

1992

Orthogonal and Nonorthogonal Expansions for Multi-Level Logic Synthesis for Nearly Linear Functions and their Application to Field Programmable Gate Array Mapping

Ingo Schafer
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

Schafer, Ingo, "Orthogonal and Nonorthogonal Expansions for Multi-Level Logic Synthesis for Nearly Linear Functions and their Application to Field Programmable Gate Array Mapping" (1992). *Dissertations and Theses*. Paper 1339.

<https://doi.org/10.15760/etd.1338>

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.

ORTHOGONAL AND NONORTHOGONAL EXPANSIONS FOR
MULTI-LEVEL LOGIC SYNTHESIS FOR NEARLY
LINEAR FUNCTIONS AND THEIR APPLICATION TO
FIELD PROGRAMMABLE GATE ARRAY MAPPING

by
INGO SCHÄFER

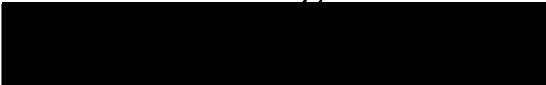
A dissertation submitted for the partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
1992


TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the dissertation of Ingo Schäfer for the Doctor of Philosophy in Electrical and Computer Engineering presented June 3, 1992.


Marek A. Perkowski, Chair



Małgorzata E. Chrzanowska-Jeske

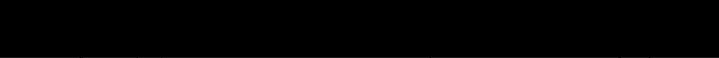

Fu Li


Yih-Chyng Jenq


Maria E. Balogh

APPROVED:


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


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

AN ABSTRACT OF THE DISSERTATION OF Ingo Schäfer for the Doctor of Philosophy in Electrical and Computer Engineering presented June 3, 1992.


Title: Orthogonal and Nonorthogonal Expansions for Multi-Level Logic Synthesis for Nearly Linear Functions and their Application to Field Programmable Gate Array Mapping


APPROVED BY THE MEMBERS OF THE DISSERTATION COMMITTEE:


Marek A. Perkowski, Chair


Małgorzata E. Chrzanowska-Jeske


Fu Li


Yih-Chyng Jenq


Maria E. Balogh

The growing complexity of integrated circuits and the large variety of architectures of Field Programmable Gate Arrays (FPGAs) require sophisticated logic design tools. In

the beginning of the eighties the research in logic design was concentrated on the development of fast two-level AND-OR logic minimizers like the well known ESPRESSO. However, most logic functions have a smaller and often faster circuit realization as a multi-level circuit. Thus, synthesis tools emerged for the minimization of the circuit area in a multi-level realization. Most of these synthesis tools are based on the "unate paradigm". Therefore, the synthesis methods are only advantageous for functions having a minimal circuit realization based on AND-OR gates. However, many common functions have a minimal circuit realization having a mix of AND, OR and EXOR gates like counters, adders, multipliers, and parity generators. Therefore, the design of such functions with synthesis tools based on the "unate paradigm" is very inefficient.

Circuits incorporating the EXOR gate have received less attention than AND-OR circuits because the EXOR gate was perceived as slower and larger in terms of its circuit realization than the AND and the OR gate. However, the upcoming of Field Programmable Gate Arrays (FPGAs) like the Xilinx Table-Look-Up (TLU) architecture the Actel *ACTTM* series and the CLi 6000 series from Concurrent Logic, which allow the realization of the EXOR gate with the same speed and circuit cost as the AND and OR gate, eliminates the disadvantages of the EXOR gate over the AND and OR gate. Thus, there is a strong need for logic synthesis tools that take advantage of EXOR gates.

The mapping to the new FPGAs recently obtained an increased interest. The developed synthesis algorithms for FPGAs are based on the mapping and restructuring of the Directed Acyclic Graph (DAG) representation of the logic function. Even though the new FPGAs allow the realization of the EXOR gate without any speed and circuit size penalty in comparison to the AND and OR gate, the synthesis methods have been based on the "unate paradigm".

To overcome the disadvantages of the current logic synthesis tools with respect to (nearly) linear functions and FPGA synthesis, this dissertation introduces an extended

theory of spectral methods for multiple-valued input, incompletely specified binary output logic. The spectral methods have not been popular in logic synthesis because of their four major drawbacks:

- (1) the computational complexity, especially if no Fast Transform exists,
- (2) the memory requirement to store the function in the necessary minterm representation,
- (3) they can not take efficiently advantage of incompletely specified functions,
- (4) suitable only for few applications in logic synthesis.

To overcome the two last stated drawbacks, this dissertation introduces the T spectrum. The T spectrum separates the information obtained for the specified and not specified parts of the underlying function. Thus, it is possible to determine directly the contribution of the specified and the not specified part of the function to a single spectral coefficient. Moreover, the T spectrum is an extension of the known spectra like Walsh-type, Adding, Arithmetic, and Reed-Muller spectra to any orthogonal and nonorthogonal transform describing logic functions. Thus, transforms can be constructed that describe certain gate structures, as for example the realizable functions of a FPGA macrocell. This allows the development of special synthesis algorithms for the different types of FPGA architectures. As an exemplification of this method, a complete multi-level synthesis algorithm is introduced for the circuit realization with multiplexer modules, which form the basic macrocell of the Actel ACTTM FPGA series.

Additionally, this dissertation presents the classification of the applications of spectral methods in logic synthesis into three categories:

- (1) The decomposition of logic functions based on the information obtained by the computation of a single spectrum. As an example the linearization procedure developed by Karpowsky is generalized to incompletely specified multi-output

Boolean functions. The linearization procedure is based on the computation of the Rademacher-Walsh spectrum with a following decomposition of the underlying function based on high value spectral coefficients.

- (2) The circuit realization of a logic function based on the repetitive application of (1). This synthesis method is exemplified by an multi-level synthesis algorithm for multiplexer gates.
- (3) The realization of a logic function as an AND-EXOR circuit based on a GF 2 (Galois Field (2)) spectrum. The GF 2 transforms exhibit the property that they describe a realization of the underlying function as a two-level AND-EXOR circuit. The Multiple-Valued Input Kronecker Reed-Muller (MIKRM) form is introduced as an application of GF 2 transforms.

To overcome the drawbacks of spectral methods concerning the computational complexity and high memory requirements, this dissertation presents a computation method for spectra from disjoint representations. The introduced application of the disjoint cube representation and the Ordered Decision Diagrams for the computation of spectra proves to be an ideal concept.

Thus, this dissertation presents general synthesis methods based on new spectral methods that overcome the deficiencies of current logic synthesis methods with respect to the synthesis for FPGAs as well as the computational complexity and memory requirements of spectral methods.

TABLE OF CONTENTS

		PAGE
	LIST OF TABLES	vi
	LIST OF FIGURES	viii
CHAPTER		
I	INTRODUCTION	1
II	ORTHOGONAL EXPANSIONS IN LOGIC SYNTHESIS	10
	II.1 Multiple-Valued Input, Binary Output Functions	11
	II.2 Discrete Orthogonal Expansions For Mv Functions ...	13
	II.2.1 Application of Orthogonal Spectra	
	II.2.2 Circuit Realization Based on Spectra	
	II.2.3 Circuit Realization Based on GF 2 transforms	
III	DISJOINT REPRESENTATIONS OF LOGIC FUNCTIONS ..	19
	III.1 Disjoint Cube Representation	20
	III.1.1 Disjoint Cube Representation for mv Functions	
	III.1.2 An Example for Boolean Functions	
	III.2 Arithmetic Cover	26
	III.3 Ordered Decision Diagrams	27
	III.3.1 Ordered Binary Decision Diagrams	
	III.3.2 Ordered Multiple-Valued Input Decision Diagrams	
IV	SPECTRAL TECHNIQUES FOR MV FUNCTIONS	34
	IV.1 Spectra of Mv Functions	34
	IV.2 Relations of Spectral Techniques to Classical Logic	37
	IV.3 The T - Spectrum	41

		iv
	IV.4 Calculation of Spectra from ODDs	43
V	EFFICIENT ALGORITHM FOR THE CALCULATION OF WALSH-TYPE SPECTRA	47
	V.1 Two-Dimensional Mapping	47
	V.2 Benchmark Results	55
VI	LINEARIZATION OF INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS	57
	VI.1 Complexity of Boolean Functions	58
	VI.2 Linear Preprocessor for Systems of Completely Specified Boolean Functions	60
	VI.2.1 General Decomposition Algorithm	
	VI.2.2 Linearization by Spectral Translation	
	VI.3 Linear Preprocessor for Systems of Incompletely Specified Boolean Functions	64
	VI.4 Calculation of the Core Function $F_{\sigma}(X)$	67
	VI.5 Evaluation and Benchmark Results	70
VII	HIERARCHICAL MULTIPLEXER SYNTHESIS FOR BOOLEAN FUNCTIONS	73
	VII.1 General Multiplexer Synthesis	76
	VII.2 Redundancy of Multiplexer Modules	79
	VII.3 Synthesis Example	88
	VII.4 Mapping of a Multiplexer Tree Circuit to the ACT 1 Macrocells	91
	VII.5 Benchmark Results	93
VIII	MULTIPLE_VALUED INPUT GENERALIZED REED-MULLER FORMS	96
	VIII.1 Canonical Multiple-Valued Input, Binary Output Generalized Reed-Muller Forms	98
	VIII.1.1 The Concept of the Polarity for a Multiple-Valued Variable	
	VIII.1.2 The MIGRM Forms	
	VIII.2 Algorithm for the Calculation of the MIGRM Form . .	108

	VIII.2.1 Transformation for One Multiple-Valued Literal	
	VIII.2.2 Transformation of a Multiple-Valued Function	
	VIII.3 A Practical Example	114
IX	ON THE MINIMAL MULTIPLE-VALUED INPUT KRONECKER REED-MULLER FORM FOR INCOMPLETELY SPECIFIED MULTIPLE-VALUED INPUT FUNCTIONS	121
	IX.1 Kronecker Reed-Muller Form	122
	IX.2 Multiple-Valued Kronecker Reed-Muller Form	124
	IX.3 Extended Truth Vector	128
	IX.4 Minimal MIKRM Form for Multi-Output Incompletely Specified <i>mv</i> Functions	133
	IX.4.1 Extension for incompletely specified <i>mv</i> functions	
	IX.4.2 Extension to systems of incompletely specified <i>mv</i> functions	
	IX.5 Calculation of the MIKRM Form from OMDDs	139
	IX.6 Evaluation and Results	143
X	CONCLUSION	146
REFERENCES	150
APPENDIX	162

LIST OF TABLES

TABLE		PAGE
I	Comparison of the number of disjoint cubes for MCNC benchmark functions	25
II	Computation times for Walsh spectrum	56
III	Linearization	71
IV	Comparison of the number of coefficients	83
V	First level spectra	90
VI	Second level spectra for f_0	91
VII	Benchmark results for multiplexer synthesis	94
VIII	All possible superpositions of the polarity literals	99
IX	The notation for the MIGRM	100
X	The spectrum of the function $F(X_1, X_2)$	105
XI	Representation of the multi-output function $F(X_1, X_2)$	106
XII	The spectrum of the function $F(X_1, X_2)$	107
XIII	Transformations for some four-valued literals	109
XIV	The code representation for the polarity literals	110
XV	The complete MIGRM spectrum of the function $G(X_1, X_2, X_3)$. .	111
XVI	The truth table of the 2-bit adder	115
XVII	The polarity for the 2-bit adder	115
XVIII	The normalized codes for the four occurring values	116
XIX	The spectrum for the first three terms from table XVI	117
XX	The MIGRM of the 2-bit adder	118

		vii
XXI	Benchmark Results for MIKRM	144

LIST OF FIGURES

FIGURE		PAGE
1.	Unique expansion for a four-variable function	15
2.	Multi-output function for disjoint sharp operation $C_a \# C_b$	21
3.	Stages of the execution of the algorithm to generate a disjoint cube representation	24
4.	OBDDs for $f(X) = x_1x_3 + \bar{x}_1x_2$ and $g(X) = x_1 \oplus x_2$	29
5.	SOBDD for $f(X) = x_1x_3 + \bar{x}_1x_2$ and $g(X) = x_1 \oplus x_2$	30
6.	OBDD of an incompletely specified Boolean function	31
7.	OMDD of $f(X_1, X_2)$	32
8.	Spectrum by matrix calculation	36
9.	Map of function $f = X^2Y^{01} + X^0Y^{34}$	36
10.	Multiple-valued transform of function f	37
11.	<i>PSTFs</i> for the Rademacher-Walsh transform	40
12.	Result of the intersection between $f(X)$ and $g(X)$, $f(X) \cap g(X)$	44
13.	OBDD of an incompletely specified Boolean function	45
14.	Intersection of the incompletely specified function $f(X)$ and $g(X)$	46
15.	Spectral map for 4-variable Boolean function	48
16.	<i>Scubes</i> for function $f(X)$	51
17.	Final Spectrum	52
18.	Complexity of a linear and a nearly unate function	59
19.	EXOR preprocessor	61
20.	EXOR preprocessor example	68

21.	Basic building block of the ACT^{TM} family	74
22.	Realizable multiplexers with ACT^{TM} family	75
23.	Standard multiplexer M(2) according to Equation (VII.4)	77
24.	Restricted Multiplexer tree circuit	78
25.	Control Function Circuit for function complementation	80
26.	Trivial functions for chosen data-select variables x_1 and x_2	81
27.	Trivial functions in OBDD form	85
28.	Incompletely specified function to 1	85
29.	Karnaugh-map of synthesis example	89
30.	Final multiplexer realization	92
31.	Mapping of complemented data-input functions	93
32.	Example of a 3×3 orthogonal matrix	99
33.	Description of polarity matrices	101
34.	Polarity matrices	101
35.	Standard trivial functions for the polarities of variables X_1 and X_2 specified in Figure 34	104
36.	The map of function $F(X_1, X_2) = X_1^{023} X_2^{01}$ from Examples VIII.2 and VIII.3	105
37.	The map of function $F_2(X_1, X_2)$	106
38.	AND-EXOR PLA implementation with input decoders of the MIGRM form of the 2-bit adder	119
39.	Input decoder for variables X_1 , and X_2	119
40.	Transform of the function $F(X_1, X_2) = X_1^{023} X_2^{01}$	128
41.	OMDD of $f(X_1, X_2)$	140
42.	Standard trivial function $u_7 = K_1^3 \cap K_2^1$	142
43.	$u_7 \cap f(X_1, X_2)$	143

CHAPTER I

INTRODUCTION

In the last two decades the complexity of digital integrated circuits grew tremendously. Therefore, sophisticated logic design tools are required to allow for their fast prototyping.

In the beginning of the eighties the research in logic design was concentrated on the development of fast two-level AND-OR logic minimizers like the well known ESPRESSO [1,2], and PALMINI [3]. However, most logic functions have a smaller and often faster circuit realization as a multi-level circuit. Thus, synthesis tools emerged for the minimization of the circuit area in a multi-level realization [4]. Synthesis tools like MISII [5], BOLD [6], SYLON [7], and RENO [8] are now the core in computer aided logic design tools as those from Mentor Graphics, Cadence or Synopsis.

Most of the above synthesis tools are based on the "unate paradigm" [9,10]. The "unate paradigm" is the assumption that most of the logic functions occurring in logic design are *unate* or *nearly unate*. The meaning of *unate* and *nearly unate* for logic minimization purposes is that the circuit realization of a (*nearly*) *unate* function with AND and OR gates is smaller in terms of the number of gates as for a circuit using the AND and the EXOR gate. On the other hand the meaning of *linear* or *nearly linear* for logic minimization purposes is that the circuit realization of a (*nearly*) *linear* function with AND and EXOR gates is smaller in terms of the number of gates as for a circuit using the AND and the OR gate. Arithmetic functions like counters, adders, multipliers, signal processing functions and error correcting logic belong to the class of (*nearly*) *linear* functions [4,11,12]. Thus, the design of such functions [11], which occur

frequently in real designs, is very inefficient with synthesis tools based on the "unate paradigm". Therefore, synthesis tools have been demanded that incorporate EXOR gates [13].

Circuits incorporating the EXOR gate have received less attention than AND-OR circuits because the EXOR gate was perceived as slower and larger in terms of its circuit realization than the AND and the OR gate. The research on Exclusive Sum of Product (ESOP) forms, the counterpart to Sum of product (SOP) forms for AND, EXOR circuits, has been mainly concentrated on canonical Reed-Muller forms [14-31]. Because of their excellent design for testability [32-37] they have been, in spite of the fact that those forms mostly lead to non-minimal circuit realizations, an alternative design approach.

Another reason that ESOP forms have been not very popular has been the lack of efficient logic optimizers for them. Recently, two-level AND-EXOR minimizers like EXORCISM [38], EXMIN [39,40], HERMES [41] and HEALEX [42,43] have been developed to design (*nearly*) *linear* logic more efficiently. But up to now, GATEMAP [44] is the only commercial logic synthesis system that at least partially incorporates EXOR gates in the design process. However, the upcoming of Field Programmable Gate Arrays (FPGAs) like the Xilinx Table-Look-Up (TLU) architecture [45] the Actel *ACTTM* series [46] the CLi 6000 series from Concurrent Logic [47], and several others [48-50] which allow the realization of the EXOR gate with the same speed and circuit cost as the AND and OR gate, eliminates the disadvantages of the EXOR gate over the AND and OR gate.

In contrast to the logic synthesis for Application Specific Integrated Circuits (ASICs), the logic synthesis for FPGAs is limited by the regular structure of the architectures. Additionally, the macrocells provided by the different FPGA architectures allow only the realization of a limited number of logic functions. An exception is the Table Look Up (TLU) architecture from Xilinx [45], where any function of up to five variables

can be implemented with a single macrocell. Therefore, mainly algorithms have been developed that map a given logic function to a chosen FPGA [51-60]. The algorithms are based on the mapping and restructuring of the Directed Acyclic Graph (DAG) representation obtained from a SOP representation of the logic function. Thus, even though the FPGAs allow the realization of the EXOR gate without any speed, or circuit size penalty these methods do not take advantage of it.

To overcome the disadvantages of the current logic synthesis tools with respect to *(nearly) linear* functions and FPGAs this dissertation introduces an extended theory of spectral methods in logic synthesis for multiple-valued input, incompletely specified binary output logic (*mv* logic). Spectral methods have the advantage over Boolean methods that they give a global information about the structure of the underlying function [61-66]. For example they allow to detect the *(nearly) linear* parts of a Boolean function. The linearization method based on the Rademacher-Walsh and the autocorrelation spectrum has been developed to extract the *(nearly) linear* part of a Boolean function [12,62,65,67]. This is the only currently known approach to include the EXOR gate into the synthesis of multi-level circuits. The linearization of a *(nearly) linear* circuit decreases the complexity of the following *(nearly)unate* core function and thus minimizes the circuit. Moreover, the resultant circuit has good testing properties [65,68,69]. Additionally, the decomposition approach based on spectral methods generates a level-by-level circuit. Therefore, such a circuit can be easily mapped to the regular structure of FPGAs.

Spectral methods also have been used in other logic synthesis areas like disjoint decomposition [62,65,66], and prime implicant extraction [65], but the computational complexity of these methods makes them inferior when compared to the classical Boolean methods.

Even though there is no Boolean counterpart for the multi-level synthesis including

EXOR gates, the spectral approach to multi-level logic synthesis has been neglected. This is due to the four severe limitations of current spectral methods:

- (i) the computational complexity,
- (ii) the memory requirement to store an mv function in the necessary minterm representation,
- (iii) the synthesis procedures based on spectral techniques can not take efficiently advantage of incompletely specified mv functions, which frequently occur in the design of sequential logic,
- (iv) only applicable to few logic synthesis applications.

The development of Fast Transforms for several discrete spectral transforms [20,65,70] has partially overcome the computational complexity. Because the mv function still has to be stored in its minterm representation, the Fast Transforms do not overcome the high memory requirements. Nevertheless, the upcoming of fast parallel computers for signal processing and matrix calculations (iWarp, iPSC) give motivation to investigate the application of spectral methods in logic synthesis. Moreover, more emphasis has been recently concentrated on minimizers that find more optimal solutions, with the trade-off of increased computation time. Thus, spectral methods in logic synthesis are becoming feasible.

In the last couple of years it has been attempted to decrease the computational complexity and memory requirement due to the necessary minterm representation of the Boolean function. New methods have been introduced that allow the calculation of Walsh-type spectra from the arithmetic cover [12] or a disjoint cube representation [10,71-74] of a Boolean function rather than minterms. The methods have been extended in [26,27,72,75] for Reed-Muller forms. Such an approach decreases the memory requirement in comparison to the minterm representation. Inherent to the algorithm presented there, the computational complexity for those methods increases linearly with

the number of terms, either in the arithmetic cover or the disjoint representation. Therefore, they are dependent on an efficient preprocessor to generate a quasi-minimal disjoint representation.

