen orderings of Arithmetic transforms are shown. From the explanation above the reader can easily derive transform matrices for Adding and Reed-Muller transforms from the Arithmetic transform.

$$\begin{array}{c}
 \text{AR} \\
 \left[\begin{array}{cccccccccccccccc}
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\
 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\
 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 -1 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \\
 -1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 \\
 -1 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 \\
 -1 & 1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & 1
 \end{array} \right]
 \begin{array}{c}
 X_R \\
 \left[\begin{array}{c}
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 1 \\
 0 \\
 1 \\
 1 \\
 1 \\
 1 \\
 0 \\
 0 \\
 0 \\
 1 \\
 1
 \end{array} \right]
 \end{array}
 =
 \begin{array}{c}
 C \\
 \left[\begin{array}{c}
 0 \\
 1 \\
 0 \\
 0 \\
 0 \\
 -1 \\
 0 \\
 0 \\
 0 \\
 1 \\
 0 \\
 1 \\
 -1 \\
 -1 \\
 0 \\
 1
 \end{array} \right]
 \begin{array}{c}
 c_0 \\
 c_4 \\
 c_3 \\
 c_2 \\
 c_1 \\
 c_{34} \\
 c_{24} \\
 c_{23} \\
 c_{14} \\
 c_{13} \\
 c_{12} \\
 c_{234} \\
 c_{134} \\
 c_{124} \\
 c_{123} \\
 c_{1234}
 \end{array}
 \end{array}$$

Figure 28. Calculation of Arithmetic spectrum in Rademacher ordering.

Arithmetic transforms can be created with all of the known orderings of Walsh transforms. Since the notion of generalized Arithmetic, Adding and Reed-Muller transforms for Hadamard ordering have already been discussed in Chapter VI and Chapter VII the following example is only for 0 polarity. The following example shows the calculation of an Arithmetic transform for a Boolean function.

Example VIII.2:

An example of the calculation of an Arithmetic transform in Rademacher ordering for 0000 polarity for the four variable Boolean function of Example VI.1 is shown in Figure 28.

The values of all the spectral coefficients for Arithmetic transforms in Hadamard and Rademacher orderings are the same as can be seen by comparing Figure 20 with Figure 28. The only difference is the order in which the coefficients are generated.

From the above example one can see how to change an Arithmetic transform matrix in order to obtain other possible orderings that are important for Walsh transforms: Paley, Kaczmarz, Cal-Sal, and X.

VIII.4 CIRCUIT INTERPRETATIONS OF DIFFERENT SPECTRAL COEFFICIENTS

Each spectral coefficient can be expressed in terms of a standard trivial function. For example, recall Property III.6 for Walsh spectral coefficients. The notion of a standard trivial function can be used for any transform and standard trivial functions for Arithmetic, Adding (and therefore also for Reed-Muller transforms) were shown in Chapter VI. Examples of standard trivial functions for generalized Arithmetic and Adding transforms were shown also in Chapter VI.

Based on the links between the spectral and classical approaches to logic design (Sections IV.5 and VI.4) the circuit interpretation of each spectral coefficient for any transform can be found. The circuit interpretation of a given coefficient, like the standard trivial function corresponding to such a coefficient, depends only on the polarity and is independent of the ordering of a given transform. Since the circuit interpretation of Reed-Muller transforms is known from [19], only generalized Walsh, Arithmetic and Adding transforms will be discussed here. The following property is valid for all

transforms in both R and S codings.

Property VIII.2: Each spectral coefficient of a generalized transform represents the logic circuit corresponding to a standard trivial function of the coefficient. The affirmation or negation of the variables describing a given standard trivial function agrees with the polarity of a given generalized transform.

All the following examples concern different spectral coefficients of a four variable Boolean function $F(x_4, x_3, x_2, x_1)$.

Example VIII.3:

Spectral coefficient s_{123} of a Walsh transform with 0000 polarity describes the function $x_1 \oplus x_2 \oplus x_3$.

The same spectral coefficient for Walsh transform with 0001 polarity describes the function $\overline{x_1} \oplus x_2 \oplus x_3$.

As can be seen from the above example with 0001 polarity the variable x_1 is negated because the least significant bit (LSB) of the polarity corresponding to it is equal to 1.

Example VIII.4:

Spectral coefficient add_{14} from Adding transform in polarity 0000 describes the function $x_1 x_4$.

When the polarity 1001 is used for the same spectral coefficient of Adding transform then this coefficient describes the logical function $\overline{x_1} \overline{x_4}$.

Example VIII.5:

Spectral coefficient arr_{14} from Arithmetic transform in polarity 0000 describes the function $(x_1 \oplus x_4)x_2 x_3$.

When the polarity 1001 is used for the same spectral coefficient of Adding transform then this coefficient represents the logical function $(\overline{x_1} \oplus \overline{x_4})x_2 x_3$.

CHAPTER IX

ALGORITHM AND ARCHITECTURE FOR GRAY CODE ORDERED FAST WALSH TRANSFORM

IX.1 DESCRIPTION

Usually Walsh functions are derived from the product of one or more Rademacher functions [51], [60], [63], [64], [73] and [74]. This principle has been used to build hardware Walsh function generators which, unfortunately, have orthogonality errors [63]. The latest designs by Besslich have overcome this problem by using synchronous clocking [57], [65] and [83]. Global Walsh function generators have been built that produce three different ordered outputs, that is, *Natural* (known also as *Hadamard*), *Strict Sequency* (known also as *Walsh* or *Walsh - Kaczmarz*) and *Dyadic* (known also as *Paley*). These three Walsh transforms are *symmetric*, i.e., the *inverse transform* for each of them is the same as the *forward transform* except for a constant coefficient. Microprocessor based generators of basic three ordered sequences have also been built [78]. Besides these three symmetric Walsh transforms there is a *non-symmetric* one, known in the literature as the *Rademacher-Walsh* transform, in which the Rademacher functions are used directly as the entries for the first $n + 1$ rows in the transform matrix of order $N = 2^n$.

Walsh function generators are built for applications in communication, signal analysis and synthesis, sequency filtering, multiplexing and, encoding and decoding [51], [64], [69], [78] and [120].

One other ordering of Walsh transforms, known as the *Cal-Sal* ordering, has the

advantage of providing discrimination between signals having even or odd symmetry [51] and [85]. In this ordering the first half of the entries are arranged in the order of the increasing number of even zero crossings (corresponding to Cal functions) while the second half of the entries are arranged in the order of the decreasing number of odd zero crossings (corresponding to Sal functions).

Besides generating Walsh transforms by using Rademacher functions, alternative constructions of Walsh functions have been derived by using the concepts of *symmetric copying* and *shift copying*. Originally, developed by Swick [80], this method was applied by Zhihua and Qisham [87] to the three known symmetric orderings mentioned previously, and also to derive a new Walsh function ordering which they called the *X-ordering*. The X-ordered Walsh transform has the following features: Lower order X-ordered entries correspond to even functions. Even order X-ordered entries are associated with lower sequencies, while odd ones correspond to higher sequencies [87].

The relationships between the three Walsh function ordering methods (Natural, Strict Sequency and Dyadic) are well known and have been described in references [51], [64], [75] and [84]. The author has observed that by using the previously discovered operations of: *bit-reversal* for the position of each component in a binary number, *Gray code conversion*, *dyadic addition*, and the *combination of some of them*, one is able to generate not only all the known Walsh orderings from the straight binary code but some new orderings as well. A procedure for carrying out some of these conversions for the known orderings was described by Yuen [84]. When the sequence of these operations is changed then it is possible to generate two new Walsh transforms in *Gray Code Ordering 1* and *Gray Code Ordering 2*. Hence, both new orderings can be generated by the same operations that are used for the known ordering. What is more, no more operations are needed than for the known orderings. Since matrix operations that will be defined later can be potentially useful in investigations of still other transforms and orderings there

follows a detailed description of such operations and of the orderings being introduced. The advantages of the new transforms for digital logic design have been discussed in the previous Chapter. The application of new transforms to image coding suggests itself and should perhaps be the topic of a future investigations.

One of the new transforms in Gray Code Ordering can be defined in a recursive form by using a new operator which in this dissertation is called *bi-symmetrical pseudo Kronecker product*. This new operator differs from the standard Kronecker product of matrices by having, when expanded, vertical and horizontal symmetries through the center of the expanded matrix. Of course, this property is valid for the bi-symmetrical pseudo Kronecker product operator for matrices of different orders with the restriction that the orders are even numbers and the order of one of the matrices is (2×2) . The result of applying the bi-symmetrical pseudo Kronecker product operation to two matrices will be called the *bi-symmetrical matrix*. Since Walsh type transforms are always described by square matrices of even orders this will be the only case considered. For an $N \times N$ square matrix the number N will be subject to the same restriction when considering the bi-symmetrical pseudo Kronecker product operator as it is with the Hadamard-Walsh transform generated using the standard Kronecker product. The number N has to be equal to 2^n , where n is an integer number greater than 2 [56], [68], [70], [77], [81] and [86]. Although this restriction on the order N is sufficient in the case presented here, an open question still remains whether there are bi-symmetrical matrices for which the order N is divisible by 4. The latter problem is due to the relationships that exist between the Hadamard matrix and the bi-symmetrical one (for orders higher than 2) when the Hadamard matrix of the second order H_2 is used in the expansion of the bi-symmetrical pseudo Kronecker product. It is still not known whether there are Hadamard matrices of order N for all N divisible by 4 [56]. Wallis noted that the highest order for which the existence of a Hadamard matrix is known is 268 [81].

The recursive definition of the Walsh transform in Gray code ordering is also derived like Good's formula for the standard Kronecker product [51], [58], [61], [64], [66], [67], [79] and [84]. This definition is the basis for the flow diagram architecture of a constant geometry Fast Walsh Transform in Gray code ordering.

IX.2 GRAY CODE ORDERED WALSH TRANSFORMS: RELATIONSHIPS AND METHODS OF GENERATION

The relationships between the three methods of ordering the Walsh functions: Natural, Dyadic, and Sequency, have been investigated by many authors [51], [62], [64], [70], [82], [84], [86] and [87]. The known operations for Gray code conversion, dyadic addition, bit reversal of the binary number, and combination of some of these operations are being used [51], [64] and [84]. The relationships between the X-ordering [87] and two new orderings in Gray code that use some of these operations but in a different order are shown in Figure 5.

Since there is no natural ordering of Walsh functions then investigations leading to the development of new, useful orderings, particularly those that can be described by sets of recursive equations, are of great importance [86]. The method by which Walsh functions are ordered is determined mainly by convenience. It is usually desirable to relate the order of the entries to the convergence of the signal representation so that the higher order spectral coefficients of this representation make a lower contribution to the transformed signal [82]. Sometimes another property, the *smoothness* of the spectrum defined as the function of the number of spectral component, is important [82]. It has been found that when this parameter is important, the smoothest spectrum is obtained by using the Strict Sequency ordering. On the other hand, the Natural ordering gives a spectrum of chaotic and unpredictable character [82].

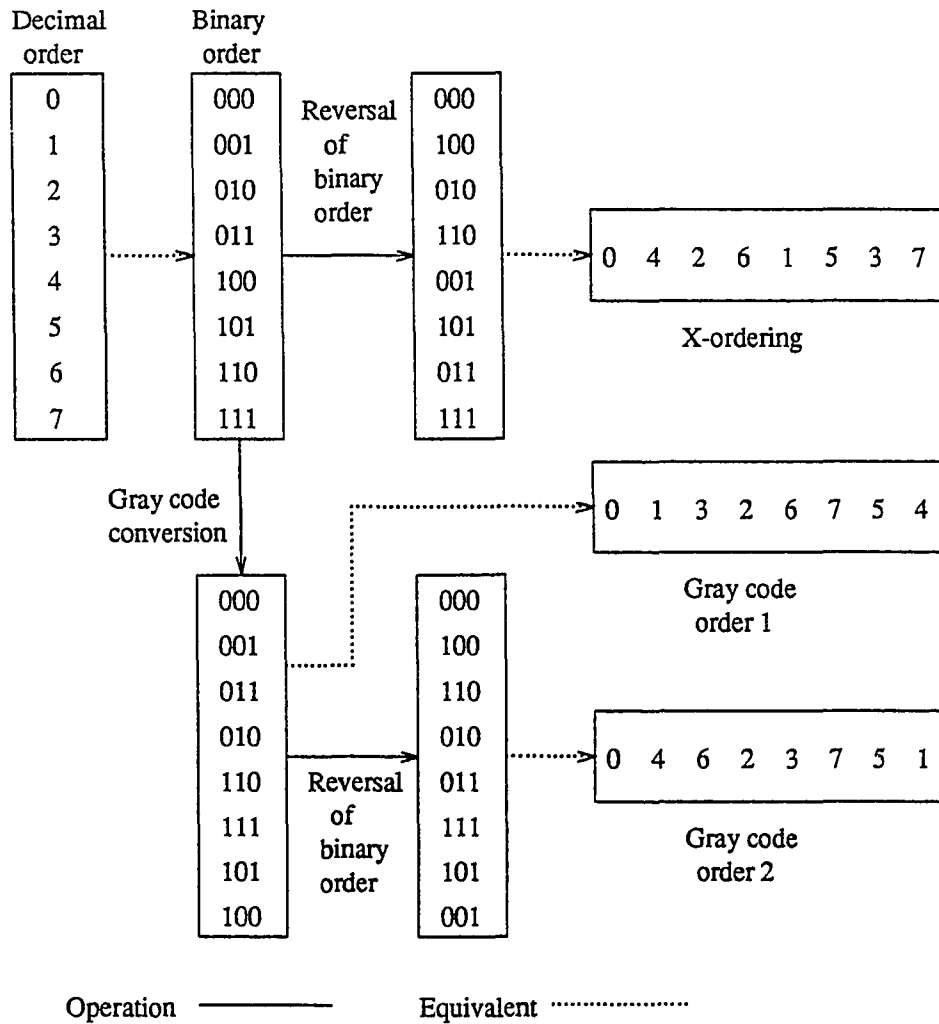


Figure 29. Relationships among 3 orderings of Walsh functions.

In order to characterize Walsh functions, Harmuth [69] coined the term *sequency* which can be expressed in terms of sign changes per unit interval (i.e., the interval for one complete cycle of the lowest sequency) as follows:

$$seq = \begin{cases} \frac{1}{2} (nzc) & \text{for even } nzc \\ \frac{1}{2} (nzc + 1) & \text{for odd } nzc \end{cases} \quad (IX.1)$$

where *nzc* stands for a number of zero crossings.

The sequencies for all previously known as well as the new orderings introduced here for Walsh transforms denoted by the names of Gray Code Orderings 1 and 2 are shown in Figure 30.

Natural		Sequency		Rademacher		Dyadic	
order	sequency	order	sequency	order	sequency	order	sequency
0	0	0	0	0	0	0	0
7	4	1	1	1	1	1	1
3	2	2	1	3	2	3	2
4	2	3	2	7	4	2	1
1	1	4	2	2	1	7	4
6	3	5	3	6	3	6	3
2	1	6	3	4	2	4	2
5	3	7	4	5	3	5	3

X		Cal-Sal		Gray Code 1		Gray Code 2	
order	sequency	order	sequency	order	sequency	order	sequency
0	0	0	0	0	0	0	0
4	2	2	1	1	1	4	2
2	1	4	2	3	2	6	3
6	3	6	3	2	1	2	1
1	1	7	4	6	3	3	2
5	3	5	3	7	4	7	4
3	2	3	2	5	3	5	3
7	4	1	1	4	2	1	1

Figure 30. Comparison of order and sequency for all Walsh orderings.

IX.3 FAST WALSH TRANSFORM FOR GRAY CODE ORDERED TRANSFORM

Two Walsh transforms with Gray Code Orderings are shown in Figure 31. The symbol + denotes +1 and - denotes -1, respectively. These two Gray Code Ordered transforms will be called Transform 1 and Transform 2 as labeled in the figure.