Based on the previous research on the computation of spectra from a disjoint cover [72,74] this dissertation introduces the computation of spectra from Ordered Decision Diagrams (ODDs) [10,76-80]. Their use as well as the modification is introduced to make them suitable for the application to spectral methods. The advantage of ODDs over the set of disjoint cube representation is the smaller memory requirement, especially for large functions [79,80]. They also allow fast tautology verification and fast calculation of the intersection of two mv functions. This advantage over the "set of disjoint cubes" representation makes them superior in applications such as the in this dissertation introduced algorithm for hierarchical multiplexer synthesis, which takes advantage of those features. However, for synthesis procedures that have to modify the disjoint representation the set of disjoint cubes is a more advantageous representation. Because a quasi-minimal set of disjoint cubes of an mv function is crucial for the efficient calculation of its spectra, an algorithm is presented as well as its comparison for Boolean functions to the existing algorithms.

While the application of spectral methods to logic synthesis have been limited to the known spectra like the Walsh-type, Arithmetic, Adding and Reed-Muller, this dissertation takes another approach. Instead of trying to apply one of the known spectra to a certain synthesis problem, the basis functions (also called *standard trivial functions* [71,81]) of a spectrum are defined by any set of logic functions. This allows the development of transforms which give the correlation of an mv function to certain gate structures. Such an approach is very advantageous for the logic minimization for FPGAs. For example special transforms for the multiplexer gate, which is a basic module in the Actel ACT^{TM} series, or the AND-EXOR gate which is a basic module of the CLi 6000 from

Concurrent Logic can be developed. For the illustration of this application this dissertation introduces a complete multiplexer synthesis system based on the spectrum having as *standard trivial functions* the logic functions describing the multiplexer gate. The SPECTRA system has been developed for further investigation of such special purpose transforms for Boolean functions. The SPECTRA system allows the user to calculate the spectral coefficients for a specified set of *standard trivial functions*. Additionally, the multiplexer synthesis system and the linearization procedure are incorporated in SPECTRA. The SPECTRA system itself is the spectral logic synthesis part of the Portland Logic Optimizer (POLO), which is currently under development at Portland State University. POLO is a logic optimizer for multi-level logic synthesis including EXOR gates.

An extended concept of spectral methods is introduced to overcome the deficiencies of the current spectral methods with respect to incompletely specified functions. The S coding of spectra [65], which is applied for the computation of spectra for incompletely specified *mv* functions, does not allow to develop efficient synthesis algorithms that would incorporate the not specified parts of the function. This is due to the property that the information about the not specified part can not be retrieved from single coefficients because the information is spread over the complete spectrum. Therefore, the *T*-spectrum introduced in this dissertation, which separates the information of the specified and the not specified parts of an incompletely specified *mv* function, proves to be a more useful and general concept.

The applications of the introduced *T* spectrum to logic synthesis can be categorized into three concepts:

- The decomposition of an *mv* function based on the information obtained from the spectrum. As an example the linearization procedure developed in [12,67,62] is generalized to incompletely specified multi-output Boolean functions.

- The circuit realization of an mv function based on the repetitive application of spectra. A general multi-level multiplexer synthesis algorithm [82] is introduced here as an exemplification of this concept.
- The realization of an mv function as AND-EXOR circuit based on the information given by a spectrum obtained over Galois Field (2) (GF 2). The Multiple-Valued Input Kronecker Reed-Muller (MIKRM) form [29,30] is presented as an application of GF 2 transforms.

The concepts of orthogonal and nonorthogonal transforms developed for Boolean functions are generalized to multiple-valued input, binary output logic. Such a logic has the advantage, that it can be readily implemented with currently available Application Specific Integrated Circuits (ASICs) and FPGAs [48]. While the circuits that realize the multiple-valued logic [83] are not yet widely used, our methods can be applied to their minimization as well. For the synthesis of mv functions this dissertation concentrates on canonical Reed-Muller forms. Such forms are special cases of canonical expansions over GF 2 [20,84]. This is the third basic concept of the application of spectral methods in logic synthesis. The transforms over GF 2 directly allow to determine an AND-EXOR circuit realization of a given mv function. The motivation to investigate Reed-Muller forms is the superior testability of their corresponding circuit realization [32-37,85]. However, the circuit area of such a circuit is usually larger than the one obtained by conventional two-level synthesis. This dissertation introduces the Multiple-valued Input Generalized Reed-Muller (MIGRM) form and the Multiple-valued Input Kronecker Reed-Muller (MIKRM) form and their circuit realizations. The advantage of the MIKRM form over the corresponding Boolean form is that its circuit realization has a smaller area and it preserves the testability properties.

The new approach to the logic synthesis for FPGAs incorporating (*nearly*) *linear* functions presented in this dissertation takes advantage of the power of spectral methods.

To be able to apply spectral methods for a wider range of applications the definition of spectra is generalized to *standard trivial functions* describing any logic function. Additionally, the introduced T spectrum takes efficiently advantage of incompletely specified Boolean functions. Its computation from disjoint representations like ODDs is introduced to overcome the computational complexity of current spectral methods. The results obtained for the implemented linearization procedure and the multiplexer synthesis algorithm prove the usefulness of such an approach.

The following list gives an outline of the organization of this dissertation.

The following Chapter gives the mathematical background for the spectral concepts developed here. First, a short review is given of the applications of canonic expansions in logic synthesis. Second, Chapter II introduces the special case of the Generalized Shannon Expansion of AND-OR type and over GF 2 for the special case of mv functions.

Because the introduced multi-level synthesis concepts are based on new spectral methods which depend on a disjoint representation, Chapter III presents a generalized and improved algorithm for the generation of a quasi-minimal disjoint cube representation for multi-output mv functions. Its implementation for Boolean functions and its results are compared to existing programs and the arithmetic cover [12]. Additionally, the modification of ODDs is investigated for the efficient use in spectral methods.

Chapter IV gives a short review of discrete spectral transformations for mv functions. The relations of spectra to classical logic are generalized to any orthogonal and nonorthogonal matrices having $\{0, 1, -1\}$ entries. The T -spectrum is introduced for incompletely specified mv functions. A method is presented to calculate a spectrum from an Ordered Decision Diagram (ODD) representation of an mv function.

The Walsh-type transforms, which are used for the linearization procedure presented in Chapter VI, exhibit certain properties that allow for the development of a

fast calculation method. Based on these properties Chapter V introduces a fast and memory efficient two-dimensional mapping method for the computation of the Walsh-type spectra.

The following Chapters discuss examples of the three application concepts of the introduced extended spectral methods for logic synthesis. First, Chapter VI reviews the basic linearization procedure to extract the (*nearly*) *linear* part of the underlying function. Then, based on the introduced T spectrum the linearization procedure is extended to take advantage of incompletely specified multi-output Boolean functions. Next, the results obtained by the linearization methods based on the Rademacher-Walsh and the autocorrelation spectrum are evaluated.

Another approach to incorporate (*nearly*) *linear* functions efficiently in the logic synthesis is to take advantage of the powerful multiplexer gate. Therefore, Chapter VII presents an algorithm for the synthesis of hierarchical multiplexer circuits. Their implementation with the ACT^{TM} FPGA family from Actel [46,48] is presented. Additionally, the obtained multiplexer circuit can directly be realized with the TPC10 series of Texas Instruments [50] and the CLi 6000 series from Concurrent Logic [47]. The combination of classical logic and spectral method makes the introduced method superior to the algorithm developed in [64].

The multiple-valued input Reed-Muller forms are introduced as a concept of the application of spectral methods over GF 2. Chapter VIII presents the concept of polarity matrices for *mv* logic and the computation of the multiple-valued input Reed-Muller forms by a *tabular pattern matching* method. Finally, Chapter IX introduces the multiple-valued input Kronecker Reed-Muller (MIKRM) form. It allows the calculation of the MIKRM by Fast Transforms. Additionally, a method is presented that allows its calculation from Ordered Multiple-valued input Decision Diagrams (OMDD).

CHAPTER II

ORTHOGONAL EXPANSIONS IN LOGIC SYNTHESIS

In logic synthesis forms of logic functions are of interest that lead to minimal circuit realizations with respect to area and delay of the circuit. The most common criteria for forms to be minimal in the sense of their minimal circuit realization is their literal count [1,5,86] or the number of terms of the form [41,43]. However the possible forms for a logic function are so numerous that it is nearly impossible to find the minimal among them. Therefore, there is a strong interest in normal forms which are unique [10]. Such forms are also called *canonical forms* [10]. The interest is to find a set of *canonical forms*, where one of the forms has a close to minimal circuit realization. Thus, the computational effort to find the minimal circuit realization by investigating all possible forms is reduced to finding a minimal form among a set of *canonical forms*.

In signal processing and image processing unique forms obtained by discrete orthogonal and unitary transforms are important tools [70,87,88]. The transform pair for unitary/orthogonal transforms is described by

$$X(k) = \sum_{n=0}^{N-1} x(n) g_k^*(n) \quad (\text{II.1})$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) g_k(n) \quad (\text{II.2})$$

where * denotes conjugated complex, $X(k)$ is the vector of N values in the transform domain, and $x(n)$ is the set of values describing the original function in the object domain [70]. The N transform functions $g_k(n)$ describe an unitary/orthogonal $N \times N$ transform matrix G where the transform matrix G is given by

$$[G] = \begin{bmatrix} g_0^*(0) & \dots & g_0^*(N-1) \\ \dots & \dots & \dots \\ \dots & g_i^*(j) & \dots \\ \dots & \dots & \dots \\ g_{N-1}^*(0) & \dots & g_{N-1}^*(N-1) \end{bmatrix} \quad (\text{II.3})$$

The transform pair given by Equation (II.1) and (II.2) can also be represented by the following matrix multiplications

$$X = [G] x \quad (\text{II.4})$$

$$x = [G]^{-1} X \quad (\text{II.5})$$

The most popular discrete orthogonal transforms are the Discrete Fourier, Hadamard-Walsh, Sine- and Cosine Transform [70,87]. For instance the set of linearly independent functions for the discrete Fourier transform is given by

$$g_k(n) = e^{-j \frac{2\pi \omega k n}{N}} \quad (\text{II.6})$$

where ω is the frequency variable.

This dissertation discusses some aspects of unitary/orthogonal discrete transforms in the special area of logic design. In particular, orthogonal discrete transforms for multiple-valued input, binary output logic are investigated. Therefore, first the concept of multiple-valued input, binary output functions is introduced.

II.1 MULTIPLE-VALUED INPUT, BINARY OUTPUT FUNCTIONS

This section introduces the basic concepts of multiple-valued input, incompletely specified binary output functions and their vector notation.

Definition II.1: A multiple-valued input, incompletely specified binary output function (*mv function* for short) is a mapping $f(X) = f(X_1, X_2, \dots, X_n): R_1 \times R_2 \times \dots, R_n \rightarrow B$, where X_i is a multiple-valued variable that takes the values from the set $R_i = \{0, 1, \dots, p_i - 1\}$ where p_i is the number of values of the variable,

and $B = \{ 0, 1, - \}$.

Definition II.2: A *literal* of the multiple-valued input variable X_i , denoted by $X_i^{S_i}$, is defined as:

$$X_i^{S_i} = \begin{cases} 1 & \text{if } X_i \in S_i \\ 0 & \text{if } X_i \notin S_i \\ - & \text{if } X_i \text{ is not specified} \end{cases} \quad (\text{II.7})$$

A literal of an *mv* variable with a single value will be called a *single-valued literal*. A *product of literals*, $X_1^{S_1}, X_2^{S_2}, \dots, X_n^{S_n}$ is referred to as a *product term* (also called *term* for short). A product term consisting of single-valued literals for each variable is called *minterm*.

Definition II.3: The truth vector representation d of an *mv* function is in a straight n -ary order.

Example II.1 illustrates the truth vector representation of an *mv* function.

Example II.1: The truth vector d for a two variable *mv* functions where variable X_1 is three-valued and variable X_2 is a four-valued one, is given by

$$\left[X_1^0 X_2^0, X_1^0 X_2^1, X_1^0 X_2^2, X_1^0 X_2^3, \dots, X_1^2 X_2^0, X_1^2 X_2^1, X_1^2 X_2^2, X_1^2 X_2^3 \right]$$

$n-1$

An n -variable *mv* function $f(X)$ can be represented by $N = \prod_{i=0}^{n-1} p_i$ values which

determine if the function is true, false, or not specified. It follows for Boolean functions that N is equal to 2^n . There exist two codings for the vector of truth values of a logic function. They are called *S* and *R* coding [65,81] where a <true, false, don't care> (ON, OFF, DC) minterm of a logic function is represented by the value <1, 0, 0.5> for the *R* coding and <-1, 1, 0> for the *S* coding.

II.2 DISCRETE ORTHOGONAL EXPANSIONS FOR MV FUNCTIONS

This dissertation discusses some aspects of the application of orthogonal and nonorthogonal transforms in logic synthesis. Therefore, this section introduces the classification of the general concepts for the applications of spectral methods in logic synthesis.

II.2.1 Application of Orthogonal Spectra

One application of orthogonal transforms in logic synthesis is to obtain a spectrum which gives a correlation of the mv function to a certain set of linear independent mv functions g_k . For this special application the general orthogonal expansion formula given by Equation (II.2) is restricted to the transform matrix G given by Equation (II.3) with mv functions g_k being linearly independent mv functions. Therefore, if the mv functions are described in R coding, the transform matrix G has 1, -1, and 0 entries only. Thus, g_k is real: $g_k = g_k^*$. The restriction of the transform matrix G having only 1, -1, and 0 as elements describes transform matrices like the ones considered in [89] which include the Walsh-type transforms, the Arithmetic and Adding transforms [74,90]. In logic synthesis such transforms can be applied to the decomposition of a mv function based on the correlation information obtained by the spectrum. One of the most important application is the linearization of Boolean functions [12,67]. In Chapter IV a special computation method for the transforms with singular and nonsingular transform matrices G having only entries 0, 1, and -1 is discussed.

II.2.2 Circuit Realization Based on Spectra

Another goal in logic synthesis is to express the mv function $f(X)$ by a set of completely specified mv functions g_k where the circuit realization of the set of mv functions g_k is smaller than the circuit realization of $f(X)$. Thus, the matrix G can have entries 1, -1,

0 only, $g_k(n) \in \{0, 1, -1\}$. Because the value of $X(k)$ determines if g_k is a function of the form or not, $X(k)$ is restricted to $X(k) \in \{0, 1\}$. Of special interest are circuit realizations of mv functions with OR (\cup) and AND (\cap) gates, Field (\cup, \cap), or EXOR (\oplus) and AND gates, Field (\oplus, \cap). A canonical expansion of an mv function is the expansion where any minterm of the mv function is obtained in a unique way. Thus, a minterm representation of an mv function is canonical. It follows directly, that for a two-level Sum of Product (SOP) realization of an mv function the only canonical form is the minterm representation. To show this formally we apply Equation (II.1) and Equation (II.2) to an mv function $f(x_0, \dots, x_{N-1}) = f(X)$ over the field $F(\cup, \cap)$:

$$X(k) = \bigcup_{n=0}^{N-1} f(n) \cap g_k(n) \quad (\text{II.8})$$

$$f(n) = \bigcup_{k=0}^{N-1} X(k) \cap g_k(n) \quad (\text{II.9})$$

where $X(k) \in \{0, 1\}$ is '1' for $g_k(n)$ being a function of the form. It follows from Equation (II.9) and the definition of the intersection, that $X(k) \in \{0, 1\}$. Therefore, the transform matrix $G = \{g_k(n)\}$ has to be the identity matrix I or matrix G obtained by any permutation of rows or columns from the identity matrix I . Thus, the minterm representation is the only two-level canonic form over the field $F(\cup, \cap)$. However, a transform does not have to be applied to each variable of the function. The simplest case to obtain a multi-level realization is to apply the expansion only to one variable x_i of the function

$$f(n) = (x_i \cap f(n)) \cup (\bar{x}_i \cap f(n)) \quad (\text{II.10})$$

This special case is the well known Shannon expansion [10]

$$f(x_0, \dots, x_n) = x_i f(x_0, \dots, 1, \dots, x_n) + \bar{x}_i f(x_0, \dots, 0, \dots, x_n) \quad (\text{II.11})$$

The Shannon expansion has been generalized in [91] to AND-OR type multiple-valued logic. In [24,84] the Shannon expansion has been generalized for Boolean rings.

Example II.2 illustrates a canonic multi-level form.

Example II.2: A unique three-level expansion for a four-variable function f is given by the linearly independent functions g_i and h_i illustrated in Figure 1.

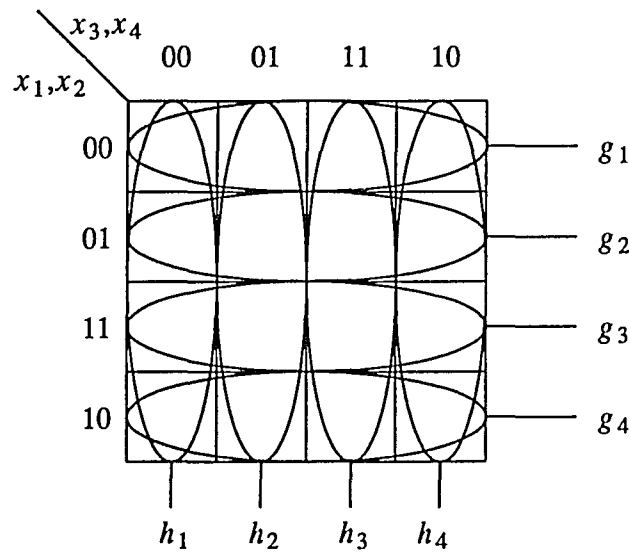


Figure 1. Unique expansion for a four-variable function.

As one can observe from Figure 1, each minterm of the Karnaugh map can be obtained by the intersection of a function g_i with a function h_j

$$f = \bigcup_{i=0}^3 \bigcup_{j=0}^3 f \cap g_i \cap h_j$$

II.2.3 Circuit Realization Based on GF 2 Transforms.

The third concept based on spectral methods applied in logic synthesis are transforms over GF 2 like the canonical Reed-Muller forms [23,24,29,92]. They have the property that they describe a two-level AND-EXOR circuit realization of the underlying

mv function. The canonical expansions over the GF 2 ($F(\oplus, \cap)$) are given by

$$X(k) = \bigoplus_{n=0}^{N-1} x(n) \cap g_k(n) \quad (\text{II.12})$$

It follows from the definition of \oplus ($1 \oplus 1 = 0, 1 \oplus 0 = 1$) that $X(k) \in \{0, 1\}$ for any function g_k . The inverse transform is given by

$$x(n) = \bigoplus_{k=0}^{N-1} X(k) \cap g_k(n) \quad (\text{II.13})$$

Equations (II.12) and (II.13) hold true for any orthogonal matrices over the GF 2 with $g_k(n) \in \{0, 1\}$, where $k, n = 0, \dots, N-1$.

As one can observe, there is only one canonic form for a two-level realization of a function in the Field (\cap, \cup) , while in the Field (\oplus, \cap) there are as many canonic forms for two-level realizations as there are orthogonal matrices $G = \{g_k(n)\}$. The following Theorem gives the special case $X(k) = f(X)$.

Theorem II.1: A multiple-valued input, binary output function f can be represented by Equation (II.14)

$$f = \bigoplus_{k=0}^{N-1} \left[g_k \cap f \right] \quad (\text{II.14})$$

where the $N-1$ functions g_k form the orthogonal transform matrix G with the property

$$\bigoplus_{k=0}^{N-1} g_k = 1. \quad (\text{II.15})$$

Proof:

$$f = \bigoplus_{k=0}^{N-1} \left[g_k \cap f \right] = f \cap \left[\bigoplus_{k=0}^{N-1} g_k \right] = f \cap 1 = f \quad \square$$

The expansion obtained by Equation (II.14) is a particular case of the orthonormal expansions presented in [93].

Let us observe, that a particular case of Equation (II.14) is described by the set G of mv functions with the property

$$\bigoplus_{k=0}^{N-1} g_k = \bigcup_{k=0}^{N-1} g_k = 1 . \quad (\text{II.16})$$

where the mv functions g_k are mutually disjoint ($g_i \cap g_j = \emptyset$). Thus, based on the property of set G according to Equation (II.16) the expansion given by Equation (II.14) can be expressed by

$$f = \bigcup_{k=0}^{N-1} \left[g_k \cap f \right] \quad (\text{II.17})$$

which is an unique AND-OR type Shannon expansion for an multi-level realization.

This Chapter introduced the three general concepts of spectral methods that can be applied to logic synthesis. As an exemplification of the first concept: decomposition of an function based on a spectrum, the linearization procedure [12,67] is generalized in Chapter VI to incompletely specified multi-output Boolean functions. The two other concepts are based on the Shannon expansion over the Field (\cap, \cup) and the Field (\oplus, \cap) . The Shannon expansion which is an important tool in multi-level logic synthesis, was generalized in this Chapter to multiple-valued input, binary output functions (mv functions for short). The problem to find a minimal expansion of an mv function according to Equation (II.13) and Theorem II.1 is very complex. Therefore, this dissertation investigates only two special cases. For the Field (\cup, \cap) a hierarchical multiplexer synthesis based on the Shannon expansion given by Equation (II.17) is presented in Chapter VII. For the application of the third concept: circuit realization based on GF 2 transforms, the mv functions g_i describing the orthogonal transform matrix G , Equation

(II.13) can be restricted to single variable *mv* functions. This restriction leads in Boolean domain to the canonic Kronecker Reed-Muller (KRM) forms [23,24,84] having the property given by Equation (II.15) and (II.16). The MIKRM form introduced in Chapter VIII and Chapter IX is an extension of the KRM form to multiple-valued input logic. It is not included in the expansion given by Equation (II.14). However it is properly included in the set of canonical expansions given by Equation (II.13).