$ \begin{array}{c} k \longrightarrow \\ i \downarrow \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{bmatrix} + & + & + & + & + & + & + & + \\ + & + & + & + & - & - & - & - \\ + & + & - & - & + & + & - & - \\ + & + & - & - & - & - & + & + \\ + & - & + & - & - & + & - & + \\ + & - & + & - & + & - & + & - \\ + & - & - & + & - & + & + & - \\ + & - & - & + & + & - & - & + \end{bmatrix} \end{array} $	$ \begin{array}{c} k \longrightarrow \\ i \downarrow \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{bmatrix} + & + & + & + & + & + & + & + \\ + & - & - & + & + & - & - & + \\ + & - & + & - & - & + & - & + \\ + & + & - & - & - & - & + & + \\ + & + & - & - & + & + & - & - \\ + & - & + & - & + & - & + & - \\ + & - & - & + & - & + & + & - \\ + & + & + & + & - & - & - & - \end{bmatrix} \end{array} $
Transform 1	Transform 2
+ = +1	- = -1

Figure 31. Matrices for two Walsh Gray ordered transforms.

Transform 2 can be defined in a recursive way by using a new operator called the bi-symmetrical pseudo Kronecker product. Before defining this operator, three additional operations on matrices will be developed. In what follows the orders of the matrices will be denoted by capital letters, and for square matrices of order N the following relationship exists: $n = \log_2 N$.

Definition IX.1: A vertical mirror transformation of a matrix A where $A = [a_{ij}]$ of order $M \times N$ is the matrix A^{VT} of order $M \times N$ such that the first column in the matrix A^{VT} is the same as the last N^{th} column in the matrix A , the second column in the matrix A^{VT} is

the same as the $(N - 1)^{th}$ column in the matrix A , and so on.

Definition IX.2: A horizontal mirror transformation of a matrix $A = [a_{ij}]$ of order $M \times N$ is the matrix A^{HT} of order $M \times N$ such that the first row in the matrix A^{HT} is the same as the last M^{th} row in the matrix A , the second row in the matrix A^{HT} is the same as the $(M - 1)^{th}$ row in the matrix A , and so on.

Definition IX.3: A vertical - horizontal double mirror transformation of a matrix $A = [a_{ij}]$ of order $M \times N$ is the matrix A^{VHT} of order $M \times N$ such that $A^{VHT} = [A^{HT}]^{VT}$.

It is obvious that the last operator can be defined in an alternative way: $A^{VHT} = [A^{VT}]^{HT}$ since both vertical and horizontal transformations are *mutually commutative*.

Example IX.1: Figure 32 shows the results of the application of operators HT , VT , and VHT to the matrix A of order 2×4 .

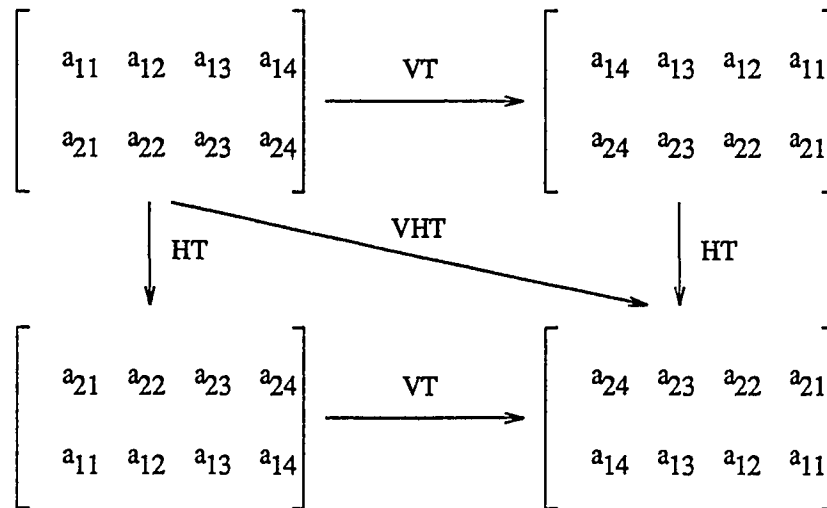


Figure 32. Transformations of the matrix A .

As was stated in the introduction to this Chapter, the new Kronecker-type operator is restricted to pairs of matrices such that at least one of them is a square matrix of order 2. The second matrix can be of any order. However, for the generation of new Walsh transforms only square matrices of orders equal to powers of 2 are needed.

Definition IX.4: A *bi-symmetrical pseudo Kronecker product* of a square, second order matrix A and a matrix B of order $(M \times N)$ is a partitioned matrix of order $(2M \times 2N)$ composed of four submatrices, where each of them is the product of one element of the first matrix and the entire second matrix or its transformation. In particular, the element a_{11} is multiplied by B , the element a_{12} by B^{VT} , the element a_{21} by B^{HT} , and finally, the element a_{22} by B^{VHT} .

Example IX.2: A comparison of the standard Kronecker product and the bi-symmetrical pseudo Kronecker product of two second order matrices A and B is shown in Figure 33.

The Walsh functions in Hadamard order are generated when the standard Kronecker product of the elementary Hadamard matrix H_2 is performed with itself. Similarly, the Gray Code Ordered Transform 2 is obtained by successive application of the bi-symmetrical pseudo Kronecker product to the core matrix H_2 . Then,

$$GK_N = (H_2)^{[n]}, \quad (\text{IX.2})$$

where $[n]$ in the exponent means the application of the bi-symmetrical pseudo Kronecker product n times.

In [59] Butin showed a permutational matrix which describes the relationship between Natural (Hadamard) and Sequency (Walsh) ordered matrices for $N = 4$. The permutational matrix P_4 that describes the relationship between the Natural (H_4) and Gray Code Ordering 2 (GK_4) is shown in Figure 34 for $N = 4$ as well.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{bmatrix}$$

$$A \overset{\text{PK}}{\otimes} B = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{12} & a_{12}b_{11} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{22} & a_{12}b_{21} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{22} & a_{22}b_{21} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{12} & a_{22}b_{11} \end{bmatrix}$$

Figure 33. Kronecker and pseudo Kronecker product of two matrices.

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

H_4
 P_4
 GK_4

Figure 34. Relationships between Natural and Gray code orders.

a)

$$\begin{bmatrix} + & + & & & & & & & \\ & + & + & & & & & & \\ & & + & + & & & & & \\ & & & - & + & & & & \\ + & - & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ + & - & & & & & & & \end{bmatrix} \quad \begin{bmatrix} + & + & & & & & & & \\ & & + & + & & & & & \\ & & & & + & + & & & \\ & & & & & & + & + & \\ & & & & & & & + & + \\ & & & & & & & - & + \\ & & & & & + & - & & \\ & & & - & + & & & & \\ & & + & - & & & & & \\ & + & - & & & & & & \end{bmatrix} \quad \text{etc.}$$

F_4 F_8

b)

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

$|F_8|^3$ GK_8

Figure 35. a) Principle of generation of matrix F, b) Factorization for N=8.

It is well known [51], [56], [58], [64], [71] and [82], that the matrix H_N describing Hadamard ordering can be factorized and represented by the n-th degree of the Good matrix G_N such that

$$H_N = (G_N)^n . \tag{IX.3}$$

Similarly, the matrix GK_N describing Gray code ordered transform 2 is also factorizable and can be represented by the following formula:

$$GK_N = (F_N)^n , \tag{IX.4}$$

where the principle of construction for the F_N matrix is shown in Figure 35 a. An example of factorization using equation (IX.4) for $N = 8$ is shown in Figure 35 b. The dotted lines on the Transform 2 matrix show the bi-symmetrical axes for the transform.

A flow diagram corresponding to equation (IX.4) is given in Figure 36 for $N = 8$.

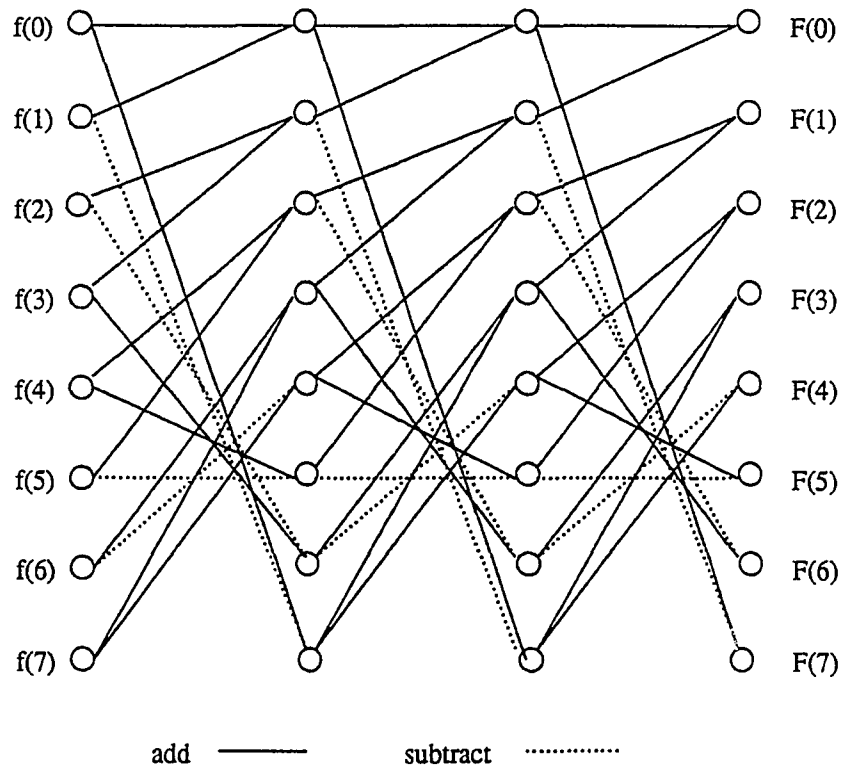


Figure 36. Flow diagram for the recursive Gray code transform.

The recursive nature of the algorithm derived from this flow diagram can be seen in the figure. The reduced computer storage requirements, achieved by the consecutive recursion steps (the total number of these steps is n), can also be seen. This algorithm is well suited for the constant-geometry hardware system presented in the next paragraph.

IX.4 CONSTANT GEOMETRY ITERATIVE ARCHITECTURE
FOR A GRAY CODE ORDERED TRANSFORM

The new Walsh transform in Gray code ordering is factored to the product of n identical matrices (equation IX.4) so this transform is well suited for constant-geometry iterative hardware systems. Since the interconnection pattern is the same for all computation stages of the new transform (Figure 36), the hardware can be reduced to only one stage and the output fed back to the input n times.

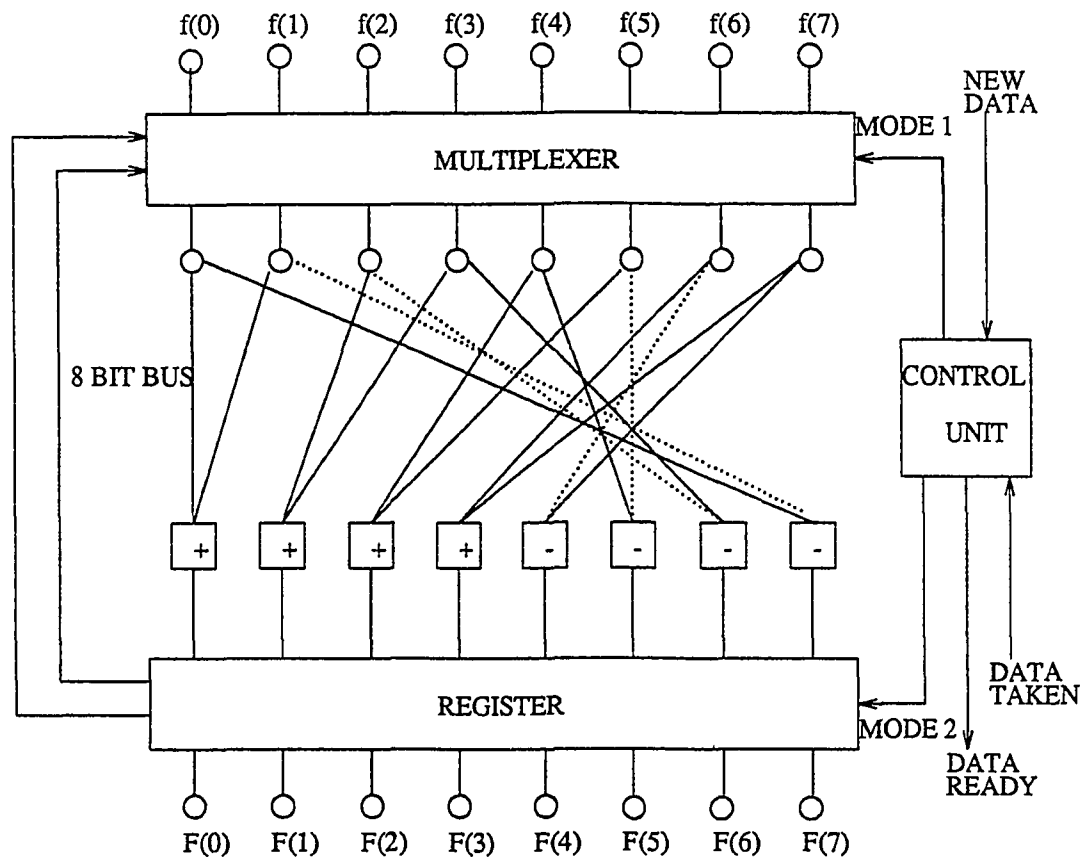


Figure 37. An iterative block of the fast Walsh transform in Gray code ordering.

The basic bloc of an *iterative implementation* of the Fast Walsh transform in Gray code ordering (for $n = 3$) is shown in Figure 37.

The basic iterative bloc is composed of a *multiplexer*, a *storage register*, a simple *control unit* and *arithmetic elements* that are able to perform ordinary *addition and subtraction* and that have a fixed arrangement of connections. The content of the storage register is fed back to the input of the multiplexer. The multiplexer is switched between two basic modes by the control unit. In the first mode new data are supplied to the arithmetic elements. In the second mode data from the storage register are applied to the arithmetic elements.

For the case of the iterative block shown in Figure 13, i.e., where $N = 8$, new data will enter through the multiplexer every third operating cycle. For the next two cycles the transformed data from the storage element will be fed back twice through the multiplexer. At the end of the conversion process, the control unit will send a *data ready* signal to report that the transformation has been performed and the spectrum is available in the storage register. When the spectrum data are taken from the storage register, a *data taken* signal is sent to the control unit which will then accept the *new data* and send it to the multiplexer for the next calculation sequence.

CHAPTER X

CONCLUSION

A new, efficient algorithm for the generation of Walsh function spectral coefficients of Boolean functions has been described. A computer method for performing this algorithm for calculating spectral coefficients has been implemented in the DIADES automation system developed at Portland State University [38], [89] and [90]. The SPECSYS (for SPECTral SYnthesis System) developed at Drexel University on a VAX 11/780 uses the Fast Walsh Transform for the calculation of the spectrum but can only process Boolean functions having up to 20 input variables [48] and [51]. The DIADES program has no limit on the number of input variables of Boolean functions and it applies the methods described in this dissertation for the generation of spectral coefficients of Boolean functions.

The authors of the SPECSYS program [16] have encountered the disadvantages which are listed below while using Fast algorithms for calculating Walsh spectra. By implementing the approach described in this dissertation, the DIADES system successfully overcomes most of these disadvantages.

The first disadvantage of SPECSYS, one which causes the use of excessive computer memory, is that the computation of the Walsh spectrum requires the representation of the Boolean function in the form of a truth table composed of minterms. While, in the DIADES system the spectrum is generated directly from the reduced representation of Boolean functions (arrays of disjoint cubes) [3], [43] and [54]. Since the number of such cubes can be considerably smaller than the number of minterms, the memory requirements can be reduced significantly. The advantages of this kind of representation, and

especially the fact that for practical functions, the number of disjoint cubes is much smaller than the number of minterms, result from [38].

The second disadvantage of SPECSYS is that all spectral coefficients must be calculated at once. In our approach, the entire spectrum - if required - can be computed incrementally for groups of coefficients - this feature does not exist in SPECSYS. Therefore our computer method is very efficient for the calculation of only the few selected spectral coefficients that is needed in many synthesis methods [20]-[25], [27]-[33], [36], [45], [46], [50] and [52].