CHAPTER III

DISJOINT REPRESENTATIONS OF LOGIC FUNCTIONS

Recently calculations of the Walsh, Adding, Arithmetic and Reed-Muller spectrum from a disjoint representation have been introduced [12,27,71,73,74,90,92,94-97]. In a disjoint representation of an *mv* function every minterm is covered only once. The main advantage of this method over the existing ones is the drastically reduced memory requirement. Another advantage is the possibility of the fast computation of spectra for functions which can be represented by relatively few disjoint terms. An algorithm and its implementation has been presented in [74,98,99] for the generation of a set of disjoint cubes for single output Boolean functions without respect to the minimality of the representation. As mentioned in the introduction, the calculation complexity of a spectrum increases linearly with the number of cubes in the disjoint representation of an *mv* function. Thus, the computation time for a spectrum can be reduced by starting from the minimal disjoint cube representation. Therefore, an algorithm is introduced to generate a quasi-minimal number of disjoint cubes for multiple-valued input, multi-output incompletely specified functions (*mv* function). Chapter III.1.1 introduces the general algorithm for *mv* functions. The special case of multi-output incompletely specified Boolean functions is illustrated in Chapter III.1.2.

The arithmetic cover for a Boolean function introduced in [12] is another disjoint representation, and will be described in Chapter III.2 for comparison purposes. A third disjoint representation are Ordered Decision Diagrams (ODD) [78,79,80]. The small memory requirement and fast computation of the intersection and the verification of the tautology of two *mv* functions makes them advantageous over the set of disjoint cube

representation. A modification to make them especially suitable for spectral applications is introduced in Chapter III.3.

III.1 DISJOINT CUBE REPRESENTATION

III.1.1 Disjoint Cube Representation for Mv Functions

Like the predecessor algorithms for single output Boolean functions [72,74] the algorithm introduced here starts from a minimal Sum of Product (SOP) form of the mv function. The algorithm presented here incorporates the basic concept of removing non-disjoint parts of the minimal SOP form of an mv function in such a way, that the number of resultant cubes is quasi minimal. For this purpose the disjoint sharp operation [100] is extended here to multi-output mv functions.

Definition III.1: Consider an n -variable input, k -output mv function $f : \{0, 1, -\}^n \rightarrow \{0, 1, -\}^k$. Denote each of the n literals as X_{n-1}, \dots, X_0 and each of the k outputs as $f_{k-1}, f_{k-2}, \dots, f_0$. Then the disjoint sharp operation $\#_j$ for multi-output mv functions is given by

$$C_a \#_j C_b = \bigcup_{i=0}^{n-1} X_0^{a_0} \cap b_0 \dots X_i^{a_i} \cap \bar{b}_i X_{i+1}^{a_{i+1}} \dots X_{n-1}^{a_{n-1}} f_0^{a_n} \cap b_n \dots f_{n+k-1}^{a_{n+k-1}} \cap b_{n+k-1} \\ + X_0^{a_0} \dots X_{n-1}^{a_{n-1}} f_0^{a_n} \cap \bar{b}_n \dots f_{n+k-1}^{a_{n+k-1}} \cap \bar{b}_{n+k-1} \quad (III.1)$$

where a_i and b_i denote the set of values of the i^{th} literal of the cubes C_a and C_b .

The first part of Equation (III.1) gives the result of the disjoint sharp operation [100] for the output functions that are true or not specified for both cubes C_a and C_b . The second part gives as result cube C_a for the output functions that are only true for C_a and not for C_b . Thus, the disjoint representation for two cubes C_a and C_b is given by the set of the cubes resulting from Equation (III.1) and C_b .

Example III.1: In this example all possible cases of output function combinations for two cubes are illustrated. The chosen set of incompletely specified Boolean functions, with $n=4$ and $k=5$, is given in Figure 2.

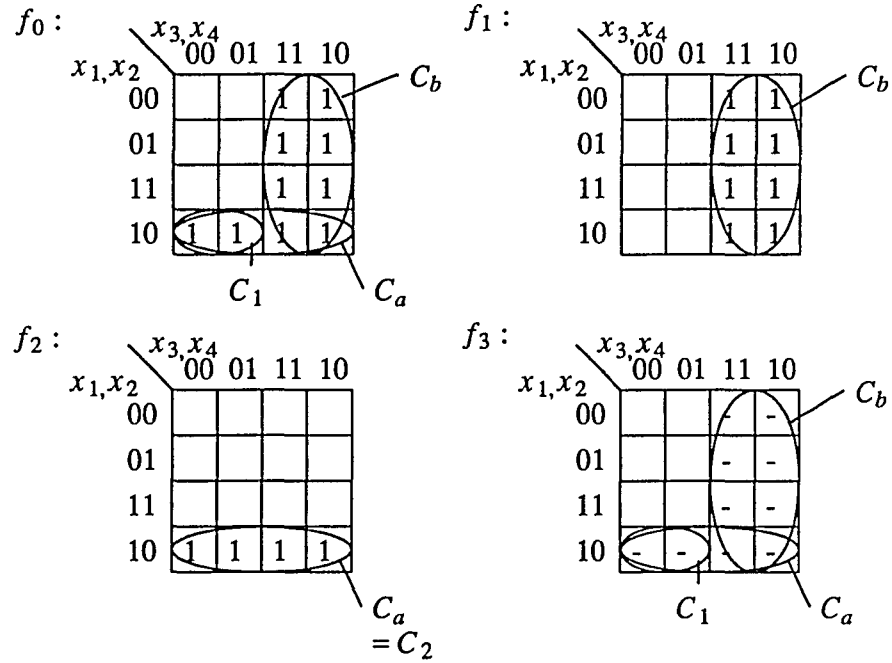


Figure 2. Multi-output function for disjoint sharp operation $C_a \# C_b$.

The function f_4 consists of false terms only. Therefore, it is not shown in Figure 2. As one can observe in Figure 2, the function f_0 and the function f_3 are not disjoint. Therefore, the disjoint sharp operator given in Equation (III.1) has to be applied. The first solution term C_1 is calculated according to the first part of Equation (III.1), and the second solution term C_2 is calculated according to the second part of Equation (III.1).

cube	x_1	x_2	x_3	x_4	f_0	f_1	f_2	f_3	f_4
C_a	1	0	-	-	1	0	1	-	0
C_b	-	-	1	-	1	1	0	-	0
C_1	1	0	0	-	1	0	0	-	0
C_2	1	0	-	-	0	0	1	0	0

With the disjoint sharp operation introduced in Definition III.1, it is possible to calculate the disjoint representation directly for a set of output functions rather than

performing the operation for each output function separately.

To obtain a quasi-minimal disjoint representation, the random ordering of cubes in the algorithm from [98,99] is replaced by a sorted list of cubes, where the cubes are sorted according to their size. The sorting can be performed in a short computation time by using a skip list [101], where tags point to the positions in the list where cubes with a different size start. Thus, the new algorithm compares the largest cube with all other cubes, starting from the smallest one. In the next step, the second largest cube is taken and compared to all smaller ones, etc. As the last step of the algorithm the cubes are merged, where possible, to obtain a smaller total number of disjoint cubes. In the example given in the next section it is shown that the number of disjoint cubes for the algorithm without sorting [72,74] has more disjoint cubes than the one presented here.

Notation:

<i>list</i>	double-linked list of cubes
<i>list->first</i>	pointer to first cube of the <i>list</i>
<i>list->last</i>	pointer to last cube of the <i>list</i>
<i>a, b, c</i>	pointers to cubes in the <i>list</i>
<i>a->cube</i>	cube of list, specified by pointer <i>a</i>
<i>a->next</i>	next cube in <i>list</i>
<i>b->previous</i>	previous cube in <i>list</i>
<i>sharp_list</i>	list of cubes resultant from sharp operation
$\#_j$	disjoint sharp operation, Equation (III.1)
absorb	checks if a cube is absorbed by another one
insert	inserts cube according to its size into <i>list</i> , sorting performed here
intersect	performs intersection among two cubes
merge	merges cubes with Hamming distance 1
remove	removes cube from <i>list</i>

III.1.2 An Example for Boolean Functions

As an illustration, the algorithm is applied to the incompletely specified Boolean function given in Example 2.1 [98], which is shown in Figure 3.a. The steps of the execution of the algorithm are illustrated in Figure 3. Because the function has only a single DC term, only the ON-terms have to be taken into account to obtain the disjoint representation. A '*' in the set of cubes indicates that the disjoint sharp operation ($\#_j$) has to be

Algorithm 1: Calculation of the Disjoint Representation for a Multi-Output Binary Function

```

a := list->first;
while ( a != list->last ){
    b := list->last;
    while ( b != a ){
        if ( ( a->cube intersect b->cube ) != empty ){
            // cubes do overlap
            if ( a->cube does not absorb b->cube ){
                // disjoint sharp
                sharp_list := a->cube # b->cube;
                insert sharp_list into list;
            }
            c := b;
            b := b->previous;
            remove c from list;
        }else{
            b := b->previous; } }
    a := a->next; }
merge cubes in list;

```

performed between those two cubes.

The set of cubes is sorted according to the size of the cubes. The algorithm starts comparing the largest cube with the smallest one. Thus, the first (cube[1]) and the last ON cube (cube[4]) have to be compared first. Because the two cubes overlap, the disjoint sharp operation (cube[4] # cube [1]) has to be applied. The result is shown in Figure 3.b. The former cube[4] has been removed and the result of the disjoint sharp operation (cube 1000) is inserted into the set according to its size. Because the size of the cubes resulting from a disjoint sharp operation is always smaller than the initial cubes, they are inserted after the initial cubes. Therefore, the next two cubes that are compared are cube[1] and cube[3] of Figure 3.b, and so on.

If the two chosen cubes in Figure 3.d would have been in a different order, the final result could not have been merged to a smaller set of cubes. To find always the optimal solution, a branching for each sharp operation would be necessary, but is not imple-

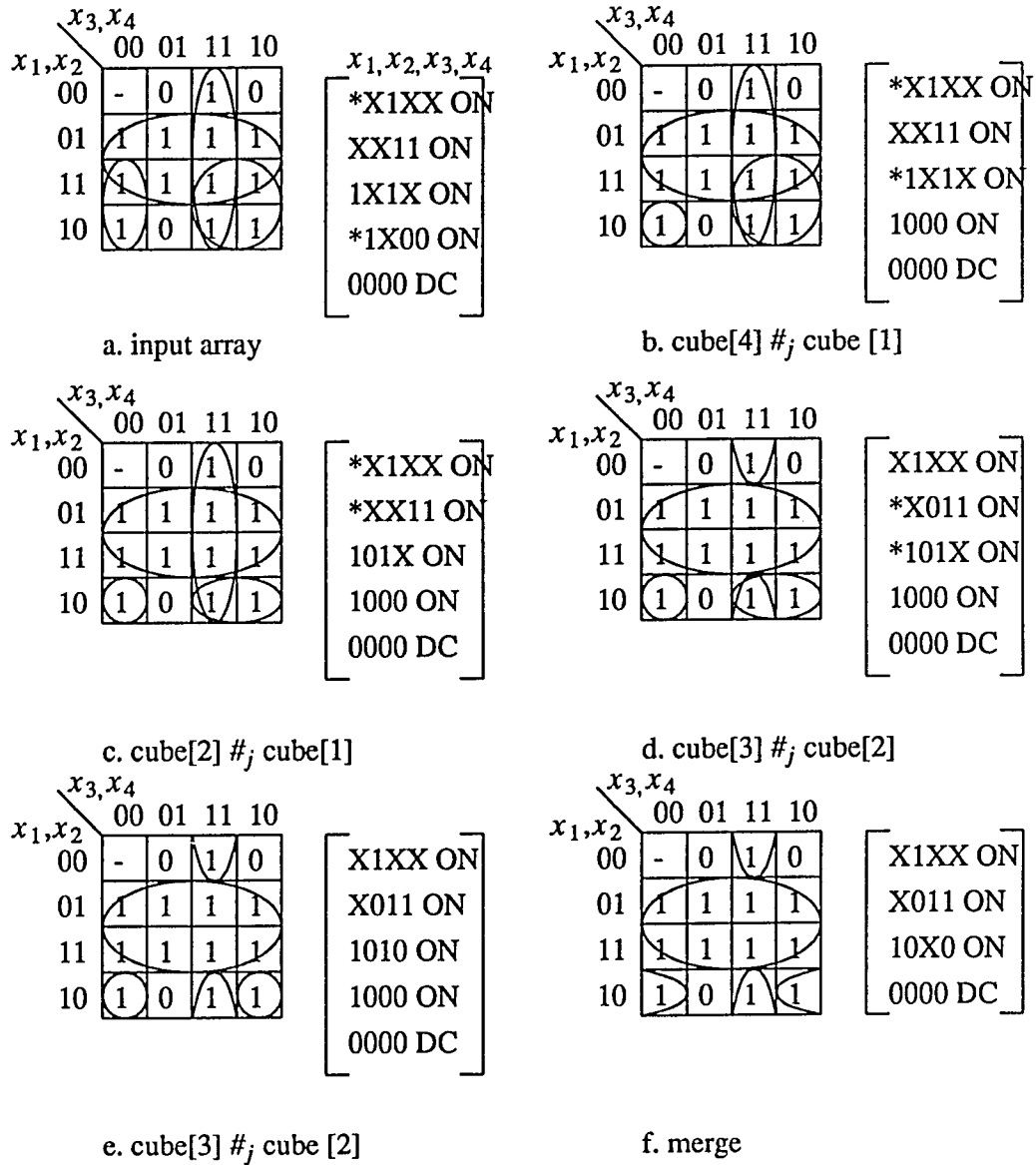


Figure 3. Stages of the execution of the algorithm to generate a disjoint cube representation.

mented in the algorithm presented here.

In Table I the new algorithm is compared to the *-Ddisjoint* option of ESPRESSO [1]. The functions shown in Table I have been taken from the MCNC benchmarks. They have been minimized by ESPRESSO before calculating their disjoint representation.

TABLE I
COMPARISON OF THE NUMBER OF DISJOINT CUBES FOR
MCNC BENCHMARK FUNCTIONS

	input var.	output var.	ESPRESSO	ESPRESSO -Ddisjoint	arithmetic cover	disjoint cover
b12	15	9	43	654		52
bw	5	28	22	26	25	26
con1	7	2	9	13	14	10
cps	24	109	163	895		219
duke2	22	29	86	181	158	103
e64	65	65	65	65		65
ex5	8	63	74	184		183
ex1010	10	10	284	1331		1081
inc	7	9	30	34		34
misex1	8	7	12	32	17	14
misex2	25	18	28	29	32	28
misex3	14	14	690	3789	*	1703
pdc	16	40	145	994		475
rd53	5	3	31	32	35	31
rd73	7	3	127	141	162	127
rd84	8	4	255	255	359	255
sao2	10	4	58	157	754	93
seq	41	35	336	1040		378
spla	16	46	260	1190		352
squar5	5	8	25	30		26
table3	14	14	175	249		182
table5	17	15	158	336		167
t481	16	1	481	2139		980
vg2	25	8	110	863	991	219
5xp1	7	10	65	99	129	71
9sym	9	1	86	189	274	145
sum			889	2017	2950	1122

(*) arithmetic cover was too large to be computed on a VAX 11/780 [12].

The second and third column of Table I give the number of input/output variables of the MCNC benchmark functions, listed in the first column. The fifth column lists the number of disjoint cubes obtained by ESPRESSO using the *-Ddisjoint* option. The sixth column gives the number of cubes for the arithmetic cover (see next section) computed in [12]. Finally, the last column shows the number of disjoint cubes obtained by our algorithm.

The last row gives the sum of cubes in the respective disjoint representation of all functions for which the arithmetic cover was available [12]. As one can observe, the results obtained by the disjoint algorithm are close to the minimal SOP form, where the number of disjoint cubes obtained by ESPRESSO *-Ddisjoint* is twice, and the arithmetic cover even three times as many.

III.2 ARITHMETIC COVER

In [12] the concept of the arithmetic cover was introduced to be able to compute the autocorrelation and Walsh spectrum of a completely specified Boolean function from a reduced representation rather than from minterms.

The arithmetic cover is described by two sets of cubes. The set of cubes in the initial nondisjoint SOP form and the set of cubes containing the intersections of the overlapping cubes of the first set.

Thus, the set of intersections of overlapping cubes describes the difference of the SOP form to a disjoint cube representation shown in Chapter III.1. Therefore, consisting of two sets of cubes, the arithmetic cover is usually larger than the disjoint representation introduced in the previous section.

The concept of the arithmetic cover is exemplified by Example III.2.

Example III.2: For the multi-output function illustrated in Example III.1 the arithmetic

cover is given by the SOP form

x_1	x_2	x_3	x_4	f_0	f_1	f_2	f_3	f_4
-	-	1	-	1	1	0	-	0
1	0	-	-	1	0	1	-	0

and the overlapping parts of the SOP form

x_1	x_2	x_3	x_4	f_0	f_1	f_2	f_3	f_4
1	0	1	-	1	0	0	-	0

III.3 ORDERED DECISION DIAGRAMS

Decision Diagrams are graph representations of mv functions. They have been proposed in [76,77,102]. The form of the Decision Diagrams where the order of input variables is fixed is called Ordered Decision Diagram (ODD). For Boolean functions the Ordered Binary Decision Diagram (OBDD) has been introduced in [78,103] and for multiple-valued functions the MDD in [80]. This unique form [78] has the property, that it is a disjoint representation of the underlying function. The advantage of the Decision Diagrams over the cube representation is their smaller memory requirement, especially for large functions. Additionally the verification of the identity of two functions as well as the intersection of two functions can be computed faster than with the cube representation. Thus, this representation is ideal for the application to spectral methods.

The joint of ODDs for different functions is called Shared Ordered Decision Diagram (SODD) [104].

For the method to calculate spectra from ODDs that will be introduced in Chapter IV, it is necessary to know the number of ON- and DC-minterms covered by the mv functions. To avoid the traversing of the complete ODD to obtain the number of covered minterms after each operation performed on the ODD, the *Node Weight* of a node in the ODD is introduced. The *Node Weight* allows the faster calculation of the value of a

spectral coefficient.

Definition III.2: The *Node Weight* ($NW(on_t, dc_t)$) of a node for completely specified *mv* functions is the number of true minterms (on_t) covered in the subsequent nodes. For incompletely specified *mv* functions the *Node Weight* ($NW(on_t, dc_t)$) of a node is defined by the pair of values: on_t for the number of true minterms covered in the subsequent nodes, and dc_t for the number of don't care minterms covered in the subsequent nodes.

The next section illustrates the calculation of the *Node Weight* for Boolean functions.

III.3.1 Ordered Binary Decision Diagrams

A Binary Decision Diagram OBDD [76-79] is a representation of a Boolean function as a directed graph. The advantage of such a representation is the smaller memory requirement for large functions in comparison to the conventional cube representation [100]. Algorithm 2 describes the procedure to obtain the *Node Weight*.

Algorithm 2: *Node Weight* for Incompletely Specified Boolean Functions

Step 1: $on_t = 1$, for each node being connected to the ON terminal.
 $dc_t = 1$, for each node being connected to the DC terminal.

Step 2: The node weight of the parent node X_p and the child nodes X_i and X_j is obtained by

$$on_t_p = on_t_i + on_t_j \quad (III.2)$$

$$dc_t_p = dc_t_i + dc_t_j \quad (III.3)$$

Step 3: Step 2 is repeated until the *Node Weight* for each node is obtained.

The method for the calculation is best illustrated on a small example.

Example III.3: The OBDDs of the function $f(X) = x_1x_3 + \bar{x}_1x_2$ and $g(X) = x_1 \oplus x_2$ with their *Node Weight* according to Definition III.2 are shown in Figure 4. Because the function $f(X)$ is completely specified only the $NW(on_t) = on_t$ for true minterms has to be calculated.

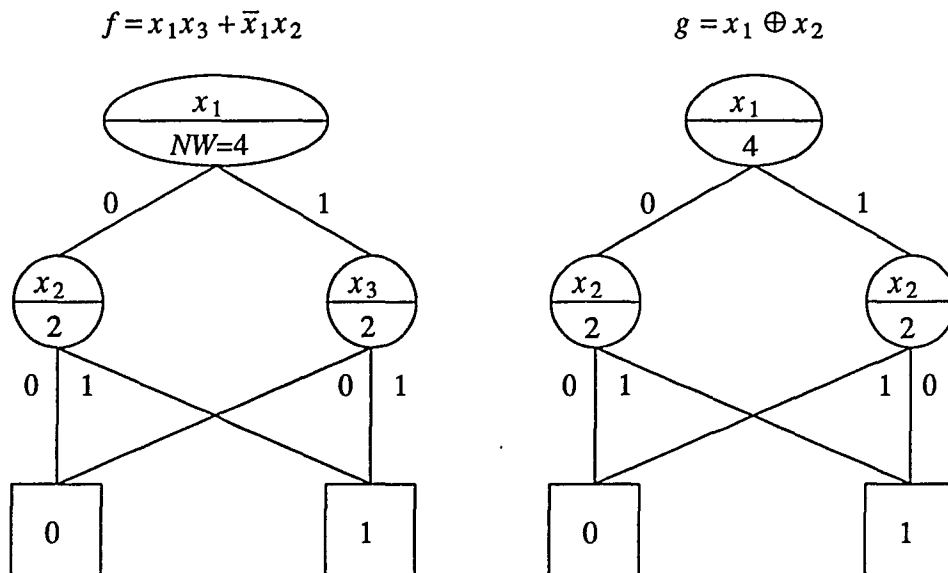


Figure 4. OBDDs for $f(X) = x_1x_3 + \bar{x}_1x_2$ and $g(X) = x_1 \oplus x_2$.