The third disadvantage of SPECSYS is that it can only use completely specified Boolean functions. DIADES operates on systems of both completely and incompletely specified Boolean functions. The other advantages of the algorithms implemented in DIADES have been described in this dissertation. The only drawback of the DIADES's approach is the exponential growth of hard disk storage requirements with the increase of the number of coefficients. This is inherent to the nature of the problem. In SPECSYS the storage requirements are even worse since internal memory is all that is used. The implementation of the algorithm described in [89] and [90] allows the calculation of the spectrum for completely and incompletely specified Boolean functions having up to 32 variables. A detailed comparison of execution times for the DIADES and SPECSYS systems is done in [89] and [90]. Since our system can calculate coefficients either by groups (whole orders - see Property III.4) or separately, in the worst case we require only enough memory to hold the first order spectral coefficients. The $n \leq 32$ constraint refers to the generation of either the complete order or the whole spectrum. It should, however, be noticed that even for the cases when n is limited, it can be increased when a list structure that describes the indices (see Property III.5) is created. With such a list, the spectrum of a Boolean function having an arbitrary number of variables can be calculated - the only limitation being the available memory on the hard disk. When the coefficients

are calculated separately then even with the current implementation there is no limit on n since the coefficients can be stored in groups on the hard disk. This increase in problem size is, however, traded off for the increased processing time. When the whole spectrum is not required, the algorithm can calculate chosen spectral coefficients for Boolean functions of an arbitrary number of variables.

Computer algorithms similar to the one presented in this dissertation have already been developed for the newly introduced Generalized Arithmetic and Adding transforms [104], Walsh type transforms of completely and incompletely specified multiple-valued input binary functions [103] and for Reed-Muller transforms (Chapter VII). The decomposition and linearization methods for the spectra of systems of incompletely specified Boolean functions with Restriction V.1 are presented in [15]. A suggested goal for future research is the development of new decomposition and linearization methods for systems of arbitrary Boolean functions. One possible approach to this problem is to apply the known methods of [15] to $S_{TOT ON}$ and $S_{TOT DC}$ in turn. More investigations are needed in this area.

The essential relationships between classical and spectral methods used in the design of digital circuits have been stated. Resulting from these relations new algorithms for the generation of spectral coefficients for both S and R spectra for completely as well as incompletely specified Boolean functions have been shown. The graphical method for the calculation of spectral coefficients directly from a Karnaugh map is a powerful and efficient tool for functions with the number of variables less than or equal to six. All possible alternative formulas for the calculation of s_I and r_I spectral coefficients are also presented.

The fundamental formulas presented in Chapter III are very helpful when used for the investigation of new transforms, relations among various transforms and the relationships between classical logic analysis and synthesis methods and spectral methods, espe-

cially when one attempts to explain the meaning of new concepts using well-established notions. Understanding these principles gives us the working tool to translate in both directions the notions of classical and spectral theories, design of new hardware realizations for various transforms (including also those that are different from Walsh type), testing procedures, and synthesis methods.

The interpretations and algorithms, analogous to those presented in Chapter III, for only s_I and r_I spectral coefficients can be derived in a similar way for the weighted sum of the spectral coefficients as well as for the autocorrelation function of the Boolean function [32], [45] and [46]. Both these parameters of Boolean functions have been found very useful in testing. For example, the testing of programmable logic arrays by the weighted sum of the spectral coefficients provides 100% coverage of all stuck-at faults and very high coverage of multiple-faults [45].

A new concept of a spectral transform for a multiple-valued input binary function has been introduced. Such a transform is composed of a vector of transforms of all pairs of the function input variables.

The class of these new transforms corresponds to the well-known transforms for Boolean logic and can find analogous applications in classification, analysis, synthesis, design for testability and test generation of multiple-valued input binary functions. Such functions have been implemented as PLA's with input-variable decoders [100], PLA's with programmable encoders [102], Mixed-Radix Exclusive Sums of Products [39], or multiple-level functions [101] and have found applications in state assignment and synthesis of any type of multiple-output Boolean functions. Since the spectral methods for Boolean functions have been used successfully to realize the PLA's, multi-level circuits, and circuits with EXOR gates and due to the fact that the multiple-valued input binary functions are generalizations of the Boolean functions, it seems natural that the spectral transforms for the multiple-valued input binary functions will find applications in

analysis, synthesis and testing of all the circuits mentioned above.

Classical Walsh transforms have applications to the design with multiplexers [29], decomposition and design with EXOR pre-processing and post-processing circuits [15] and [21]-[24]. One can expect that similar applications can be found for the multiple-valued technologies described above. The interpretation of spectral coefficients from Chapter IV is not only useful for hand calculations of coefficients but, what is even more important, it helps to formulate new theorems and algorithms in the spectral domain by analogy to the ones in the classical domain. Some of the new developments deal with the decomposition and testing of circuits described by multiple-valued input binary functions.

It would also be interesting to investigate the relations of the new transforms with the multidimensional transforms used in image coding and the application of the new transforms to image processing. There are also possibly other formulations of transforms for multiple-valued input binary functions that do not use complex numbers as the coefficients. The work in these areas as well as formulations of the mutual relationships between different kinds of transforms are the topics of the current research of the author.

By using two types of coding, each of the three basic types of transforms considered has two types of spectra for a given Boolean function. The considerations are confined to the transforms that are created by Kronecker products of three elementary order-2 matrices. Such a limitation has been applied in order to satisfy the requirements of hardware and software realizations of transforms in recursive data-flow or systolic architectures [24], [64] and [82]. This approach enables creation of the corresponding fast transforms for each of the considered transforms, using Good's formula [24], [64].

Since the Walsh spectral coefficients have recently received a considerable attention for network analysis, synthesis and test purposes it is interesting to consider applications of the new transforms in these areas as well. For instance, the author sees the possi-

bility of using these transforms for spectral-based testing, layered Boolean network decomposition and adaptive image coding.

Besides the applications in designing and testing digital circuits the new transforms can have applications in multidimensional digital signal processing (including image processing) [82]. It is well known that the simplest representation form for images is a binary or ternary vector representation. By applying new transforms the structure of the binary or ternary images can be represented in compact form.

In Chapter VII, a new algorithm that generates fixed-polarity GRM expressions from the disjoint cube representation of Boolean functions is shown. The Boolean functions are represented in the form of arrays of disjoint cubes (Chapter V). For each cube the appropriate partial GRME is obtained. Its fixed polarity GRME is found by adding, modulo-2, all the entries corresponding to the full set of disjoint cubes describing the given Boolean function. This algorithm can be executed in parallel.

A new Walsh transform in Gray code ordering is described in Chapter IX. Hardware implementation of this transform is based on a matrix factorization that is suitable for a constant geometry iterative design. Since there is no natural ordering for Walsh functions the investigations leading to the development of new, useful orderings, particularly those that can be described by sets of recursive equations, are of great importance [86]. The method by which Walsh functions are ordered is primarily determined by convenience. It is well known, that Sequency ordering is favored by communication engineers [24], [64], [69] and [86]. On the other hand, Dyadic ordering is used in most mathematical derivations involving Walsh functions [84]-[86]. Natural and Rademacher orderings are used in the applications of Walsh functions in digital logic design [24]. Further research on the applications of two new transforms in Gray Code orderings is necessary - one can suspect that they will find many useful applications in different areas like existing transforms have. The ordering that best suits a given situation is determined

by the application.

The research, the results of which are presented here, and that which is currently continuing, will have an impact on the application of Boolean and multiple-valued input logic, not only in the synthesis, analysis, and testing of digital circuits but in areas of pattern recognition, and signal processing as well. The goal of future research is to develop new decomposition methods for systems of incompletely specified Boolean functions based on the representation of the Rademacher - Walsh spectrum presented. A major advantage of the presented approach to Walsh spectrum calculation is its convenience for computer implementation and its ability to yield solutions to problems with very high numbers of dimensions.

REFERENCES

- [1] A. C. Parker, and S Hayati, "Automating the VLSI design process using expert systems and silicon compilation," *Proc. IEEE*, vol. 75, no. 6, pp. 777-785, June 1987.
- [2] A. R. Newton, and A. L. Sangiovanni-Vincentelli, "CAD Tools for ASIC Design," *Proc. IEEE*, vol. 75, no. 6, pp. 765-776, June 1987.
- [3] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Hingham, MA: Kluwer Academic Publishers, 1985.
- [4] R. Rudell, "ESPRESSO II-C user's manual," Department of Electrical Engineering and Computer Science, University of California, Berkeley.
- [5] R. Rudell, and A. L. Sangiovanni-Vincentelli, "ESPRESSO-MV: Algorithms for multiple-valued logic minimization," *Proc. of IEEE Custom Integrated Circuits Conf.*, Portland, OR, pp. 230-234, May 1985.
- [6] J. A. Darringer, W. H. Joyner, Jr., C. L. Berman, and L. Trevilyan, "Logic synthesis through local transformations," *IBM Journal of Research and Development*, vol. 25, no. 4, pp. 272-280, July 1981.
- [7] J. A. Darringer, W. H. Joyner, Jr., C. L. Berman, and L. Trevilyan, "LSS: A system for production logic synthesis," *IBM Journal of Research and Development*, vol. 28, no. 5, pp. 537-545, September 1984.
- [8] R. K. Brayton, C. McMullen, and A. L. Sangiovanni-Vincentelli, "Automated implementation of switching functions as dynamic CMOS circuits," *Proc. of IEEE Custom Integrated Circuits Conf.*, Rochester, New York, pp. 184-188, May 1984.
- [9] A. J. de Geus, and W. Cohen, "A rule-based system for optimizing combinational logic," *IEEE Design and Test of Computers*, vol. 2, no. 4, pp. 22-33, August 1985.
- [10] R. K. Brayton, C. McMullen, and A. L. Sangiovanni-Vincentelli, "Multiple level logic optimization system," *Proc. of IEEE International Conf. on Computer-Aided Design*, Santa Clara, CA, pp. 356-361, November 1986.
- [11] R. K. Brayton, and C. T. McMullen, "The decomposition and factorization of Boolean expressions," *Proc. of IEEE Int. Symp. on Circuits & Systems*, Rome, Italy, pp. 49-54, May 1982.

- [12] E. Detjens, "FPGA devices require FPGA - specific synthesis tools," *Computer Design*, p. 124, Nov. 1990.
- [13] H. Landman, "Logic synthesis at Sun," *IEEE Conference Paper*, CH 2686-4/89/0000/0469, 1989.
- [14] M. A. Perkowski, M. Driscoll, J. Liu, D. Smith, J. Brown, L. Yang, A. Shamsapour, M. Helliwell, B. Falkowski, and A. Sarabi, "Integration of logic synthesis and high-level synthesis into the Diades design automation system," *Proc. of 22nd IEEE Int. Symp. on Circuits & Systems*, pp. 748-751, 1989.
- [15] M. G. Karpovsky, *Finite Orthogonal Series in Design of Digital Devices*. New York: Wiley, 1976.
- [16] D. Varma, and E. A. Trachtenberg, "Design automation tools for efficient implementation of logic functions by decomposition," *IEEE Trans. Computer-Aided Design*, vol. CAD-8, pp. 901-916, Aug. 1989.
- [17] C. R. Edwards, "The application of the Rademacher-Walsh transform to Boolean function classification and threshold logic synthesis," *IEEE Trans. Comput.*, vol. C-24, pp. 48-62, Jan. 1975.
- [18] M. Helliwell, M., and M. A. Perkowski, M., "A fast algorithm to minimize multi-output mixed-polarity generalized Reed-Muller forms," in *Proc. 25th ACM/IEEE Design Automation Conf.*, Anaheim, CA, pp. 427-432, June 1988.
- [19] D. Green, *Modern Logic Design*. Wokingham: Addison-Wesley, 1986.
- [20] T. Hsiao, and S. C. Seth, "An analysis of the use of Rademacher-Walsh spectrum in compact testing," *IEEE Trans. Comput.*, vol. C-33, pp. 934-938, Oct. 1984.
- [21] S. L. Hurst, *The Logical Processing of Digital Signals*. New York, NY: Crane-Russak, 1978.
- [22] S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic*. London: Academic Press, 1985.
- [23] S. L. Hurst, "Use of linearization and spectral techniques in input and output compaction testing of digital networks," *IEE Proc. Comput. & Digital Tech.*, vol. 136, pp. 48-56, Jan. 1989.
- [24] K. G. Beauchamp, *Applications of Walsh and Related Functions*. New York, NY: Academic Press, 1984.
- [25] M. G. Karpovsky, Ed., *Spectral Techniques and Fault Detection*. Orlando: Academic Press, 1985.
- [26] B. W. Kernighan, and D. M. Ritchie, *The C Programming Language*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

- [27] R. Lechner, "Harmonic analysis of switching functions," in *Recent Developments in Switching Theory*. A. Mukhopadhyay, Ed., New York: Academic Press, 1971.
- [28] A. M. Lloyd, "Spectral addition techniques for the synthesis of multivariable logic networks," *IEE Proc. Comput. & Digital Tech.*, vol. 125, no. 1, pp. 152-164, 1978.
- [29] A. M. Lloyd, "Design of multiplexer universal-logic-module networks using spectral techniques," *IEE Proc., Comput. & Digital Tech.*, vol. 127, pp. 31-36, 1980.
- [30] P. K. Lui, and J. C. Muzio, "Spectral signature testing of multiple stuck-at faults in irredundant combinational networks," *IEEE Trans. Comput.*, vol. C-35, pp. 1088-1092, Dec. 1986.
- [31] D. M. Miller, and J. C. Muzio, "Spectral fault signatures for single stuck-at faults in combinational networks," *IEEE Trans. Comput.*, vol. C-33, pp. 765-768, Aug. 1984.
- [32] D. M. Miller, Ed., *Developments in Integrated Circuit Testing*. London: Academic Press, 1987.
- [33] J. C. Muzio, and S. L. Hurst, "The computation of complete and reduced sets of orthogonal spectral coefficients for logic design and pattern recognition purposes," *Comput. & Elect. Engng.*, vol. 5, pp. 231-249, 1978.
- [34] J. C. Muzio, "Composite spectra and the analysis of switching circuits," *IEEE Trans. Comput.*, vol. C-29, pp. 750-753, 1980.
- [35] J. C. Muzio, D. M. Miller, and S. L. Hurst, "Number of spectral coefficients necessary to identify a class of Boolean functions," *Electron. Lett.*, vol. 25, pp. 577-578, 1982.
- [36] J. C. Muzio, and D. M. Miller, "Spectral fault signatures for internally unate combinational networks," *IEEE Trans. Comput.*, vol. C-32, pp. 1058-1062, 1983.
- [37] L. Nguyen, M. Perkowski, and N. Goldstein, "PALMINI - fast Boolean minimizer for personal computers," *Proc. of 24th ACM/IEEE Design Automation Conference*, pp. 615-621, 1987.
- [38] M. J. Ciesielski, S. Yang, and M. A. Perkowski, "Minimization of multiple-valued logic based on graph coloring," *Submitted to IEEE Transactions on CAD*, September 1990. Earlier version of this paper appeared as: "Multiple-valued minimization based on graph coloring," *Proc. of IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors*, pp. 262-265, 1989.