According to Step 1, the *Node Weight* for the nodes having connections to the 1-terminal is calculated first. Because for the functions in Figure 4 each of those nodes covers two minterms, the *Node Weight* for them is 2. The *Node Weight* in the parent nodes is calculated according to Step 2 by the summation of the *Node Weights* of the child nodes given in Equation (III.1).

The SOBDD for $f(X) = x_1x_3 + \bar{x}_1x_2$ and $g(X) = x_1 \oplus x_2$ shown in Figure 5, is obtained by taking identical branches or nodes of different functions only once while the *Node Weight* of the identical nodes that are combined have to be added together. Thus, the Node x_2 for functions f and g are combined to one node. The sum of their *Node Weight* is 4. Therefore, the node x_2 in Figure 5 has the *Node Weight* 4.

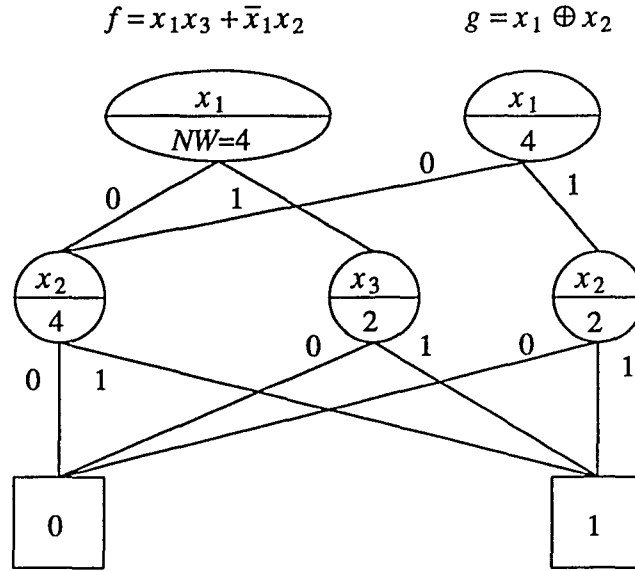


Figure 5. SOBDD for $f(X) = x_1x_3 + \bar{x}_1x_2$ and $g(X) = x_1 \oplus x_2$.

In the case of an incompletely specified Boolean function the *Node Weight* additionally contains the number of don't care minterms. Therefore, the number of don't care terms have to be calculated for each node too.

Example III.4: Figure 6 shows the OBDD of the function f with the completely specified part $x_1x_2\bar{x}_3 + \bar{x}_1x_2$ and the not specified part $x_1x_2x_3$ and its *Node Weight* $NW(on_t, dc_t)$. The number of covered ON- and DC- minterms is calculated analogously as in Example III.3. In each node the *Node Weight* is given by the pair of values on_t, dc_t . Thus, the root node gives the information that the function covers three ON-minterms and one DC-minterm.

III.3.2 Ordered Multiple-Valued Input Decision Diagrams

In this section the calculation of the *Node Weight* for *mv* functions is illustrated. The Ordered Decision Diagram for multiple-valued logic functions (OMDD) has been introduced in [80].

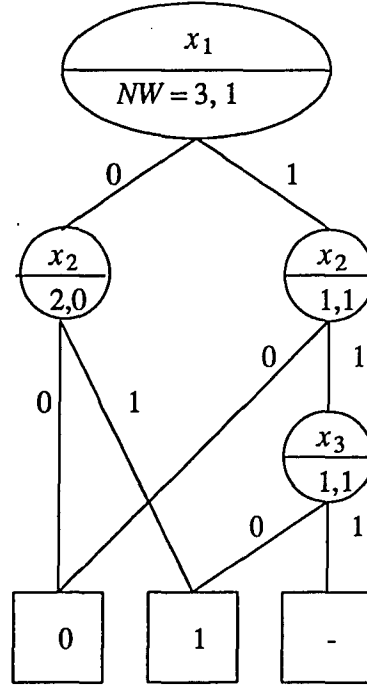


Figure 6. OBDD of an incompletely specified Boolean function.

In this dissertation only *mv* functions are investigated. Therefore, only the subset of two-valued output functions in their OMDD representation are considered. The algorithm for the computation of the *Node Weight* is described by Algorithm 3.

An execution of Algorithm 3 is illustrated in Example III.5.

Example III.5: Figure 7 shows the map and the OMDD of the function $f(X_1, X_2) = X_1^{023} X_2^{01}$, where variable X_1 is four-valued and variable X_2 is three-valued. The *Node Weight* is calculated according to the Algorithm 3. Only node X_2 is connected to the ON-terminal, thus *on_t* is calculated for X_2 . Because X_2 has two values being connected to the ON-terminal $on_t = 2$. According to Equation (III.4) the *Node Weight* of the parent node X_1 is obtained by the summation of the *Node Weight* for each value being connected to a child node. In our example three values of node X_1 are connected to node X_2 . Thus, the *Node Weight* of X_1 is $on_t = 3 \times 2 = 6$.

Algorithm 3: Calculation of the *Node Weight* for a Binary Function

Step 1: The *Node Weight* of a Node being connected to a ON-terminal or DC-terminal is given by
 $on_t = k$, where k is the number of values connected to the ON-terminal,
 and $dc_t = l$, where l is the number of values connected to the DC-terminal.

Step 2: The *Node Weight* of a parent node X_i can be calculated by

$$on_t_i = \sum_{x=0}^{p_i-1} on_t_x \quad (III.4)$$

where on_t_x is the number of ON-minterms covered by the child node connected to the node X_i by value x of node X_i .

$$dc_t_i = \sum_{x=0}^{p_i-1} dc_t_x \quad (III.5)$$

where dc_t_x is the number of DC-minterms covered by the child node connected to the node X_i by value x of node X_i .

Step 3: Step 2 is repeated until the *Node Weight* $NW(on_t_i, dc_t_i)$ for each node of the OMDD has been determined.

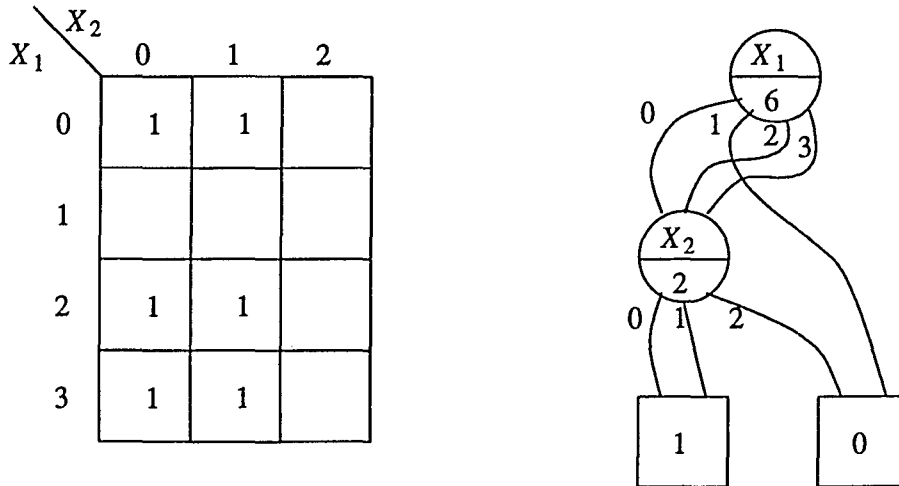


Figure 7. OMDD of $f(X_1, X_2)$.

In this Chapter two methods for the fast calculation of a quasi-minimal disjoint representation have been shown. One based on the cube representation of $m\nu$ functions and the other on Ordered Decision Diagram which are especially fast if only the intersection of two functions has to be computed. The two algorithms form the basis for the methods to compute spectra from disjoint representations that are introduced in the following chapters. The *Node Weight* introduced for the OBDDs allows the computation of the value of a spectral coefficient.

CHAPTER IV

SPECTRAL TECHNIQUES FOR *MV* FUNCTIONS

With the introduction of discrete orthogonal transforms to digital logic it was attempted to apply them to unify the logic synthesis [61-63,65,70]. As mentioned in the introduction, their shortcomings, i.e. the computational complexity, enormous memory requirements and difficult application to incompletely specified Boolean functions, did not allow to use these techniques in design automation. However, the combination of classical logic techniques with spectral techniques, the main contribution of this thesis, overcomes these shortcomings and makes it superior to strict classical and spectral methods. One approach is to compute a spectrum for *mv* functions directly from a disjoint representation like those introduced in the previous chapter.

In this chapter the concept of discrete orthogonal and nonorthogonal transforms will be reviewed and new calculation methods are introduced for *mv* functions based on the disjoint representation as cubes and Decision Diagrams. The difficult incorporation of incompletely specified *mv* functions into the synthesis with spectral methods is overcome by the introduction of the *T* spectrum.

IV.1 SPECTRA OF *MV* FUNCTIONS

The conversion of a Boolean function, given in a truth table representation *F*, by the multiplication of a nonsingular orthogonal transform matrix will be called spectrum [62,65]. For the sake of brevity the conversion of an *mv* function by singular or nonorthogonal transform matrices is also called spectrum. However, such a spectrum does not have to be unique. The vector of spectral coefficients *C* for an incompletely

specified n -variable mv function can be described according to Equation (II.4) by

$$C = [T] \times F \quad (IV.1)$$

where for Boolean functions T is an orthogonal $n \times k$ transform matrix [63,65,70,105], where k is an arbitrary number, F is the vector of truth values for the function $f(X)$, and C

is the vector of spectral coefficients. For mv functions T is an orthogonal $\sum_{i=0}^n p_i \times \sum_{i=0}^n p_i$

or a nonorthogonal $\sum_{i=0}^n p_i \times k$ matrix. Such a description of transform matrices allows

for the development of special transforms in logic synthesis. One particular application are transforms to compute spectra that give a correlation of the underlying mv function to limited set of mv functions, e.g. the realizable functions of a logic block in FPGAs. Chapter VII will present such a transform for the Actel ACT^{TM} macrocells.

The Walsh-type transforms have been investigated most in logic design [62,63,65,70,74]. Therefore, the calculation of the Rademacher-Walsh spectrum, which is one of the several Walsh-type transforms, is used to illustrate the computation of a spectrum by matrix calculation.

Example IV.1: The Rademacher-Walsh transform matrix and the spectrum of the three-variable function $f(X) = x_1 x_3 + \bar{x}_1 x_2$ is shown in Figure 8. The S coding is used for the vector of truth values F of the function $f(X)$.

The next example illustrates the possibility of defining a nonorthogonal transform for mv functions.

Example IV.2: The mv function $f = X^2 Y^{01} + X^0 Y^{34}$, Figure 9, consists of a three-valued literal X and a five-valued literal Y . Thus, the function can be described by 3×5 min-terms. The example of an nonorthogonal multiple-valued transform matrix for this function is given in Figure 10. The calculation of the transform by matrix multiplication is shown in Figure 10.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 4 \\ 4 \\ -4 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_{12} \\ s_{13} \\ s_{23} \\ s_{123} \end{matrix}$$

Figure 8. Spectrum by matrix calculation.

X \ Y		0	1	2	3	4
0					1	1
1						
2	1	1				

Figure 9. Map of function $f = X^2Y^{01} + X^0Y^{34}$.

The next section presents the meaning of the values of the spectral coefficients by their relation to *mv* functions. Additionally, orthogonal and nonorthogonal transforms having any matrix with $\{0, 1, -1\}$ as entries are introduced. This allows the definition of spectra based on a given set of *mv* logic functions rather than using one of the known discrete transforms.

$$\begin{bmatrix}
 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0
 \end{bmatrix}
 \times
 \begin{bmatrix}
 0 \\
 0 \\
 0 \\
 1 \\
 1 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 1 \\
 1 \\
 0 \\
 0 \\
 0
 \end{bmatrix}
 =
 \begin{bmatrix}
 4 & 1 \\
 2 & X^0 \\
 0 & X^1 \\
 2 & X^2 \\
 1 & Y^0 \\
 1 & Y^1 \\
 0 & Y^2 \\
 1 & Y^3 \\
 1 & Y^4 \\
 3 & X^0 \oplus Y^0 \\
 3 & X^0 \oplus Y^1 \\
 2 & X^0 \oplus Y^2 \\
 1 & X^0 \oplus Y^3 \\
 1 & X^0 \oplus Y^4 \\
 1 & X^1 \oplus Y^0 \\
 1 & X^1 \oplus Y^1 \\
 0 & X^1 \oplus Y^2 \\
 1 & X^1 \oplus Y^3 \\
 1 & X^1 \oplus Y^4 \\
 1 & X^2 \oplus Y^0 \\
 1 & X^2 \oplus Y^1 \\
 2 & X^2 \oplus Y^2 \\
 3 & X^2 \oplus Y^3 \\
 3 & X^2 \oplus Y^4
 \end{bmatrix}$$

Figure 10: Multiple-valued transform of function f .

IV.2 RELATIONS OF SPECTRAL TECHNIQUES TO CLASSICAL LOGIC

The relations of spectral coefficients to classical logic have been shown for the special case of Walsh-type [65,71,106], Adding and Arithmetic [107], and Reed-Muller [20,75] transforms. Each spectral coefficient is calculated by the multiplication of a row

vector of the transform matrix T and the function vector F , as in Equation (IV.1). This lead to the development of algorithms to compute the Walsh and Reed-Muller spectrum from an arithmetic cover [12,26], disjoint representations [73,75,95,94,97], and the computation of the Arithmetic and Adding spectrum from a minterm representation [90,108]. The methods introduced there are specific for the respective transforms. However, the underlying concept can be generalized to all possible orthogonal and nonorthogonal transforms having $\{-1, 1, 0\}$ as values in the transform matrices. To be able to introduce the generalization, first the concept of *standard trivial function* [71,81] will be extended.

Definition IV.1: The *Positive Standard Trivial Function (PSTF)* p_I is the *mv* function represented by the minterms defined by the $\{1\}$ entries of the corresponding row vector $T[I]$ of the transform matrix T .

The *Negative Standard Trivial Function (NSTF)* n_I is the *mv* function represented by the minterms defined by the $\{-1\}$ entries of the corresponding row vector $T[I]$ of the transform matrix T .

It should be observed that a *PSTF* or *NSTF*, being an *mv* function, can be represented by a set of disjoint cubes, ODDs, or minterms.

Example IV.3: The *PSTF* for the second row of the transform matrix shown in Figure 8 is given by $p_2 = x_1$ and the corresponding *NSTF* by $n_2 = \bar{x}_1$.

Definition IV.2 extends the notation and definitions for the calculation of the values of the spectral coefficients introduced in [71,81] for *PSTF* and *NSTF*.

Definition IV.2:

a_I number of true minterms of a *mv* function $f(X)$ covered by the *PSTF* p_I .

b_I number of false minterms of a *mv* function $f(X)$ covered by the *PSTF* p_I .

c_I number of true minterms of a *mv* function $f(X)$ covered by the *NSTF* n_I .

- d_I number of false minterms of a mv function $f(X)$ covered by the $NSTF$ n_I .
- e_I number of don't care minterms of a mv function $f(X)$ covered by the $PSTF$ p_I .
- f_I number of don't care minterms of a mv function $f(X)$ covered by the $NSTF$ n_I .
- where I is the row number of the corresponding transform matrix.

One distinguishes between two types of transforms [74,107].

Definition IV.3: A transform with a transform matrix having only $\{1, -1\}$ as elements is called a *global* transform.

Definition IV.4: A transform with a transform matrix having at least one $\{0\}$ as element is called a *local* transform.

The spectral coefficients of a *global* transform contain information about the whole mv function while the spectral coefficients of a *local* transform contain only information about parts of the mv function. Chapter VIII presents an application of the *local* transform to multiplexer synthesis.

The value of a spectral coefficient for the Walsh spectrum of an mv function in S coding is given by [74]:

$$s_I = (a_I - b_I) - (c_I - d_I) \quad (IV.2)$$

for the R coding by:

$$r_I = (a_I - c_I) + \frac{1}{2} (e_I - f_I) \quad (IV.3)$$

Other formulas derived from Equations (IV.2) and (IV.3) can be found in [73,74]. For a *global transform* the values b_I , d_I , and f_I can be obtained directly from the total number of true, or don't care minterms of the function, and parameters a_I , c_I , and e_I . In the case of a transform over the Galois Field (2) [24] the Equations (IV.2) and (IV.3) are not valid. There, the value of a spectral coefficient $C_I \in \{0, 1\}$ is determined by r_I , or s_I being even or odd.

Example IV.4: The eight *PSTFs* of the Rademacher-Walsh transform matrix in Example IV.1 are described by the circled areas in Figure 11. Because the Walsh-type transforms are *global* transforms it is not necessary to determine the *NSTFs* for them.

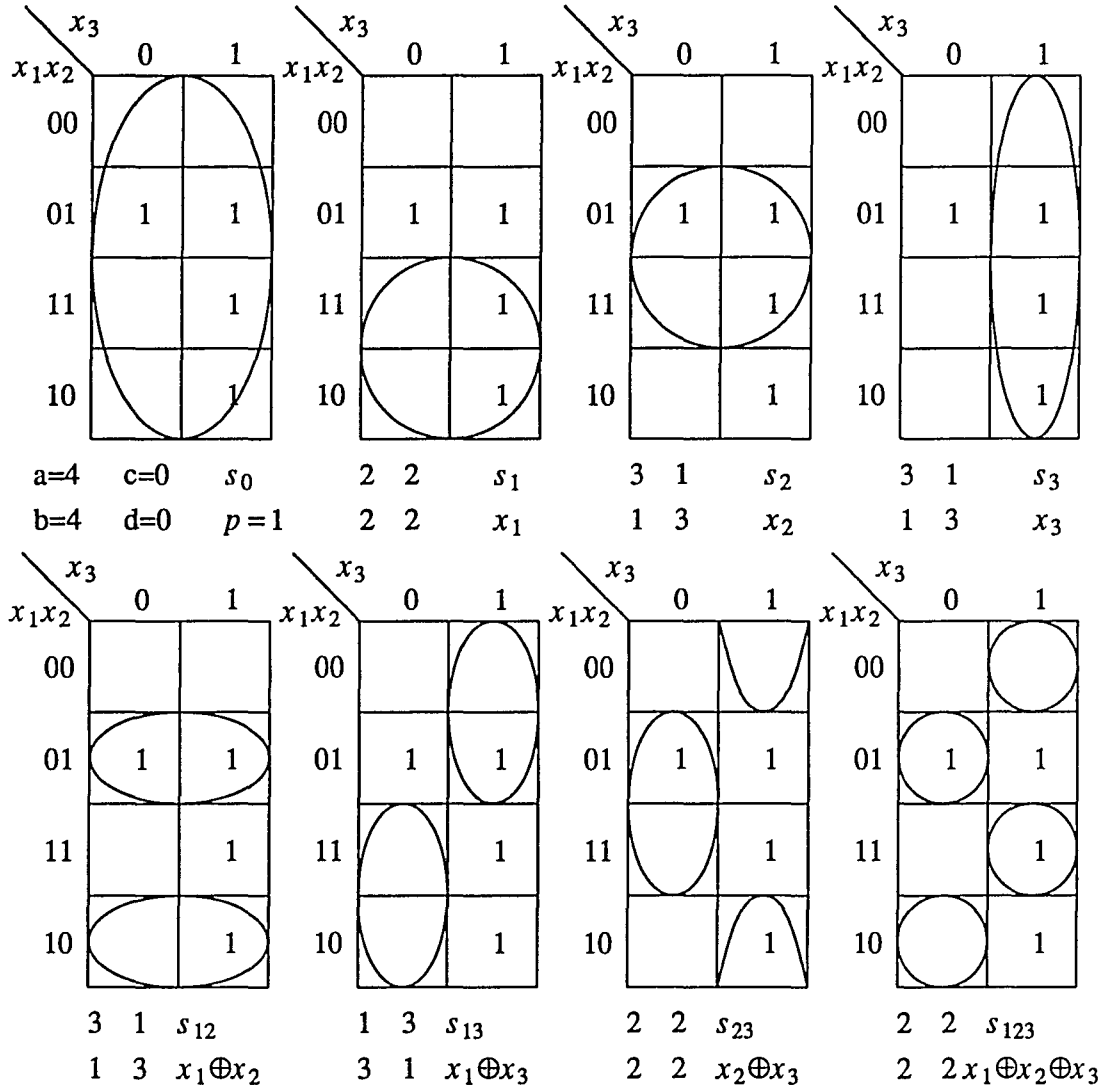


Figure 11. *PSTFs* for the Rademacher-Walsh transform.