- [39] M. Perkowski, M. Helliwell, and P. Wu, "Minimization of multiple-valued input multi-output mixed-radix exclusive sums of products for incompletely specified Boolean functions," *Proc. of 19th Int. Symp. on Multiple-Valued Logic*, pp. 256-263, 1989.
- [40] M. A. Perkowski, P. Dysko, and B. J. Falkowski, "Two learning methods for a tree-search combinatorial optimizer," *Proc. of 9th IEEE Int. Phoenix Conf. on Computers & Communications*, Scottsdale, AR, pp. 606-613, March 1990.
- [41] S. M. Reddy, "Easily testable realizations for logic functions," *IEEE Trans. Comput.*, vol. C-21, pp. 1183-1188, Nov. 1972.
- [42] B. R. K. Reddy, and A. L. Pai, "Reed-Muller transform image coding," *Computer Vision, Graphics, and Image Processing*, vol. 42, pp. 48-61, 1988.
- [43] J. P. Roth, *Computer Logic, Testing and Verification*. Potomac, MD: Computer Science Press, 1980.
- [44] J. M. Sanchez, J. Ballesteros, and A. Vaquero, "Study of the complexity of an algorithm to derive the complement of a binary function," *Int. J. Electronics*, vol. 66, pp. 245-250, 1989.
- [45] M. Serra, and J. C. Muzio, "Testing Programmable Logic Arrays by sum of syndromes," *IEEE Trans. Comput.*, vol. C-36, pp. 1097-1101, Sept. 1987.
- [46] M. Serra, and J. C. Muzio, "Space compaction for multiple-output circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-7, pp. 1105-1113, Oct. 1988.
- [47] V. H. Tokmen, "Disjoint decomposability of multi-valued functions by spectral means," *Proc. of IEEE Int. Symp. on Multiple-Valued Logic*, pp. 88-93, 1980.
- [48] E. A. Trachtenberg, and D. Varma, "A design automation system for spectral logic synthesis," *Proc. of Int. Workshop on Logic Synthesis*, Research Triangle Park, NC, May 12-15, 1987.
- [49] E. A. Trachtenberg, "Designing standard computer components using spectral techniques," *Proc. of IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors*, pp. 630-633, 1987.
- [50] D. Varma, and E. A. Trachtenberg, "A fast algorithm for the optimal state assignment of large finite state machines," *Proc. of IEEE Int. Conf. on Computer-Aided Design*, Santa Clara, CA, pp. 152-155, 1988.
- [51] R. C. Debnath, and A. K. Karmakar, "Method for finding Rademacher-Walsh spectral coefficients of Boolean functions," *Int. J. Electronics*, vol. 60, pp. 245-250, Feb. 1986.

- [52] D. Varma, and E. A. Trachtenberg, "On the estimation of logic complexity for design automation applications," *Proc. of IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors*, pp. 368-371, Cambridge, MA, September 17-19, 1990.
- [53] H. S. Wilf, *Algorithms and Complexity*. Englewood Cliffs, New Jersey: Prentice-Hall, 1986.
- [54] D. L. Dietmayer, *Logic Design of Digital Systems*. Boston, MA: Allyn and Bacon, 1978.
- [55] P. Davio, J. P. Deschamps, and A. Thayse, *Discrete and Switching Functions*. New York: George and McGraw-Hill, 1978.
- [56] S.S. Aghaian, *Hadamard Matrices and Their Applications*. Lect. Notes in Mathematics No. 1168, Berlin: Springer-Verlag, 1980.
- [57] P.W. Besslich, "Walsh Function Generator for Minimum Orthogonality Error," *IEEE Trans. Electromagn. Compat.*, vol. EMC-15, pp. 177-180, Nov. 1973.
- [58] R.D. Brown, "A Recursive Algorithm for Sequency-Ordered Fast Walsh Transforms," *IEEE Trans. Comput.*, vol. C-26, pp. 819-822, Aug. 1977.
- [59] H. Butin, "Comments on 'Walsh Summing and Differencing Transforms'," *IEEE Trans. Electromagn. Compat.*, vol. EMC-18, pp. 87-88, May 1976.
- [60] J.W. Carl, "Comments on 'A Simplified Definition of Walsh Functions'," *IEEE Trans. Comput.*, vol. C-20, p. 1617, Dec. 1971.
- [61] J.W. Carl, and R.V. Swartwood, "A Hybrid Walsh Transform Computer," *IEEE Trans. Comput.*, vol. C-22, pp. 669-672, July 1973.
- [62] M. Cohn, "Walsh Functions, Sequency and Gray Codes," *SIAM J. Appl. Math.*, vol. 21, pp. 442-447, Nov. 1971.
- [63] A.C. Davies, "On the Definition of Walsh Functions," *IEEE Trans. Comput.*, vol. C-21, pp. 187-189, Feb. 1972.
- [64] D.F. Elliot, and K.R. Rao, *Fast Transforms: Algorithms, Analysis, Applications*. London: Academic Press, 1982.
- [65] L.C. Fernandez, and K.R. Rao, "Design of a Synchronous Walsh-Function Generator," *IEEE Trans. Electromagn. Compat.*, vol. EMC-19, pp. 407-410, Nov. 1977.
- [66] B.J. Fino, and V.R. Algazi, "Unified Matrix Treatment of the Fast Walsh-Hadamard Transform," *IEEE Trans. Comput.*, vol. C-25, pp. 1142-1146, Nov. 1976.

- [67] Y.A. Geadah, and M.J. Corinthios, "Natural, Dyadic and Sequency Order Algorithms and Processors for the Walsh Hadamard Transform," *IEEE Trans. Comput.*, vol. C-26, pp. 435-442, May 1977.
- [68] A. Graham, *Kronecker Products and Matrix Calculus With Applications*. Chichester: Ellis Horwood, 1981.
- [69] H.F. Harmuth, "A Generalized Concept of Frequency and Some Applications," *IEEE Trans. Inform. Theory*, vol. IT-14, pp. 375-382, May 1968.
- [70] K.W. Henderson, "Some Notes on the Walsh Functions," *IEEE Trans. Electron. Comp.*, vol. EC-13, pp. 50-52, Feb. 1964.
- [71] K.W. Henderson, "Comment on 'Computation of the Fast Walsh-Fourier Transform'," *IEEE Trans. Comput.*, vol. C-19, pp. 850-851, Sept. 1970.
- [72] K.W. Henderson, "Walsh Summing and Differencing Transforms," *IEEE Trans. Electromagn. Compat.*, vol. EMC-16, pp. 130-134, Aug. 1974.
- [73] M.I. Irshid, "A Simple Recursive Definition for Walsh Functions," *IEEE Trans. Electromagn. Compat.*, vol. EMC-28, pp. 276-279, Nov. 1986.
- [74] R.B. Lackey, and O. Meltzer, "A Simplified Definition of Walsh Functions," *IEEE Trans. Comp.*, vol. C-20, pp. 211-213, Feb. 1971.
- [75] K. Muniappan, and R. Kitai, "Walsh Spectrum Measurement in Natural, Dyadic, and Sequency Ordering," *IEEE Trans. Electromagn. Compat.*, vol. EMC-24, pp. 46-49, Feb. 1982.
- [76] J. Pearl, "Application of Walsh Transform to Statistical Analysis," *IEEE Trans. Syst. Man. Cybern.*, vol. SMC-1, pp. 111-119, April 1971.
- [77] W.K. Pratt, J. Kane, and H.C. Andrews, "Hadamard Transform Image Coding," *Proc. IEEE*, vol. 57, pp. 58-68, Jan. 1969.
- [78] S.K. Sen, S. Chatterjee, A. Deb, and A.K. Datta, "A Microprocessor Based New Software Technique for Global Walsh Function Generation," *Int. J. Electronics*, vol. 66, pp. 193-200, Feb. 1989.
- [79] J.L. Shanks, "Computation of the Fast Walsh-Fourier Transform," *IEEE Trans. Comp.*, vol. C-18, pp. 457-459, May 1969.
- [80] D.A. Swick, "Walsh Function Generation," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 167-168, Jan. 1969.
- [81] W.D. Wallis, A.P. Street, and J.S. Wallis, *Combinatorics: Room Squares, Sum-Free Sets, Hadamard Matrices*. Lect. Notes in Mathematics No. 292, Berlin: Springer-Verlag, 1972.