The values of the spectral coefficients can be calculated according to Equation (IV.2). The values of a , b , c , and d are given in Figure 11. The same values are obtained as in Example IV.1 as one can easily verify. The values of the spectral coefficients can either

be calculated by the graphical method, counting the number of minterms inside or outside the *standard trivial functions*, or by the corresponding cube calculus method introduced in Chapter III. The intersection of the *PSTF* and the *mv* function in a disjoint representation has to be performed. Then the value of $a_I, b_I, c_I, d_I, e_I, f_I$ can be determined by the *Node Weight* of the intersection. Next the value of the spectral coefficients r_I , or s_I can be calculated according to Equation (IV.2) or (IV.3).

IV.3 THE T - SPECTRUM

Spectra in the *R*- or *S* coding for *mv* functions do not allow to determine the direct correlation of the function to a *PSTF* or *NSTF*. Especially in the case of incompletely specified *mv* functions the contribution of the not specified part of the function to the single spectral coefficient can not be determined directly because the information is spread over the complete spectrum. This section introduces the concept of the *T*-spectrum to overcome disadvantages of the *R* and *S* coding.

Definition IV.5: The spectral coefficient t_I of the *T*-spectrum for a *local* transform of a completely specified *mv* function is given by the pair of values $t_I \in \{a_I, b_I\}$; for an incompletely specified *mv* function by the quadruple of values $t_I \in \{a_I, b_I, e_I, f_I\}$.

The *R* or *S* spectrum for a *local* transform can be obtained from the *T*-spectrum by Equations (IV.2) or (IV.3).

Property IV.1: The spectral coefficient t_I of the *T*-spectrum for a *global* transform of a completely specified *mv* function is given by the value $t_I \in \{a_I\}$; for an incompletely specified *mv* function by the pair of values $t_I \in \{a_I, e_I\}$.

Property IV.2: For a *global* transform of a completely specified *mv* function Equation (IV.2) becomes

$$s_I = 2 \times (a_I - b_I) \quad (IV.4)$$

If we denote the number of minterms covered by the *PSTF* by pt we obtain

$$s_I = 4 \times a_I - 2 \times pt \quad (\text{IV.5})$$

Thus, Equation (IV.5) gives the relation of the *T*-spectrum for a *global* transform to the *S* coding.

Property IV.3: The *Node Weight* of an incompletely specified *mv* function is given by on_t and dc_t of the root node. Thus, Equation (IV.3) can be described by

$$r_I = 2 a_I + e_I - on_t - \frac{1}{2} dc_t \quad (\text{IV.6})$$

which can be calculated directly from the *T*-spectrum of an incompletely specified *mv* function.

It can be observed from Property IV.2 and IV.3, that the information obtained by the *S* or *R* coding of a spectrum can be easily obtained from the *T*-coding. However, the *T* spectrum gives additional information about the correlation of an *mv* function to a *PSTF* or *NSTF*.

Property IV.4: The set of values of a spectral coefficient t_I of the *T*-spectrum for a *global* transform are given by the values of the *Node Weight* for the intersection of the *mv* function with the *PSTF*. For a *local* transform the spectral coefficient t_I has additionally the values of the *Node Weight* for the intersection of the *mv* function with the *NSTF*.

Example IV.5: The *T*-spectrum is determined for the function in Example IV.1. Only a_I has to be calculated because the function in Example IV.1 is completely specified, and the Rademacher-Walsh transform is *global*. The value of a_I for each spectral coefficient is given in Figure 11. The value for the *S* coding of the spectrum given in Figure 8 can be obtained by Equation (IV.5), where $pt = 8$ for s_0 and $pt = 4$ for all other coefficients.

The introduced *T*-spectrum separates the information obtained for the completely

specified and the not specified part of the underlying mv function. Thus, the linearization algorithm introduced in Chapter VI, and the multiplexer synthesis algorithm introduced in Chapter VII, taking advantage of the T -spectrum, can directly determine the contribution of the not specified part of the mv function.

The next section presents the computation of a spectrum from the OBDD representation of the mv function.

IV.4 CALCULATION OF SPECTRA FROM ODDS

The method for the calculation of a spectral coefficient introduced in the previous section is based on the calculation of the intersection of the $PSTFs$ and the $NSTFs$ with the given mv function. Therefore, the intersection and the tautology of two functions has to be computed frequently. Because with an Ordered Decision Diagram (ODD) representation of the functions these operations can be performed relatively fast [79,80], ODDs are very well suited for this application. Moreover, in the case of orthogonal and nonorthogonal transforms where no Fast Transforms [70] exist they are more favorable than the general matrix multiplication method. Nonorthogonal transforms include also the case of the computation of selected spectral coefficients from an orthogonal transform.

Up to now only two algorithms have been introduced for spectral methods based on OBDDs [94,97]. The applied properties, which are similar to the ones used for the calculation from the disjoint cube representation [27,106] are specific to the Walsh-type and Reed-Muller transforms, while the approach shown here is general and can be applied to any orthogonal and nonorthogonal matrices having $\{0, 1, -1\}$ as entries.

Example IV.6: The OBDDs of the Boolean function $f(X)$ and the $PSTF$ $p = g(X)$ from Example III.3 are given in Figure 4. The $PSTF$ is the representation for the spectral coefficient s_{12} of a Rademacher-Walsh transform. Because the function $f(X)$ is

completely specified and the Rademacher-Walsh transform is a *global* transform, the spectral coefficients of the T -spectrum are given by the numerical values of the *Node Weight* $NW(on_t) = on_t$ of the OBDD obtained from the intersection of $f(X)$ and $g(X)$ (Property IV.1). The OBDD of the result of the intersection $f(X) \cap g(X)$ is shown in Figure 12.

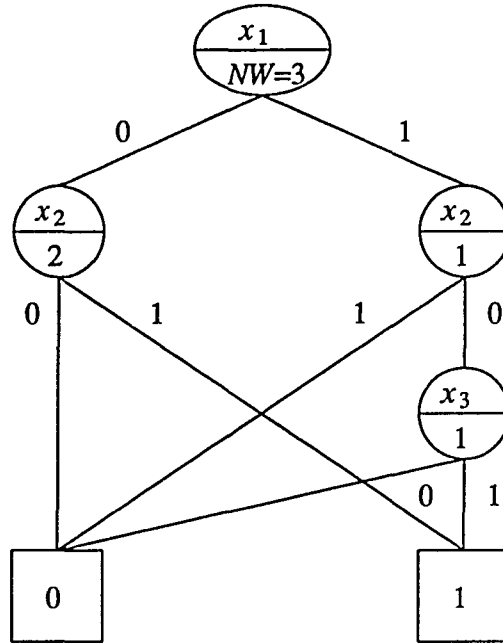


Figure 12. Result of intersection between $f(X)$ and $g(X)$: $f(X) \cap g(X)$.

As one can observe by comparison of Figure 4 and Figure 12, the left side of the OBDD $f(X)$ and the OBDD of the intersection are the same. Because a connection in the OBDD implementation is represented by a pointer, the result of the intersection and the function $f(X)$ have this part of the OBDD in common. Thus, the *Node Weight* does not have to be calculated on the left side of the OBDD. The spectral coefficient t_{12} for the T -spectrum is given by the *Node Weight* of the intersection: $t_{12} = a_{12} = 3$. The number of minterms covered by the *PSTF* is given by the *Node Weight* of the *PSTF*: $pt = 4$, Figure 4. Then the value of the spectral coefficient s_{12} is obtained by Equation (IV.5):

$s_{12} = 4 \times a_{12} - 2 \times pt = 4$, which has been also obtained in Example III.3.

The complete nonorthogonal or orthogonal spectrum can be calculated by performing the above shown operations for every *PSTF* describing the transform.

The *T*-spectrum or the *R* coding has to be used in the case of an incompletely specified *mv* function. Thus, the number of don't care minterms covered by a node in the ODD has to be calculated as well.

Example IV.7: The OBDD of the Boolean function with the completely specified part $f_c(X) = x_1x_2\bar{x}_3 + \bar{x}_1x_2$ and the not specified part $f_{in}(X) = x_1x_2x_3$ and its *Node Weight* $NW(on_t, dc_t)$ is shown in Figure 13.

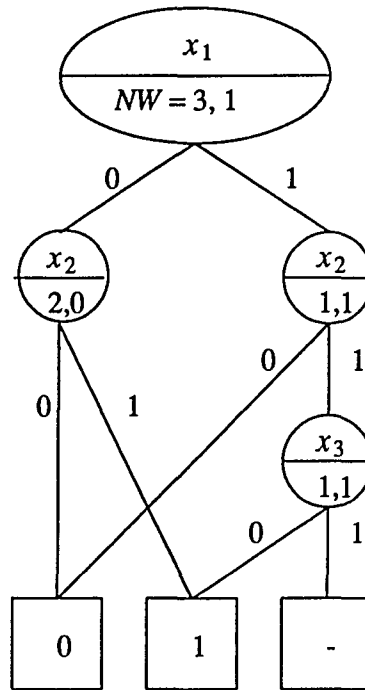


Figure 13. OBDD of an incompletely specified Boolean function.

The intersection of $f(X)$ and the *PSTF* $g(X) = x_1$ has to be performed (Figure 14) to calculate the spectral coefficient t_1 or r_1 . The intersection of the function $f(X)$ and the *PSTF* $g(X)$ is a complete subtree of the initial function $f(X)$. Thus, the *Node Weight*

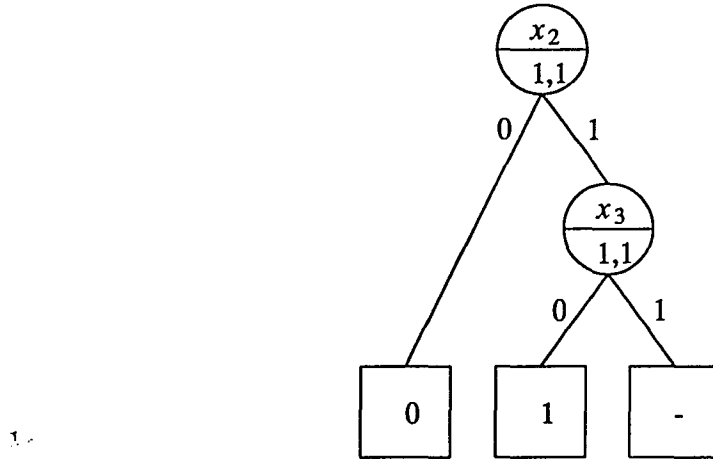


Figure 14. Intersection of the incompletely specified function $f(X)$ and $g(X)$.

does not have to be recalculated. By Property IV.4, the spectral coefficient t_1 is given by the *Node Weight* of $f(X) \cap g(X)$: $t_1 := (on_t=1, dc_t=1)$. The *Node Weight* of $f(X)$ has to be determined to calculate the spectral coefficient r_1 from the spectral coefficient t_1 by Equation (IV.6). The *Node Weight* of $f(X)$ is given in Figure 13 by $NW := (on_t=3, dc_t=1)$. Thus,

$$r_1 = 2 \times 1 + 1 - 3 - \frac{1}{2} \cdot 1 = -0.5$$

The Walsh-type transforms [65,74] have certain properties which allow the development of special algorithms for their computation from a disjoint representation. Because the fast calculation of the Walsh transform is crucial for the efficient linearization of a Boolean function, which is illustrated in Chapter VI, the next Chapter introduces a computation and memory efficient concept for the computation of the Walsh spectrum for incompletely specified Boolean functions that can be represented by few large disjoint cubes.

CHAPTER V

EFFICIENT ALGORITHM FOR THE CALCULATION OF WALSH-TYPE SPECTRA

The linearization method which will be introduced in Chapter VI is based on the Walsh transform. Thus, an algorithm which allows the fast and memory efficient calculation of the Walsh transform is crucial. Especially the memory requirement to store 2^n spectral coefficients for an n -variable Boolean function as well as its disjoint representation is a major hurdle to overcome.

The Walsh-type transforms, e.g. Rademacher-Walsh, Hademard-Walsh, Kaczmarz-Walsh and Walsh-Paley [65,70,74] exhibit a special property which allows the development of an algorithm for their fast computation from a disjoint representation [73,94]. Similarly to the general calculation method of transforms for mv logic presented in the previous chapter, the algorithms for the calculation of the Rademacher-Walsh spectrum for Boolean functions introduced in [73,94] are based on calculating the value for each spectral coefficient separately. However, the Walsh-type transforms allow a direct two-dimensional mapping of the function in Boolean domain to the Rademacher-Walsh spectrum. This method is especially fast and memory efficient for incompletely specified Boolean functions being represented by a few large disjoint cubes.

V.1 TWO-DIMENSIONAL MAPPING

To further decrease the computation complexity of the Walsh-type spectra, this section introduces a direct two-dimensional mapping method from the Boolean to the spectral domain. First let us observe some properties of the Walsh-type transforms.

Property V.1: The non-zero spectral coefficients of a Walsh-type spectrum for a Boolean cube exhibit the property that they describe a cube in the spectral map [65].

Definition V.1: The cube in the spectral map describing the non-zero spectral coefficients of a cube in the Boolean domain is called *scube*.

The concept of the two-dimensional spectral map introduced in [65] is illustrated with Example V.1.

Example V.1: The map of the spectrum for a 4-variable Boolean function is given in Figure 15.

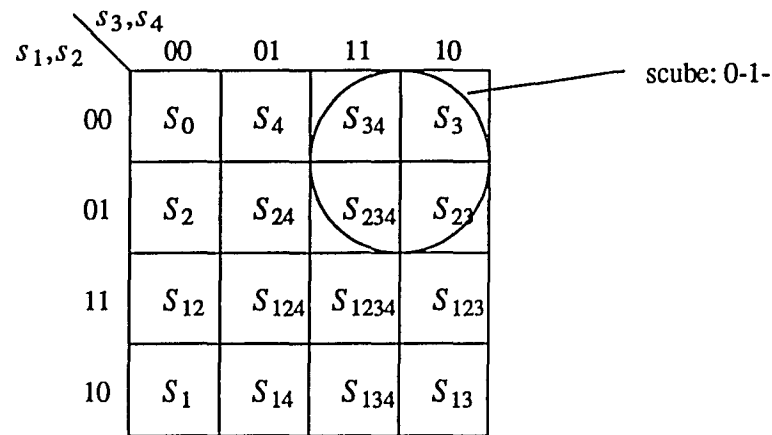


Figure 15. Spectral map for 4-variable Boolean function.

The following Properties give the relations of a cube in Boolean domain to a cube in the spectral domain.

Property V.2: The *scube* is obtained from a Boolean cube by the following literal substitution:

cube		scube
0	→	-
1	→	-
-	→	0

Example V.2: The *scube* describing the non-zero spectral coefficients in the spectral map of the cube 1-00 obtained according to Property V.2 is given by

$$\begin{array}{r} \text{cube} \quad 1-00 \\ \hline \text{scube} \quad -0-- \end{array}$$

Property V.3 gives the relation between the values of the spectral coefficients described by *scube* and the underlying cube in Boolean domain.

Property V.3: The magnitude of the value of the spectral coefficients for a cube c describing the completely specified part of a Boolean function is 2^{k+1} where k is the number of dc literals in the cube c . The value of the spectral coefficients for a cube c describing the not specified part of a Boolean function is given by 2^k .

The sign of the value of a spectral coefficient depends on the negative literals of the cube in Boolean domain and the order of the spectral coefficient [27]. The two conditions for which the value of a spectral coefficient has to be negated is described by the following Properties.

Property V.4: The values of the spectral coefficients for a Boolean cube c described by *scube* s have to be negated for

$$\bigoplus_{x_{c_i}=0} \overline{x_{c_i}} \quad (V.1)$$

where x_{c_i} is the i^{th} literal of cube c .

Property V.5 The value of the spectral coefficient has to be negated if the number of true literals in its minterm representation is even

$$\bigoplus_{i=0}^{n-1} \overline{x_i} \quad (V.2)$$

It follows from Equations (V.1) and (V.2) that the spectral coefficients having a negative value are given by the list of *scubes* s^-

$$s^- = \bigoplus_{x_{c_i} \neq 0} \overline{x_{c_i}} \quad (V.3)$$

and the spectral coefficients described by *scube* s having a positive value are given by the list of *scubes* s^+

$$s^+ = s \oplus s^- = s \cap \overline{s^-} = s \# s^- \quad (V.4)$$

The application of the Properties V.2-V.5 allows the calculation of the value and the cube description of the non-zero spectral coefficients for any cube of a Boolean function. Example V.3 illustrates the computation of the values of the spectral coefficients for the disjoint cubes of an incompletely specified Boolean function.

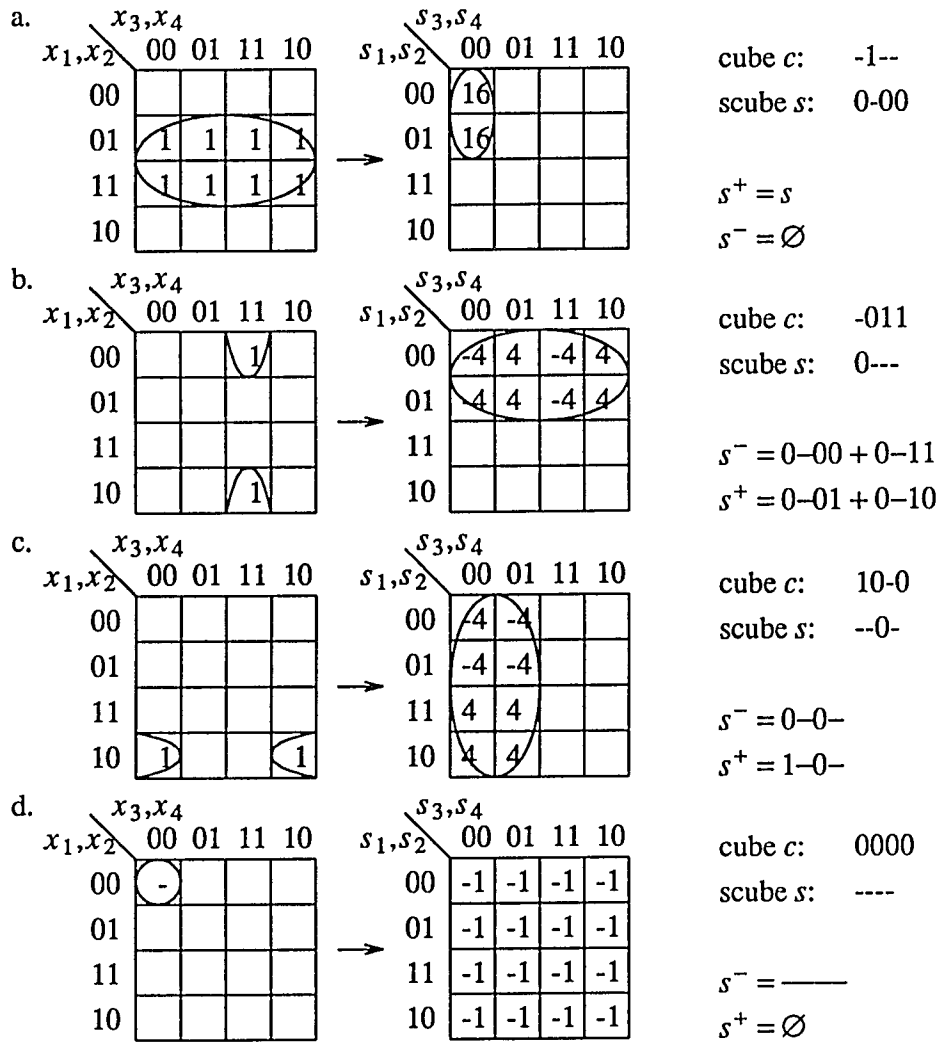
Example V.3: The disjoint representation of the completely specified part of the four-variable Boolean function $f(X)$ obtained in Chapter III.1.2 is given by:

$$f_s(X) = x_2 + \overline{x_2}x_3x_4 + x_1\overline{x_2}\overline{x_4}$$

and the not specified part by:

$$f_n(X) = \overline{x_1x_2x_3x_4}$$

Figure 16 shows the cubes and their *scubes* calculated according to Properties V.2-V.5. First, the *scube* s is determined for each cube c of the function $f(X)$. Then, the corresponding sets of *scubes* s^- and s^+ are calculated according to Property V.5. Next the values are negated for the spectral coefficients determined according to Property V.5. Thus, the values are obtained for the spectral coefficients given in the spectral maps of Figure 16. The spectrum for the complete function $f(X)$ is obtained by adding values of

Figure 16. *Scubes* for function $f(X)$.

the overlapping *scubes*. The result is given in Figure 17.

As one can observe from Figure 17, the complete spectrum can be described by a set of *scubes* having different values. A modified disjoint algorithm to the one described in Chapter III, can be applied to obtain the final spectrum, shown in Figure 17, from the partial spectra, shown in Figure 16. Let us first observe some properties for two overlapping *scubes*.

$s_1, s_2 \backslash s_3, s_4$					
		00	01	11	10
00	7	-1	-5	3	
01	7	-1	-5	3	
11	3	3	-1	-1	
10	3	3	-1	-1	

Figure 17. Final spectrum.

Property V.6: If two *scubes* s_1 and s_2 are identical ($s_1 = s_2$), the spectrum obtained by the summation of the respective values can be described by the list of *scubes* s_3 , s_4 and s_5 , where s_i^+ denotes a list of *scubes* describing spectral coefficients with positive value and s_j^- denotes a list of *scubes* describing spectral coefficients with negative value.

- the value of s_1 and s_2 is positive:

$$s_3^+ = s_1^+ \cap s_2^+ \quad (V.5)$$

- the value of s_1 is positive and the value of s_2 is negative:

$$s_4 = s_1^+ \cap s_2^- \quad (V.6)$$

the value of s_4 can be either positive or negative depending on the value of s_1 and s_2 .

- the value of s_1 is negative and the value of s_2 is positive:

$$s_5 = s_1^- \cap s_2^+ \quad (V.7)$$

the value of s_5 can be either positive or negative depending on the value of s_1 and s_2 .

- the value of s_1 and s_2 is negative:

$$s_3^- = s_1^- \cap s_2^- \quad (V.8)$$

The values of the new *scubes* are obtained by the summation of the respective values of s_1 and s_2 . The *scubes* s_1 and s_2 are replaced by the new *scubes* s_3 , s_4 and s_5 . If the obtained value for s_4 is zero, s_4 and s_5 can be removed.

Property V.7: For the case s_1 and s_2 overlap but $s_1 \neq s_2$, additionally to the *scubes* obtained by Property V.7 the *scubes* s_1 and s_2 have to be replaced by

$$s_1 = s_1 \# (s_1 \cap s_2) \quad (V.9)$$

$$s_2 = s_2 \# (s_1 \cap s_2) \quad (V.10)$$

To be able to use a modified version of the disjoint algorithm introduced in Chapter III, a disjoint sharp operation for *scubes* has to be defined.

Definition V.2: The disjoint sharp operation $\#_s$ for *scubes* given by

$$s_1 \#_s s_2 = s_1 + s_2 + s_3 + s_4 + s_5$$

where the coefficients s_i are obtained according to the Properties V.5-V.7.

The algorithm to obtain the disjoint spectrum representation is described by Algorithm 4. The notation follows the one for Algorithm 1.

Example V.4: In Example V.3 the spectra for the cubes of the disjoint incompletely specified Boolean function $f(X)$ have been computed. The initial nondisjoint spectrum given by the set of *scubes* has been obtained in Example V.3 by applying Properties V.2-V.5 to the disjoint cubes of the function $f(X)$. To obtain the spectrum description by disjoint *scubes* Algorithm 4 is applied to the above set of *scubes*. Because the first and the last *scube* (indicated by a "*") do overlap, the sharp operation introduced by Definition V.2 has to be applied. The result is given by

Notation:

$a \rightarrow \text{scube}$ *scube* of list, specified by pointer a
 $\#_s$ disjoint sharp operation according Definition V.2

Algorithm 4: Disjoint Spectrum Representation

```

a := list->first;
while ( a != list->last ){
    b := list->last;
    while ( b != a ){
        if ( ( a->scube intersect b->scube ) != empty ){
            // cubes do overlap
            sharp_list := a->scube #s b->scube;
            insert sharp_list into list;
            c := a;
            a := a->next;
            remove c from list;
            remove b from list;
            b := list->last;
        }else{
            b := b->previous; } }
    a := a->next; }
merge cubes in list;

```

$$\begin{bmatrix} 0 & - & 0 & 0 & 16 \\ 0 & - & - & - & \pm 4 \\ - & - & 0 & - & \pm 4 \\ - & - & - & - & -1 \end{bmatrix} *$$

$$\begin{bmatrix} 0 & - & 0 & 0 & 15 \\ 0 & - & 0 & 1 & -1 \\ 0 & - & 1 & - & -1 \\ 1 & - & - & - & -1 \\ 0 & - & - & - & \pm 4 \\ - & - & 0 & - & \pm 4 \end{bmatrix} *$$

Again the first and the last *scube* overlap. Thus, the disjoint sharp operation has to be applied again

$$\begin{bmatrix} 0 & - & 0 & 0 & 11 \\ 0 & - & 0 & 1 & 5 \\ 1 & - & 0 & - & 4 \\ 0 & - & 1 & - & -1 \\ 1 & - & - & - & -1 \\ 0 & - & - & - & \pm 4 \end{bmatrix} * \begin{bmatrix} 0 & - & 0 & 0 & 7 \\ 0 & - & 0 & 1 & -1 \\ 0 & - & 1 & 1 & -4 \\ 0 & - & 1 & 0 & 4 \\ 1 & - & 0 & - & 4 \\ 0 & - & 1 & - & -1 \\ 1 & - & - & - & -1 \end{bmatrix} * \begin{bmatrix} 0 & - & 0 & 0 & 7 \\ 0 & - & 0 & 1 & -1 \\ 0 & - & 1 & 1 & -4 \\ 0 & - & 1 & 0 & 4 \\ 0 & - & 1 & - & -1 \\ 1 & - & 1 & - & -1 \end{bmatrix} *$$

The disjoint sharp operation is applied until finally the following disjoint description of *scubes* for the spectrum is obtained

$$\begin{bmatrix} 0 & - & 0 & 0 & 7 \\ 0 & - & 0 & 1 & -1 \\ 0 & - & 1 & 1 & -5 \\ 0 & - & 1 & 0 & 3 \\ 1 & - & 1 & - & -1 \end{bmatrix}$$

The next section gives the results obtained for the introduced algorithm for some benchmark examples.

V.2 BENCHMARK RESULTS

As stated in the previous section, the introduced algorithm works efficiently only for functions represented by few large cubes. Therefore, Table II gives the computation time (in seconds) on a Sparc 4/370 for the calculation of the Walsh spectrum for certain subset of the largest cubes in the disjoint representation (obtained by the algorithm presented in Chapter III) of the respective functions. The column *10 cubes* gives the computation time for the Walsh spectrum for the 10 largest cubes in the disjoint representation of the respective function. Similarly for the columns *20*, *30*, and *40 cubes*.

It should be observed, that the partial spectra for the Boolean functions *vg2*, *seq*, *e64*, and *cps* could be computed in a resonable amount of time, while their calculation

TABLE II
COMPUTATION TIMES FOR WALSH SPECTRUM

	input var.	output var.	disjoint cover	10 cubes	20 cubes	30 cubes	40 cubes
b12	15	9	52	1.8	12.4	51.1	127.8
bw	5	28	22	0.7	1.5		
clip	9	5	163	1.9	6.9	26.8	54.9
con1	7	2	10	1.4			
cps	24	109	219	5.6	22.0	49.0	113.8
duke2	22	29	103	6.7	37.0	81.2	131.8
e64	65	65	65	34.8	80.9	150.8	749.2
ex5	8	63	183	3.0	13.0	34.5	70.2
f51m	8	8	78	0.4	2.1	7.6	18.7
inc	7	9	34	1.4	4.2	9.5	
misex1	8	7	14	0.9	1.0		
misex3c	14	14	518	1.3	7.0	18.1	37.9
pdc	16	40	475	3.5	15.8	44.5	119.2
rd53	5	3	31	0.7	1.1	2.2	
rd73	7	3	127	1.5	11.7	17.5	23.7
sao2	10	4	93	0.9	11.0	31.1	60.9
seq	41	35	378	16.1	81.8	203.9	404.9
spla	16	46	352	4.0	14.8	43.2	68.4
squar5	5	26	26	0.5	0.9	1.8	
vg2	25	8	219	4.4	18.6	39.4	251.1
5xp1	7	10	71	0.7	3.1	7.5	13.4

would be impossible with all other methods.

The linearization method presented in the next Chapter takes advantage of the introduced here algorithm to compute the Walsh spectrum.

CHAPTER VI

LINEARIZATION OF INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS

Synthesis tools for multi-level logic synthesis [4-8] are based on the unate paradigm [9,10]. Thus, they do not include the synthesis for the efficient incorporation of EXOR gates. However, many real life circuits like adders, counters, multipliers have a much smaller circuit realization if EXOR gates are incorporated in the synthesis process [11]. Therefore, there is a need for synthesis tools which include the EXOR gate in the synthesis process.

Only the SPECSYS [12] and the GATEMAP [44] system allow for the synthesis of multi-level circuits including the EXOR gates in the synthesis procedure. While the GATEMAP system uses Reed-Muller forms, the SPECSYS system makes use of the linearization procedure introduced in [62]. The linearization of a Boolean function is based on extracting an EXOR preprocessor of the Boolean function in such a way, that the following function is nearly unate. The linearization procedure used in the SPECSYS system is based on the autocorrelation spectrum. Another approach has been presented in [65,67,109] for the linearization of Boolean functions using the Rademacher-Walsh spectrum. It is based on the complexity criterion $\tilde{\eta}(f)$ which exhibits advantageous properties in the Rademacher-Walsh domain. While the linearization method described in [65,109] is based on the spectral translation method, the method introduced in [67] has the same basic procedure as the approach making use of the autocorrelation spectrum [12,62].

In this Chapter the linearization procedures based on the autocorrelation spectrum, the Rademacher-Walsh spectrum, and the spectral translation are reviewed. A

generalization of the linearization procedure based on the Rademacher-Walsh spectrum to systems of incompletely specified Boolean functions is introduced. The new linearization procedure takes advantage of the fast algorithms for the computation of the Walsh spectrum introduced in Chapter IV and Chapter V. Therefore, the disadvantage of the high memory requirements of the existing methods being calculated from minterms is overcome.

VI.1 COMPLEXITY OF BOOLEAN FUNCTIONS

For the synthesis with EXOR gates a method which allows to determine if a function is nearly unate or nearly linear is necessary. It was conjectured in [110] that the effort to find the exact complexity of a Boolean function is about of the same order as to construct the minimal network. Thus, the so called complexity criteria [5,62,111,112] have been developed. They provide an upper bound for the complexity and induce a classification of functions according to the complexity criterion. One criterion which is especially suited to distinguish between nearly linear and nearly unate functions is the complexity criterion $\tilde{\eta}(f)$ [62].

$$\tilde{\eta}(f) = \sum_{||\tau||=1} \sum_{x=0}^{x=2^n-1} (f(x)f(x \oplus \tau) + \overline{f(x)} \overline{f(x \oplus \tau)}) \quad (VI.1)$$

where $||\tau||$ is the number of units in the binary expansion of τ .

The complexity criterion $\tilde{\eta}(f)$ is based on the summation of the number of minterms surrounding each minterm that take the same function value $\{0, 1\}$. Thus, the value of the complexity criterion $\tilde{\eta}(f)$ increases as the ON and OFF minterms of the Boolean function are more grouped. Hence, it is a measure of the complexity for an AND-OR realization of the Boolean function.

Example VI.1: The complexity of the linear and the nearly unate function given in Figure

18 is computed according to Equation (VI.1). For each minterm the number of surrounding minterms that have the same value is computed. Then, $\tilde{\eta}(f)$ is calculated by the summation of the obtained values.

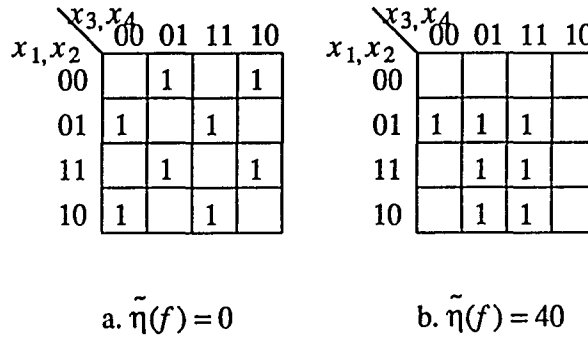


Figure 18. Complexity of a linear and a nearly unate function.

The complexity criterion $\tilde{\eta}(f)$ can be formally used to introduce a Definition for nearly unate and nearly linear functions. Let us first observe some properties of $\tilde{\eta}(f)$.

Property VI.1: For a Boolean function which is linear in all variables $\tilde{\eta}(f) = 0$.

Property VI.2: For a Boolean function of n variables which is unity $\tilde{\eta}(f) = n \times 2^n$.

Definition VI.1 introduces a possible criterion for the definition of nearly unate and nearly linear.

Definition VI.1: A Boolean function of n -variables is called nearly linear if $\tilde{\eta}(f) < n \times 2^{n-1}$ and nearly unate if $\tilde{\eta}(f) \geq n \times 2^{n-1}$.

Definition VI.1 can be used as a heuristic to determine, when a linearization procedure should be performed on a Boolean function. However, this heuristic does not take into account if only subsets of the function are nearly linear or nearly unate.

In [111] the complexity criterion $\tilde{\eta}(f)$ has been transformed into the Rademacher-Walsh spectrum domain.

$$\tilde{\eta}(f) = n2^n - \frac{1}{2^{n-2}} \sum_{\tau=0}^{2^n-1} ||\tau|| r_{\tau}^2 = n2^n - \frac{1}{2^n} \sum_{\tau=0}^{2^n-1} ||\tau|| s_{\tau}^2 \quad (\text{VI.2})$$

where $|\tau|$ is the order of the spectral coefficient. From this equation one can conclude that a function with large-magnitude spectral coefficients in low-order positions is simpler to realize with an AND-OR circuit than one whose large-magnitude coefficients are in high-order coefficients. This property is used by [65,67] to calculate an EXOR preprocessor, which is illustrated in Chapter VI.2.3.

The complexity of a system of Boolean functions $F(X)$ is given by the sum of the complexities of the single functions [62].

$$\tilde{\eta}(F) = \sum_{i=0}^r \tilde{\eta}(f_i) \quad (\text{VI.3})$$

where r is the number of functions f_i in the system of Boolean functions $F(X)$. The property given by Equation (VI.3) is applied in the here introduced linearization procedure for systems of incompletely specified Boolean functions.

VI.2 LINEAR PREPROCESSOR FOR SYSTEMS OF COMPLETELY SPECIFIED BOOLEAN FUNCTIONS

The linearization of a system of Boolean functions $F(X)$ for a given complexity criterion α can be formulated as follows [12,62,113]: "Find a logic block σ , consisting of EXOR gates, that will minimize the complexity measure α of the block F_{σ} to be implemented over a complete basis."

$$F_{\alpha}(\sigma \oplus X) = F(X) \quad (\text{VI.4})$$

where σ is a nonsingular (mod2) $n \times n$ matrix. The block circuit for a system $F(X)$ of k Boolean functions with n inputs obtained by Equation (VI.4) is shown in Figure 19. The logic block σ describing the EXOR preprocessor is determined in such a way, that the following core function F_{σ} has a minimal realization according to the chosen complexity

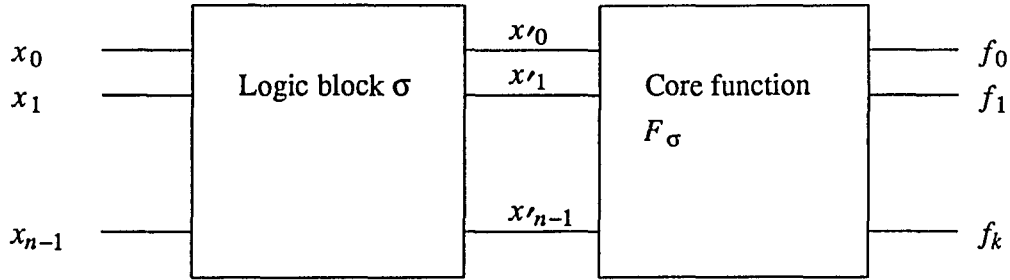


Figure 19. EXOR predecessor.

measure α .

This approach has been used by [12,62,67] while the method introduced by [65,109] is based on the spectral translation method for completely specified Boolean functions.

As shown in the previous section, the complexity criterion $\tilde{\eta}(f)$ is very suitable for the linearization of Boolean functions. Thus, all linearization procedures have been based on this criterion. The basic underlying algorithm to obtain the transform matrix σ given in [12,62,67] is illustrated in the following section. In Chapter VI.2.2 the linearization based on the spectral translation method [65,109] is reviewed.

VI.2.1 General Decomposition Algorithm

Based on the complexity criterion $\tilde{\eta}(f)$ two linearization methods to obtain σ for completely specified Boolean functions have been developed, one based on the Rademacher-Walsh spectrum [65,67] while the linearization algorithm proposed by [62] and developed by [12] is based on the autocorrelation spectrum $B(\tau)$

$$B(\tau) = \sum_{x=0}^{2^n-1} f_i(x) f_i(x \oplus \tau) \quad (\text{VI.5})$$

where τ is the minterm obtained by the binary representation of the integer number $0 \leq \tau \leq 2^n - 1$.

Both methods have the same basic linearization procedure to determine a minimal logic block σ which is given by Algorithm 5.

Algorithm 5: Linearization Procedure

- Step 1: Calculation of the appropriate spectrum.
- Step 2: Selection of a spectral coefficient (not the dc coefficient).
 - (i) select the coefficient(s) with the largest magnitude
 - (ii) if more then one coefficient satisfies (i) select the coefficient(s) of the lowest order.
 - (iii) from the remaining coefficients select the one with the largest decimal index.
- Step 3: Insertion of the binary representation of the index of the chosen spectral coefficient as a new column in the transform matrix σ .
- Step 4: Removal of all spectral coefficients having an index being linear dependent on the column vectors of matrix σ . (to obtain a nonsingular (mod2) $n \times n$ matrix σ)
- Step 5: Repeat Step 2 - Step 4 n times.

To obtain an EXOR preprocessor with minimal fan in and therefore with minimal complexity, the spectral coefficients are selected in such a way, that the transform matrix σ has the least possible number of 1s, Step 2 of Algorithm 5. One special property of such a selection is that the obtained realization of the Boolean function is easily testable [67]. It was observed in [12] that the selection of spectral coefficients to minimize the number of 1s in the transform matrix σ decreases the circuit size of the EXOR preprocessor about 50-75 percent, while it was found that the complexity of the remaining core function $f_{\sigma}(X)$ increases by such an selection by only less than 10 percent.

The main disadvantage of the both methods is the computation of their respective spectra from a minterm representation [67] or from the arithmetic cover [12]. The dis-

joint cube representation obtained by the method described in Chapter III was shown to have on the average less than half the number of terms than the arithmetic cover. Thus, because the computation time of the spectrum increases linearly with the number of terms in the disjoint representation, the computation time necessary to compute the transform matrix σ can be reduced by 50 percent by using the disjoint cube representation instead of the arithmetic cover. Therefore, our linearization procedure introduced in Chapter VI.3 takes advantage of the computation method introduced in Chapter III.

VI.2.2 Linearization by spectral translation

In [65,109] five invariance operations on spectral coefficients have been introduced. The so called spectral translation operations are based on interchanging and negation of spectral coefficients in the spectrum of a completely specified Boolean function.

One of these operators introduces EXORs at the input of a completely specified Boolean function. It is based on the replacement of any input variable x_i by any EXOR function $x_i \oplus x_j$, $i \neq j$ that is a second order *positive standard trivial function* of the Rademacher-Walsh spectrum. It can be described by the interchange of 2^{n-2} pairs of spectral coefficients:

$$\begin{aligned} s_i &\longleftrightarrow s_{ij} \\ s_{ik} &\longleftrightarrow s_{ijk} \\ &\dots \end{aligned}$$

The interchange can be realized also by performing the substitution of the variable x_i by the EXOR function $x_i \oplus x_j$ with following computation of the Rademacher-Walsh spectrum for the resulting function.

As stated in Chapter VI.1, the complexity criterion $\tilde{\eta}(f)$ described in the Rademacher-Walsh domain (Equation (VI.2)), gives the information that a completely specified Boolean function having a spectrum with high values in low order coefficients

and low values in high order coefficients has a high complexity. Therefore, such a function has a small AND-OR circuit realization.

Thus, the method making use of the spectral translation interchanges high value coefficients in high orders with low value coefficients in low orders [65,109].

As stated in [65] the problem of decreasing the complexity to the possible minimum by this method leads to a simple realization with majority gates, however this is not necessarily attractive for AND-OR realization. This problem is overcome by the method developed by [67] which was described in the previous section.

The linearization procedure given by Algorithm 5 and the Rademacher-Walsh spectrum is extended in Chapter VI.3 to systems of incompletely specified Boolean functions. The introduced algorithm takes advantage of the calculation of the Rademacher-Walsh spectrum from the quasi-minimal disjoint representation of the incompletely specified, multi-output Boolean function. Chapter VI.4 introduces a cube calculus method to overcome the complexity of the computation of the inverse Rademacher-Walsh transform to obtain the nearlyunate core function.

VI.3 LINEAR PREPROCESSOR FOR SYSTEMS OF INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS

The techniques to determine an EXOR preprocessor reviewed in the previous section are extended in this section to incompletely specified multi-output Boolean functions. Let us recall the meaning of a high magnitude of a spectral coefficient of the Rademacher-Walsh spectrum. A high positive value indicates that the underlying Boolean function is highly correlated to the *positive standard trivial function* of the corresponding spectral coefficient. A high negative value indicates that the underlying Boolean function is highly correlated to the negation of the *positive standard trivial function*, which is the *negative standard trivial function*. Thus, to efficiently incorporate the

not specified part of an incompletely specified Boolean function into the linearization procedure, the not specified part of the function has to be assigned to a specified part in such a way, that the maximal magnitude of a spectral coefficient is obtained. This leads to maximal values in low order coefficients which relate to a nearly unate function and maximal values in high order coefficients which relate to nearly linear functions.