- [82] L.P. Yaroslavsky, *Digital Picture Processing - An Introduction*. Berlin: Springer-Verlag, 1985.
- [83] L. Yihua, "Design of a Walsh Function Generator," *IEEE Trans. Electromagn. Compat.*, vol. EMC-29, pp. 83-86, Feb. 1987.
- [84] C.K. Yuen, "Walsh Functions and Gray Code," *IEEE Trans. Electromagn. Compat.*, vol. EMC-13, pp. 68-73, Aug. 1971.
- [85] C.K. Yuen, "Comments on 'Application of Walsh Transform to Statistical Analysis,'" *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-2, p. 294, April 1972.
- [86] C.K. Yuen, "Remarks on the Ordering of Walsh Functions," *IEEE Trans. Comput.*, vol. C-21, p. 1452, Dec. 1972.
- [87] L. Zhihua, and Z. Qisham, "Ordering of Walsh Functions," *IEEE Trans. Electromagn. Compat.*, vol. EMC-25, pp. 115-119, May 1983.
- [88] A.D. Poularikas, S. Seely, *Signals and Systems*. Boston: PWS-KENT Publishing Company, 1991.
- [89] B. J. Falkowski, I. Schäfer, and M. A. Perkowski, "Effective computer methods for the calculation of Rademacher-Walsh spectrum for completely and incompletely specified Boolean functions", *Resubmitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, November 1990.
- [90] B. J. Falkowski, I. Schäfer, and M. A. Perkowski, "Effective computer methods for the calculation of Rademacher-Walsh spectrum for completely and incompletely specified Boolean functions", *Submitted to ICCAD 1991*, April 1991.
- [91] B. J. Falkowski, and M. A. Perkowski, "An algorithm for the generation of disjoint cubes for completely and incompletely specified Boolean functions," *Int. J. of Electronics*, vol. 70, no. 3, pp. 533-538, March 1991.
- [92] I. Schäfer, *An Effective Cube Comparison Method for Discrete Spectral Transformations of Logic Functions*. Master of Science in Electrical Engineering Thesis, Portland State University, May 1990.
- [93] Ph. W. Besslich, "Efficient Computer Method for EXOR Logic Design," *IEE Proc. E., Comput. & Digital Tech.*, vol. 130, No. 6, pp. 203-206, Nov. 1983.
- [94] Ph. W. Besslich, Spectral Processing of Switching Functions Using Signal-Flow Transformations, in *Spectral Techniques and Fault Detection*. (M.G. Karpovsky, ed.), Orlando: Academic Press, 1985.
- [95] D. Green, "Reed-Muller Expansions of Incompletely Specified Functions," *IEE Proc. E., Comput. & Digital Tech.*, vol. 134, No. 5, pp. 228-236, Sept. 1987.

- [96] S. L. Hurst, The Interrelationships Between Fault Signatures Based Upon Counting Techniques, in *Developments in Integrated Circuit Testing*, (D.M. Miller, ed.), London: Academic Press, 1987.
- [97] A.M. Lloyd, "A Consideration of Orthogonal Matrices, Other than the Rademacher-Walsh Types, for the Synthesis of Digital Networks," *Int. J. Electron.*, vol. 47, No. 3, pp. 205-212, March 1979.
- [98] A.J. Rubio-Ayuso, "Algebraic Representation of Boolean Functions," *Int. J. Electronics*, vol. 56, pp. 735-739, May 1984.
- [99] M. A. Perkowski, P. Wu, and K. A. Pirki, "KUAI-EXACT: A new approach for multi-valued logic minimization in VLSI synthesis," *Proc. of IEEE Int. Symp. Circ. & Systems (ISCAS)*, Portland, OR, pp. 401-404, May 1989. No. 38, 1988.
- [100] T. Sasao, "An application of multiple-valued logic to a design of programmable logic arrays," *Proc. of 8th Int. Symp. Multiple-Valued Logic (ISMVL)*, 1978.
- [101] T. Sasao, "MACDAS: Multi-level AND-OR circuit synthesis using two-variable function generators," *Proc. 23rd Design Automation Conference (DAC)*, pp. 86-93, 1986.
- [102] S. Yang, and M.J. Ciesielski, "A generalized PLA decomposition with programmable encoders," *Proc. of Int. Workshop on Logic Synthesis*, 1989.
- [103] I. Schäfer, B. J. Falkowski, and M. A. Perkowski, "An efficient computer algorithm for the calculation of Walsh transform for completely and incompletely specified multiple-valued input binary functions," *Accepted for presentation at 34th IEEE Midwest Symposium on Circuits and Systems*, Monterey, CA, May 1991.
- [104] I. Schäfer, B. J. Falkowski, and M. A. Perkowski, "A fast computer implementation of adding and arithmetic multi-polarity transforms for logic design," *Accepted for presentation at 34th IEEE Midwest Symposium on Circuits and Systems*, Monterey, CA, May 1991.
- [105] T. Damarla, and M. G. Karpovsky, "Reed-Muller spectral techniques for fault detection," *IEEE Trans. Comput.*, vol. C-38, pp. 788-797, June 1989.
- [106] E. Eris, and J. C. Muzio, "Spectral testing of circuit realizations based on linearizations," *IEE Proc. Comput. & Digital Tech.*, vol. 133, no. 2, pp. 73-78, 1986.
- [107] B. J. Falkowski, and M. A. Perkowski, "One more method for the calculation of Hadamard-Walsh spectrum for completely and incompletely specified Boolean functions," *Int. J. of Electronics*, vol. 69, no. 5, pp. 595-602, Nov. 1990.

- [108] B. J. Falkowski, and M. A. Perkowski, "Algorithms for the calculation of Hadamard-Walsh spectrum for completely and incompletely specified Boolean functions," *Proc. of 9th IEEE Int. Phoenix Conf. on Computers & Communications*, Scottsdale, AR, pp. 868-869, March 1990.
- [109] B. J. Falkowski, and M. A. Perkowski, "Essential relations between classical and spectral approaches to analysis, synthesis, and testing of completely and incompletely specified Boolean functions," *Proc of 23rd IEEE Int. Symp. on Circuits & Systems*, New Orleans, LA, pp. 1656-1659, May 1990.
- [110] B. J. Falkowski, and M. A. Perkowski, "A family of all essential radix-2 addition/subtraction multi-polarity transforms: algorithms and interpretations in Boolean domain," *Proc. of 23rd IEEE Int. Symp. on Circuits & Systems*, New Orleans, LA, pp. 2913-2916, May 1990.
- [111] B. J. Falkowski, and M. A. Perkowski, "Algorithm and architecture for Gray code ordered fast Walsh transform," *Proc. of 23rd IEEE Int. Symp. on Circuits & Systems*, New Orleans, LA, pp. 1596-1599, May 1990.
- [112] B. J. Falkowski, and M. A. Perkowski, "Walsh type transforms for completely and incompletely specified multiple-valued input binary functions," *Proc. of 20th IEEE Int. Symp. on Multiple-Valued Logic*, Charlotte, NC, pp. 75-82, May 1990.
- [113] B. J. Falkowski, and M. A. Perkowski, "On the calculation of generalized Reed-Muller canonical expressions from disjoint cube representation of Boolean functions," *Proc. of 33rd Midwest Symp. on Circuits & Systems*, Calgary, Alberta, August 1990.
- [114] B. J. Falkowski, I. Schäfer, and M. A. Perkowski, "A fast computer algorithm for the generation of disjoint cubes for completely and incompletely specified Boolean functions," *Proc. of 33rd Midwest Symp. on Circuits & Systems*, Calgary, Alberta, August 1990.
- [115] I. R. Horng, and S. J. Ho, "Synthesis of digital control systems using Walsh polynomials," *Int. J. Syst. Sci.*, vol. 17, no. 10, pp. 1399-1408, 1986.
- [116] L. W. King, P. G. Phillips, and M. Harwit, "Asymmetric Hadamard transform spectrometers," *Proc. of Int. Symp. on Applications of Walsh functions*, Washington, D.C., pp. 117-121, 1973.
- [117] J. J. Knab, "Effects of round-off noise on Hadamard transformed imagery," *IEEE Trans. Commun.*, vol. 25, no. 6, pp. 615-620, 1977.
- [118] R. D. Swift, R. B. Watson, and J. A. Decker, "Hadamard transform imager and imaging spectrometer," *Appl. Opt.*, vol. 15, no. 6, pp. 1959-1609, 1976.
- [119] G. A. Vanasse, "Hadamard matrices in spectroscopy," *Proc. of Int. Symp. on Applications of Walsh functions*, pp. 210-228, 1974.

- [120] C. J. Zarowski, and M. Yunik, "Spectral filtering using the fast Walsh transform," *IEEE Trans. Acoust. Speech, and Signal Process.*, vol. 33, no. 5, pp. 1216-1252, 1985.
- [121] S. Y. Oh, "Walsh - Hadamard based distributed storage device for the associative search of information," *IEEE Trans. Pattern Anal. and Mach. Intel.*, vol. 6, no. 5, pp. 615-623, 1984.