Because this task is very cumbersome to solve with the Rademacher-Walsh spectrum for R- or S-coding, the method introduced here takes advantage of the presented T -spectrum. This allows to solve the problem of assigning the not specified part to obtain a maximal magnitude easily.

The T -spectrum of an incompletely specified Boolean function is given by the values a_I, b_I, e_I, f_I (Definition IV.2). According to Equation (IV.2) the problem is to assign e_I and f_I to false or true terms in such a way that s_I becomes maximal.

There are four possible assignments for each spectral coefficient of the not specified part to obtain the maximal value of s_I .

- (1) all don't care terms in the *positive standard trivial function* are assigned to false terms and all don't care terms in the *negative standard trivial function* are assigned to true terms.
- (2) all don't care terms in the *positive standard trivial function* are assigned to true terms and all don't care terms in the *negative standard trivial function* are assigned to false terms.
- (3) all don't care terms are assigned to false terms.
- (4) all don't care terms are assigned to true terms.

It is obvious, that assignments (3) and (4) do not lead to a maximal magnitude of s_I . Thus, the two values obtained for s_I according to assignments (1) and (2) have to be computed to determine the maximal possible magnitude of s_I for an incompletely

specified Boolean function. It follows from Equation (IV.2) that s_{I_1} for assignment (1) can be computed by Equation (VI.6) and s_{I_2} for assignment (2) by Equation (VI.7). Calculating s_{I_1} as

$$s_{I_1} = (a_I - b_I - e_I) - (c_I - d_I + f_I) \quad (\text{VI.6})$$

leads to the possible maximal negative value. Calculating s_{I_2}

$$s_{I_2} = (a_I - b_I + e_I) - (c_I - d_I - f_I) \quad (\text{VI.7})$$

leads to the possible maximal positive value.

The elimination of b_I and d_I which are not directly available in the T -spectrum leads to

$$s_{I_1} = 2 \times (a_I - (c_I + f_I)) \quad (\text{VI.8})$$

$$s_{I_2} = 2 \times ((a_I + e_I) - c_I) \quad (\text{VI.9})$$

As one can observe, Equations (VI.8) and (VI.9) are similar to Equation (IV.3) for completely specified Boolean functions. However, in both cases, s_{I_1} and s_{I_2} , the intersection of the incompletely specified Boolean function with the *positive standard trivial function* p_I and the *negative standard trivial function* n_I has to be computed to obtain a_I , c_I , e_I , and f_I .

For the linearization the assignment of the not specified part is determined by the coefficient s_{I_1} or s_{I_2} having the larger magnitude. After the first coefficient has been selected, the not specified part has to be specified according to the selected assignment of s_{I_1} or s_{I_2} . Therefore, the following linearization procedure is based on a completely specified Boolean function, which was discussed in the Chapter VI.2.1.

The complexity of a system of Boolean functions is given by the sum of the complexities of the single functions (Equation(VI.3)). Therefore, the spectral coefficient s_I for the system of Boolean functions is computed as follows

$$s_I = \sum_{j=0}^k s_{I_j} \quad (\text{VI.10})$$

where k is the number of functions, and s_{I_j} is the value of the spectral coefficient for the j^{th} function obtained as described above. Thus, the linearization procedure given by Algorithm 5 can be applied to the spectrum for incompletely specified, multi-output functions without any change.

Similarly to the approach shown for the linearization of incompletely specified Boolean functions based on the Rademacher-Walsh spectrum, a method for the calculation of the autocorrelation based on the T -spectrum can be developed.

The next step in the synthesis of systems for incompletely specified Boolean functions by linearization is to calculate the remaining nearly unate core function $F_{\sigma}(X)$.

VI.4 CALCULATION OF THE CORE FUNCTION $F_{\sigma}(X)$

The EXOR preprocessor, described by the matrix σ , is obtained by the methods described in the previous section. The input variables x_i of the original system of functions $F(X)$ are now the input to the EXOR preprocessor σ . The output functions x'_i of the EXOR preprocessor are directly obtained from the core matrix σ .

Example VI.2: Let us assume the following matrix σ

$$\sigma = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{matrix}$$

The matrix σ describes the EXOR preprocessor $x'_1 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$, $x'_2 = x_3$, $x'_3 = x_3 \oplus x_4$, and $x'_4 = x_2 \oplus x_3$, where x'_i are the input variables to the core function F_{σ} .

The remaining core function $F_\sigma(X)$ depending on the variables x_i can be computed by determining the spectrum of the core function with following computation of the core function by the inverse Walsh transform like applied in [65,67] or by converting the minterms of the original function by multiplication with the matrix σ [12]. Both methods are based on the computation of minterm representations and are therefore very inefficient. In [12] a matrix method to compute the core function f_σ directly from the original function f in any SOPE form has been suggested. This method is here modified to take advantage of incompletely specified Boolean functions and the calculation by simple cube calculus operations.

Example VI.3: In Figure 20.a the function $f(X) = \bar{x}_1\bar{x}_2 + x_2x_3$ is given. The introduction of an EXOR preprocessor for $x'_2 = x_2 \oplus x_3$ results in the interchange of the minterms illustrated in Figure 20.a. The obtained core function which is the result of the interchange is given in Figure 20.b.

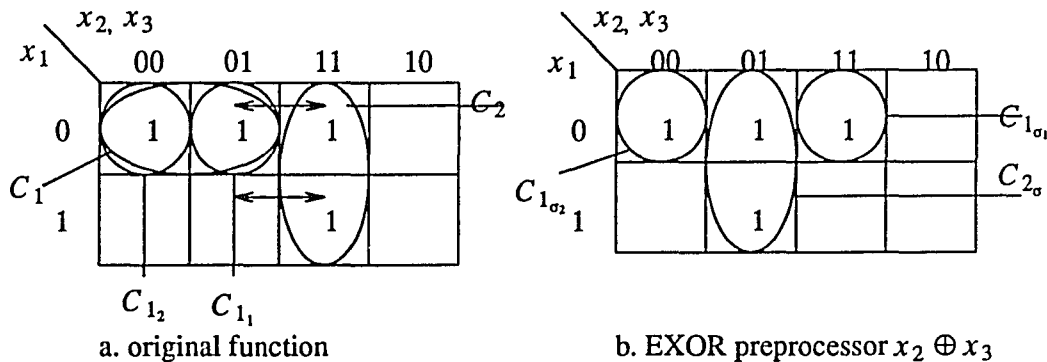


Figure 20. EXOR preprocessor example.

As one can observe from Figure 20, the interchange has to be performed for the area of the map covered by variable x_3 , where the interchange is given by the negation of variable x_2 .

In general there are three cases that have to be distinguished to obtain the cube c_σ of the core function $F_\sigma(X)$ for a cube c of the original function when an input variable x_i is replaced by $x'_i = x_i \oplus x_j$.

Case 1: The intersection of the original cube with literal x_j is empty:

$$c \cap x_j = \emptyset$$

therefore the obtained cube c_σ of the core function is given by $c_\sigma = c$.

Case 2: The original cube c is covered by the cube described by variable x_j

$$c \cap x_j = c$$

c_σ is obtained from c by the negation of variable x_i . In case variable x_i of the cube c is a dc literal, $c_\sigma = c$.

Case 3: The original cube c has the property

$$c \cap x_j \neq \emptyset$$

and

$$c \cap \bar{x}_j \neq \emptyset$$

then cube c has to be split into the two cubes $c_1 = c \cap x_j$ and $c_2 = c \cap \bar{x}_j$.

The cubes c_{σ_1} and c_{σ_2} of the core function $F_\sigma(X)$ are obtained by applying

Case 1 for c_2 and Case 2 for c_1 .

The following example illustrates the application of the three cases.

Example VI.4: For the function $f(X)$ given in Example VI.3, the cube C_1 is of the type given by Case 3. Therefore it has to be splitted into two cubes $C_{1_1} = \bar{x}_1 \bar{x}_2 \bar{x}_3$ and $C_{1_2} = \bar{x}_1 \bar{x}_2 x_3$. Thus, $C_{1_{\sigma_1}} = C_{1_1} = \bar{x}_1 \bar{x}_2 \bar{x}_3$. The two other cubes describing the core function are obtained by the negation of the variable x_2 of C_{1_2} and C_2 : $C_{1_{\sigma_2}} = C_{1_2}(\bar{x}_2) = \bar{x}_1 x_2 x_3$, and $C_{2_\sigma} = \bar{x}_2 x_3$.

VI.5 EVALUATION AND BENCHMARK RESULTS

The linearization procedure based on the Rademacher-Walsh spectrum and the autocorrelation spectrum for incompletely specified multi-output Boolean functions has been implemented for the evaluation of the efficiency of the presented method.

It is known [4,12] that arithmetic, signal processing, and error correcting logic functions have highly correlated output functions. Therefore, the approach to compute the core function σ based on Equation (VI.3), where the complexity of the system of Boolean functions is the sum of the complexities of each Boolean function, should be applicable. However, many control functions and randomly generated functions have very weakly correlated output functions. Thus, they do not benefit from the linearization performed on the system of functions. Therefore, we additionally implemented a linearization procedure to linearize each output function separately.

The number of product terms and the literal count of a function proved to be a good heuristic to determine the complexity of a function with respect to its realization with logic synthesis tools based on theunate paradigm [4]. Therefore, to evaluate the performance of the linearization procedure based on the different spectra and its application to systems and separate Boolean functions, the number of product terms and their literal count has been computed before and after the linearization.

Table III presents the results for the different applications of the linearization procedure for standard MCNC benchmark functions. The columns *prod* give the number of product terms obtained by Espresso [1,4]. The columns *lit* give the literal count of the results. The results obtained for the initial input function are given in the column *original*. The columns *walsh*, *autocorrelation* and *separate* give the results for the core function obtained by the application of the linearization procedure based on the respective spectrum, where *separate* gives the result for the application of the linearization pro-

cedure based on the Rademacher-Walsh spectrum to each output function separately. The results obtained by the SPECSYS system [12] are given as comparison.

TABLE III
LINEARIZATION

function	in	out	original		walsh		autocorrelation		SPECSYS		separate	
			prod	lit	prod	lit	prod	lit	prod	lit	prod	lit
adr2	4	3	11	32	10	29	4	10			7	15
b12	15	9	43	149	79	405	42	143			65	193
bw	5	28	22	102	22	107	22	107			107	107
clip	9	5	120	614	93	510	48	207	46	263	112	456
con1	7	2	9	23	10	28	11	29	11	42	10	23
f51m	8	8	76	319	81	411	70	324			44	130
inc	7	9	29	133	30	138	31	141			54	212
misex1	8	7	12	51	12	47	12	48	12	94	29	93
misex3c	14	14	196	1299	332	2356			195	1614		
rd53	5	3	31	140	16	61	12	40	12	55	12	39
rd73	7	3	127	756	78	393	78	393	78	474	78	393
rd84	8	4	255	1774	156	947	156	947	156	1107	156	947
sao2	10	4	58	421	51	342	45	251	47	324	102	542
squar5	5	8	25	87	25	87	25 (*)	89			26	76
table3	14	14	175	2001	218	2489	181	2087				
5xp1	7	10	65	262	113	643	65	287			48	142
Z5xp1	7	10	65	287	108	645	67	289	67	406	48	142
Z9sym	9	1	86	516	56	280	56	280	56	370	56	280

(*) no EXOR introduced

The results obtained prove the validity of the stated properties for the application of the linearization procedure to the two classes of functions. Where the linearization on systems of Boolean functions leads to core functions that have a higher complexity than the original function, the procedure based on the linearization of each output function gives better results. Therefore, it can be concluded that the functions Z5xp1 and f51m have very weakly correlated output functions. For all other functions the linearization performed directly on the system of functions lead to a reduction of the complexity with

respect to the number of product terms and the literals count. It can be observed, that on the average the linearization procedure based on the autocorrelation spectrum leads to better results than to the one based on the Rademacher-Walsh spectrum. However, this is traded off with an in magnitudes increased computation time.

This Chapter introduced the generalization of the linearization procedure to systems of incompletely specified Boolean functions. The application of the computation methods for the Walsh transform presented in Chapter IV and Chapter V decrease both, the computation time and the memory requirement. Additionally, the computation of the remaining core function by the introduced cube calculus method further speeds up the linearization process. The selection of the spectral coefficients based on the lowest Rademacher order has been selected to decrease the complexity of the linear preprocessor σ .

Another approach for the efficient synthesis of nearly linear functions it to take advantage of the powerful multiplexer gate. Therefore, a complete multi-level synthesis algorithm for multiplexer circuits will be presented in the next Chapter.

CHAPTER VII

HIERARCHICAL MULTIPLEXER SYNTHESIS FOR BOOLEAN FUNCTIONS

Several multi-level synthesis tools have been developed for the minimization of logic functions to obtain a low literal count Boolean network [4-7,86,114]. A drawback of these tools is that they do not take the efficient synthesis for nearly linear functions into consideration.

One approach for the synthesis of logic functions is to take advantage of the powerful multiplexer gate. It has been shown [63,115,116] that multiplexers are universal logic modules, where a multiplexer of k data-select inputs can realize any function of $k + 1$ input variables, under the assumption, that the complements of the input variables are also available. Thus, they can be used for the synthesis of multilevel logic networks. One can observe, that an n -variable linear function can be realized by a cascade multiplexer circuit of $M(k)$ multiplexers where $\left\lceil \frac{n-1}{k} \right\rceil$ multiplexers are necessary.

Many synthesis algorithms for different kinds of multiplexer circuits have been developed [117-128]. One synthesis method with multiplexers is to find a single multiplexer, if possible of the minimum size, which realizes the given Boolean function [121-123,127]. Algorithms have been presented in [117,125,128] to find a possible cascade multiplexer circuit of a Boolean function. The algorithms [122-125] are based on graphical methods which allow only the synthesis for functions with up to six variables. A third realization, which will be further developed in this Chapter, is known as multiplexer tree circuit [64,65,118-120,123,124,129]. All the previous algorithms operate on minterms,

while the method presented here is based on disjoint cubes.

One of the motivations to investigate the synthesis for multiplexer tree circuits gives the ACT^{TM} FPGA family from Actel [46], where the basic building block consists of multiplexers, Figure 21.

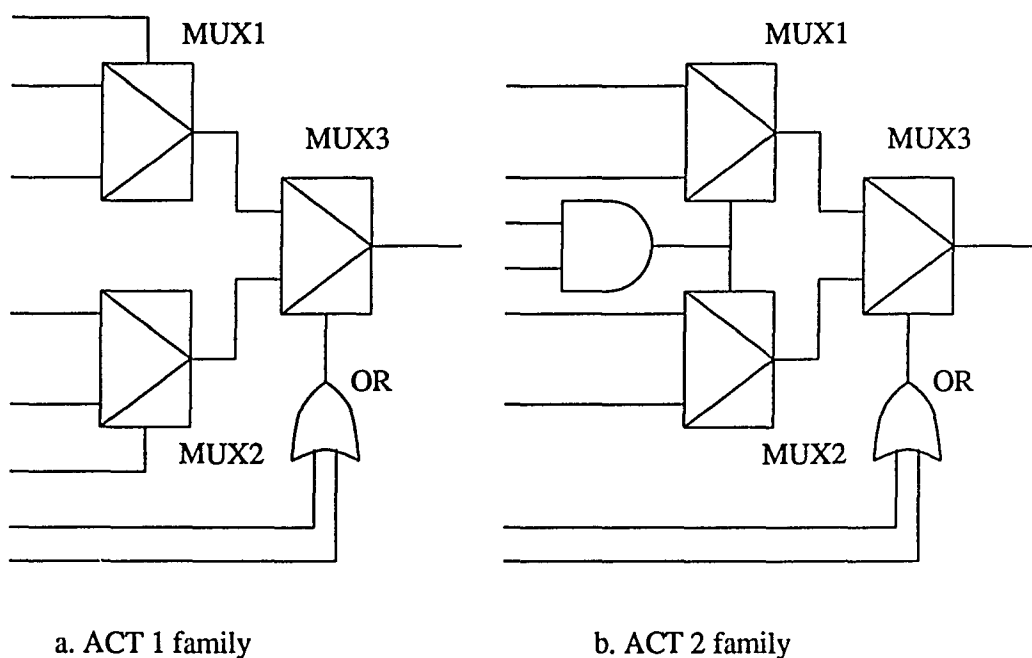
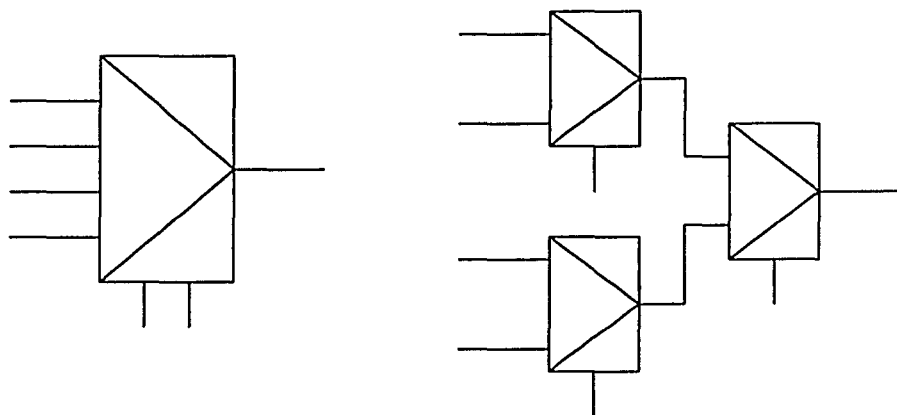


Figure 21. Basic building block of the ACT^{TM} family.

Each basic building block of the ACT^{TM} family allows the implementation of a M(2) multiplexer (Figure 22.a), and in the case of the ACT 1 family also of three hierarchical M(1) multiplexers (Figure 22.b). As can be observed from Figure 21.b the ACT 2 family allows only a restricted realization of three hierarchical M(1) multiplexers. Another motivation for the interest in multiplexer circuits is that a function realized by multiplexers should have less modules than one constructed with conventional logic gates (NAND, NOR) [128]. We will compare the results obtained by technology mappers that support the explicit library of realizable functions of the Actel FPGAs [130,131] and the results of special mapping algorithms which take advantage of the structure of the Actel



a. M(2) multiplexer

b. three M(1) multiplexers

Figure 22. Realizable multiplexers with ACT^{TM} family.

macrocells [52,130,132] to the results of the mapped multiplexer circuit obtained by the synthesis algorithm presented here.

In this Chapter we further develop the methods to find redundant multiplexer modules in a tree circuit [64,117,119,127-129] for the level-by-level top down (starting from the output) minimization of multiplexer tree circuits [64,119]. The two methods for single-output completely specified Boolean functions, one based on Ratio Parameters for M(1) multiplexer synthesis [117,127,128] and the second based on the Rademacher-Walsh spectrum for M(2) multiplexer synthesis [64,65] are here uniformly generalized to the M(k) multiplexers synthesis for multi-output incompletely specified Boolean functions. Because computing the complete Walsh spectrum [64] to determine the redundant next level multiplexer modules is complex we introduce the concept of a *local* T-transform (see Chapter IV) for the data-select variables. This notion is similar to the Ratio Parameter developed for the M(1) synthesis in [117].

VII.1 GENERAL MULTIPLEXER SYNTHESIS

First we have to introduce the concept of polarity to be able to give the general formula of a multiplexer $M(k)$.

Definition VII.1: The *polarity* of a product term

$\dot{x}_1 \dot{x}_2 \dots \dot{x}_i \dots \dot{x}_n$ is defined by the binary string of values being 0 for $\overline{x_i}$ and 1 for x_i respectively.

The general case of a multiplexer $M(k)$, where k is the number of data-select inputs, can be defined as a special case of Equation (II.9).

Definition VII.2: The output function $f(x_0, x_1, \dots, x_{n-1}) = f(X)$ of a multiplexer $M(k)$ with k data-select inputs $d_j(X)$ is given by

$$f(X) = \bigcup_{i=0}^{2^k-1} \dot{d}_1(X) \cap \dots \cap \dot{d}_j(X) \cap \dots \cap \dot{d}_k(X) \cap f(X) \quad (\text{VII.1})$$

where $\dot{d}_j(X) \in \{d_j(X), \overline{d_j(X)}\}$, being any Boolean function, with the polarity determined by the binary representation of i .

For multiplexer synthesis algorithms the data-select functions $d_j(X)$ are commonly restricted to input variables [64,119]. Thus, the form of Equation (VII.2) is obtained

$$f(X) = \bigcup_{i=0}^{2^k-1} \left[f(X) \right]_{\dot{x}_1 \dots \dot{x}_k} \quad (\text{VII.2})$$

where the polarity of the data-select variables $\dot{x}_1 \dots \dot{x}_k$ is determined by the binary representation of i . This is the well known Shannon expansion. It follows that the data-input function $f_i(X)$ is given by

$$f_i(X) = \left[f(X) \right]_{\dot{x}_1 \dots \dot{x}_k} \quad (\text{VII.3})$$

Example VII.1: An n variable Boolean function $f(X)$ with the data-select inputs x_1, x_2 is

decomposed to

$$f(X) = \bar{x}_1 \bar{x}_2 f_0(X) + \bar{x}_1 x_2 f_1(X) + x_1 \bar{x}_2 f_2(X) + x_1 x_2 f_3(X) \quad (\text{VII.4})$$

which is illustrated in Figure 23.

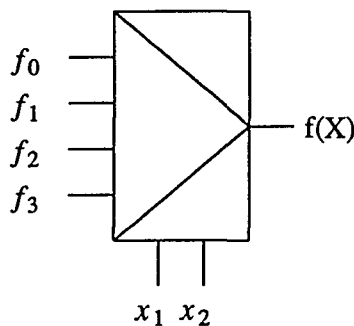


Figure 23. Standard multiplexer M(2) according to Equation (VII.4).

The functions f_i are the data input functions and the variables x_1, x_2 are the data select variables of the multiplexer. Let us observe that the functions f_i do not depend on x_1 and x_2 .

The general problem in multiplexer synthesis is to find a circuit realization with the minimum number of multiplexers for a given multi-output incompletely specified Boolean function. According to Equation (VII.1) such a circuit can have multiplexers with various numbers of data-select inputs where each data-select input can be any function $d_j(X)$ [115,129]. Because finding the optimal data-select functions being any possible Boolean function is very complex, the multiplexer synthesis algorithms find (quasi)-optimal realizations according to Equation (VII.2), where the data-select variables are input variables [118-121,123-128].

Such a minimal multiplexer circuit can have different permutations of data-select variables in any branch of the circuit [120,129]. To decrease the complexity of the minimization problem, most multiplexer synthesis algorithms assume the same data-select variables in any level of the tree circuit [64,118-120,129], as shown in Figure 24.

This restriction is also advantageous for a possible mapping to the *ACT2* family, Figure 21.b, where the data-select input variables of MUX1 and MUX2 are given by the AND of two variables.

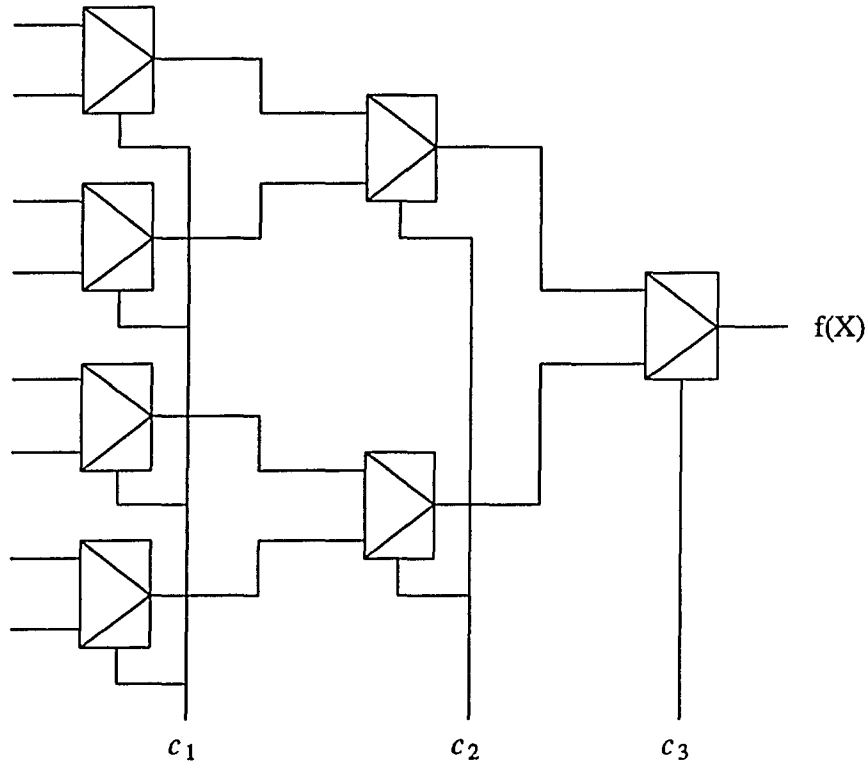


Figure 24. Restricted Multiplexer tree circuit.

As was stated in [129] the minimal upper bound for the levels in such a restricted multiplexer tree circuit is given by

$$L = \frac{n-1}{k} \quad (\text{VII.5})$$

and the minimal upper bound for the number of multiplexer modules $M(k)$ by

$$M = \sum_{i=0}^{L-1} 2^{ik} \quad (\text{VII.6})$$

Still an exhaustive search has to be performed to find the optimal permutation of the data-select variables [118,124,129]. Level-by-level minimization algorithms

[64,119] decrease the necessary computation and storage requirements for the implicitly exhaustive algorithms to find the optimum tree circuit. It was conjectured [119] that the level-by-level minimization is still optimum or near optimum. It should be observed that the special case of M(1) multiplexer synthesis is equivalent of finding the minimal Shared Ordered Binary Decision Diagram (SOBDD) representation of the Boolean function with attributed edges [4,133,134]. Thus, the multiplexer synthesis algorithm for M(1) multiplexers can be applied as a heuristic to find a good variable ordering for the SOBDD.

The next section investigates the conditions for which the next level multiplexer modules are redundant.

VII.2 REDUNDANCY OF MULTIPLEXER MODULES

The basic principle of the level-by-level minimization algorithm from [64,119] is to find the minimal number of next level modules for a given level. This approach will be adopted here. A similar principle is used for the realization of Boolean functions as cascade multiplexer circuits or single multiplexer circuits, where only one next level module is allowed or no module at all [117,121-123,125,127,128].

There exist three basic conditions for which a next level module is redundant

Condition 1: a data-input function is a trivial

$$\begin{aligned} f_i &= 0 \\ f_i &= 1 \\ f_i &= x_j \\ f_i &= \overline{x_j} \end{aligned}$$

Condition 2: a data-input function is identical to another data-input function to a multiplexer in the same level of the tree circuit

$$f_i = f_j \quad i \neq j$$

Condition 3: a data-input function is the complement of another data-input function to a multiplexer in the same level of the tree circuit

$$f_i = \overline{f_j} \quad i \neq j$$

In most algorithms only the first condition is taken into consideration to decrease the number of next level modules [118-120,123,127,129]. The case of a data-input function being the complement of another data-input function has not been taken into consideration in any synthesis algorithm. The advantage of the presented method is, that it verifies also Condition 3. Even if no inverters are available (like in the Actel FPGAs), only one multiplexer is necessary to realize the complemented function instead of a complete subtree. The complemented function can be realized by a control function circuit [129] as shown in Figure 25.

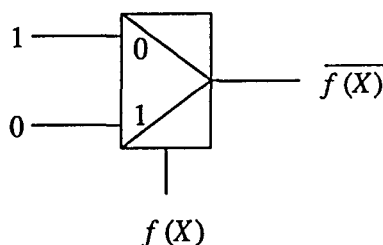


Figure 25. Control Function Circuit for function complementation.

In the case that in a particular level of the multiplexer circuit different combinations of k data-select variables require the same number of next level multiplexer modules, a selection should take into account the possible further minimization in the next lower levels of the multiplexer circuit. Spectral methods are ideally suited for this case, because they give insights into the global structure of the underlying Boolean function [64,65,117]. Therefore, the multiplexer synthesis can be based on the information obtained from the correlation between the Boolean function and the *standard trivial functions* which are determined by the data-select variables.

Definition VII.3: The *positive standard trivial function* p_i represented by the k data-select variables for an $M(k)$ multiplexer is given by

$$p_i = \dot{x}_1 \dots \dot{x}_j \dots \dot{x}_k \quad (\text{VII.7})$$

where i is the binary representation of the polarity of the k data-select variables. It follows from Equation (VII.3)

$$f_i(X) = p_i \cap f(X) \quad (\text{VII.8})$$

The number of minterms covered by the *positive standard trivial function* p_i is given by

$$pt = 2^{n-k} \quad (\text{VII.9})$$

Because in this Chapter we only deal with *positive standard trivial functions* we denote them just by *trivial function*.

The concept of Ratio Parameters [117,127,128] has been developed to determine if data-input functions to an $M(1)$ multiplexer are trivial. The Ratio Parameter method is essentially a spectral method like the method for $M(2)$ multiplexer synthesis [64,65]. Both methods are based on determining the number of minterms covered by the *trivial functions* p_i given by the data-select variables, as shown in the Example in Figure 26.

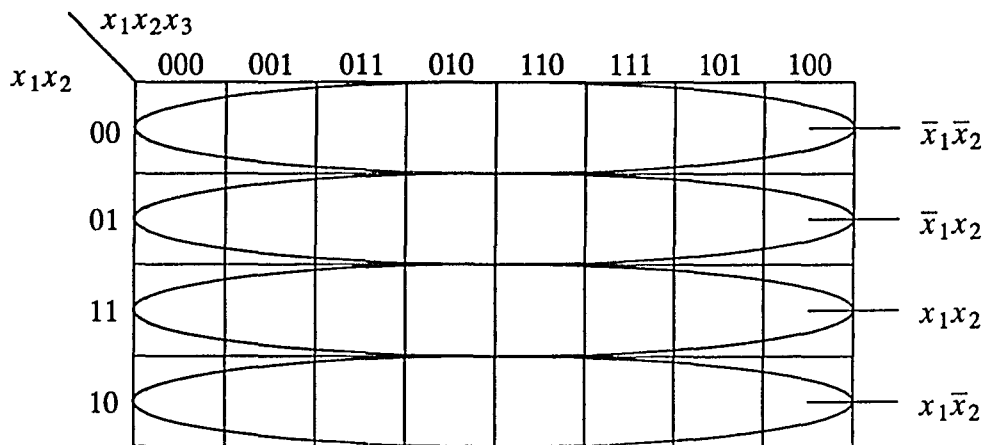


Figure 26. Trivial functions for chosen data-select variables x_1 and x_2 .

The spectral method and the Ratio Parameter method can be uniformly generalized for the synthesis with $M(k)$ multiplexers by the introduction of a *local* transform. To be able to incorporate incompletely specified Boolean functions we apply the T-spectrum introduced in Chapter IV.3.

For $M(2)$ multiplexer synthesis [64] the Rademacher-Walsh spectrum was adopted to find the correlation of the Boolean function to the *trivial functions*. A further summation [64,65] of subsets of spectral coefficients of the Rademacher-Walsh spectrum is necessary to obtain the final spectrum describing the correlation to the *trivial functions* p_i . This summation is realized by a 4×4 Rademacher-Walsh transforms on subsets of the initial Rademacher-Walsh spectrum. In the general case of a $M(k)$ multiplexer the summation can be realized by a $2^k \times 2^k$ Rademacher-Walsh transform on the respective subset of the initial Rademacher-Walsh spectrum for each possible set of k data-select variables.

The *local* T-transform performs in one step both the initial calculation of the complete Rademacher-Walsh spectrum of the function and the following summation of a subset of coefficients. Thus, one avoids the complexity of computing the whole Rademacher-Walsh spectrum for a given function. An $M(k)$ multiplexer for an n -variable Boolean function can have $\binom{n}{k}$ different possible combinations of k data select variables. Table IV compares the number of coefficients that have to be calculated for a $M(2)$ multiplexer synthesis by using the Rademacher-Walsh transform and the *local* T-transform.

The first column of Table VI gives the number of input variables of a Boolean function. The second column gives the number of spectral coefficients of the initial Rademacher-Walsh spectrum. Finally, the last column gives the number of spectral coefficients for all possible pairs of data-select variables, where a spectrum for a data-select pair consists of four coefficients. The formulaes for the calculation of the number

TABLE IV
COMPARISON OF THE NUMBER OF COEFFICIENTS

variables	Rademacher-Walsh coefficients	spectral coefficients for $\binom{n}{2}$ pairs of data-select variables
n	2^n	$\binom{n}{2} \times 4$
5	32	10×4
10	1024	45×4
20	1048576	190×4
30	10^9	435×4

of coefficients are given in the second row.

As one can observe from Table VI, the computation of spectra for functions with more than 20 variables is not feasible with general Fast Transforms. The Rademacher-Walsh spectrum can be calculated directly from a disjoint representation [12,73,94], but still the complexity of calculating all 2^n coefficients can be prohibitive. For the decomposition to multiplexer modules given by Equation (VII.2) we are only interested in the correlation between the *trivial functions* and the data-input functions to the multiplexer. Thus, the direct calculation of only the necessary spectral coefficients for the combinations of k data-select variables by the *local* T-transform is much more efficient.

The following property gives the relation of the spectral coefficients t_i obtained by the *local* T-transform for completely specified Boolean functions to the spectral coefficients s_i used in [64].

Property VII.1: The spectral coefficients s_i of the spectrum $S_{x_1 \dots x_k}$ for k data-select variables $x_1 \dots x_k$ are computed as follows

$$s_i = (pt - 2 \times a_i) \times 4 \quad (\text{VII.10})$$

where pt is the number of minterms covered by the *trivial function* p_i , and $t_i \in \{a_i\}$.

For incompletely specified Boolean functions the value of $t_i \in \{a_i, pt - b_i\}$ is

chosen which ever leads to a heigher absolute value of s_i .

A criterion has been introduced in [64]. for the selection of one combination of k data-select variables for the case that more than one combination leads to the same minimal number of next level modules . The criterion is based on the information given by the spectral coefficients. It was stated in [64] that a high value of the sum of absolute values of the spectral coefficients,

$$sum = \sum_{i=0}^{2^n-1} |s_i| = |S_{\dot{x}_1 \dots \dot{x}_k}| \quad (\text{VII.11})$$

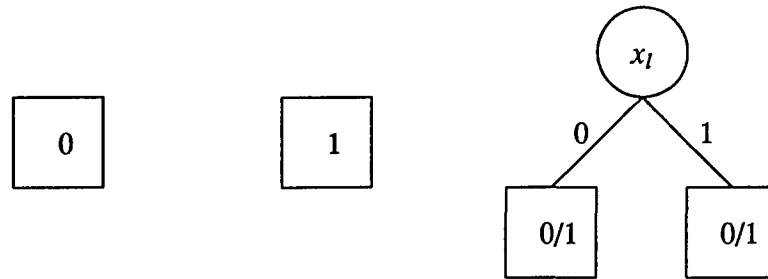
gives a higher possibility for further minimization in the following levels. This is based on the assumption, that a higher density of false or true minterms in the data-input functions allows the further minimization in the next levels. Therefore, we can formulate an additional Condition 4 for multilevel realizations

Condition 4: for different combinations of k data-select variables with the same minimal number of next level modules the one with the highest value of sum is chosen. If several combinations have the same highest value of sum , one of them is selected randomly.

With the above Definitions and Properties we are able to formulate the conditions for redundancy of next level modules by spectral and Boolean means. For the verification of some of the conditions for the redundant next level modules, the OBDD representation [78,79,103,104] of a Boolean function is very useful. Figure 27 shows that it is easy to verify directly from the OBDD representation that a completely specified Boolean function is trivial. If the OBDD consists of only terminal nodes or an additional node, then the function is trivial. However, the identification of *trivial functions* for incompletely specified Boolean functions is more complicated.

Verification of Condition 1: The trivial incompletely specified data-input functions f_i have the following properties in the spectral domain, for $f_i = 0$:

$$a_i = 0 \quad (\text{VII.12})$$



a. trivial function 0 b. trivial function 1 c. trivial function x_i

Figure 27. Trivial functions in OBDD form.

and for $f_i = 1$:

$$b_i = 0 \quad (\text{VII.13})$$

Equation (VII.12) can be verified in Boolean domain for the computation with a cube representation by

$$f_i = f \cap p_i = f_{dc} \quad (\text{VII.14})$$

where f_{dc} consists only of not specified terms. Equation (VII.13) can be similarly verified by

$$f_i = f_{\rightarrow 1} \cap p_i = p_i \quad (\text{VII.15})$$

where the notation $\rightarrow 1$ denotes the specification of the not specified part of the incompletely specified Boolean function f to true terms. In the cube representation just the respective output literals have to be changed from not specified to true terms to obtain this transformation. Symbolically in an OBDD this can be realized by connecting the don't care terminal to the 1/0 terminal, Figure 28.



Figure 28. Incompletely specified function to 1.

However, this requires a complete restructuring of the OBDD for an incompletely specified Boolean function to the one for completely specified Boolean functions.

A necessary condition for a completely specified function being dependent on only one variable x_l is

$$s_i = 0 \quad (\text{VII.16})$$

which is equivalent to

$$a_i = 2^{n-k-1} \quad (\text{VII.17})$$

For an incompletely specified Boolean function it is necessary that the not specified terms can be specified in such a way that Equation (VII.16) is fulfilled. This can be verified in Boolean domain by

$$f_{\rightarrow 1}(X) \cap x_l = x_l \quad (\text{VII.18})$$

and

$$f(X) \cap \bar{x}_l = f_{dc} \quad (\text{VII.19})$$

where f_{dc} consists only of not specified terms.

Verification of Condition 2: The assignment of the not specified minterms to true or false ones to obtain identical data-input functions ($f_i = f_j$), if possible, can be easily verified in spectral domain for completely specified Boolean functions by:

$$a_i = a_j \quad (\text{VII.20})$$

However, if Equation (VII.20) is true, still the identity of the two functions has to be checked. With OBDDs in Boolean domain this can be easily performed by a simple pointer comparison. For incompletely specified Boolean functions it has to be computed if the not specified parts of the functions f_i and f_j can be specified in such a way that $f_i = f_j$. This can be verified by the complement of the intersection of the two incompletely specified functions

$$f_{in} = f_i \cap f_j \quad (\text{VII.21})$$

having no common minterms with the completely specified part of function f_i

$$f_i \cap \overline{f_{in}} = f_i \# f_{in} = f_{dc_1} \quad (\text{VII.22})$$

and no common minterms with the completely specified part of function f_j

$$f_j \cap \overline{f_{in}} = f_j \# f_{in} = f_{dc_2} \quad (\text{VII.23})$$

where $\#$ denotes the sharp operation [100] and f_{dc_1} and f_{dc_2} consist only of not specified terms. If the Equations (VII.22) and (VII.23) are true, the not specified part of f_i and f_j can be specified in such a way that $f_i = f_j$.

Verification of Condition 3: A necessary condition for a data-input function f_i being the complement of a completely specified data-input function f_j is

$$b_i + b_j = a_i + a_j = 2^{n-k-1} \quad (\text{VII.24})$$

To find a possible specification of the not specified part of the incompletely specified Boolean function such that two data-input functions are complemented, Equation (VII.25) and (VII.26) have to be true

$$f_i \cap f_j = f_{dc} \quad (\text{VII.25})$$

to verify that the specified parts of both functions have no common minterms. The equivalent of Equation (VII.24) in Boolean domain is given by

$$f_{i \rightarrow 1} + f_{j \rightarrow 1} = p_i \quad (\text{VII.26})$$

If both formulas can be fulfilled, the not specified part of the data-input functions f_i and f_j can be chosen in such a way that $f_i = \overline{f_j}$.

The above conditions have to be computed for all possible combinations of data-select variables. In case there is more than one set of data-select variables that have a minimum number of next level variables, the one according to Condition 4 is chosen.

The verification of the four conditions allows the determination of the data-select select variables for a single level of multiplexer modules. For the bottom-up level-by-level minimization of a multi-output incompletely specified Boolean function the primary output level of the function (which consists of more than one multiplexer) is treated like any other level of the multiplexer tree circuit.

The overall structure of the multi-level synthesis algorithm is given by Algorithm 6.

Algorithm 6: Multiplexer Synthesis algorithm

```

// k      : number of data-select variables
// output_functions : number of output functions for given Boolean function
// input_variables : number of input variables for given Boolean function

mux_synthesis(k)
{
    min_level_modules = output_functions * 2k;
    level = 0;      // determines current level in circuit
    while ( min_level_modules != 0 ){
        // computation of minimal number of next level modules for given level
        for ( i = 0 ; i <  $\left\lfloor \frac{\text{input\_variables} - \text{level} * k}{k} \right\rfloor$  ; i++ ) {
            // all permutations of data-select variables
            data_select_variables = generate_data_select(k,i);
            // compute number of next level modules for given data_select_variables
            level_modules = compute_modules(data_select_variables);
            sum =  $\sum_{n=0}^{\text{level\_modules}}$  sumn; // sum of all multiplexers
            if ( min_level_modules > level_modules ){
                // current data_select_variables lead to less next next level modules
                max_sum = sum;
                min_level_modules = level_modules;
                best_select = data_select_variables;
            } else if ( min_level_modules == level_modules ){
                if ( sum > max_sum ){
                    max_sum = sum;
                    best_select = data_select_variables;
                }
            }
        }
    }
}

```

The developed methods are illustrated in the next section with a complete example.

VII.3 SYNTHESIS EXAMPLE

The synthesis steps from the previous section will be illustrated on the example from [64] :

$$f(X) = \bar{x}_1 x_4 \bar{x}_5 + x_1 x_2 x_4 + \bar{x}_1 x_2 x_3 \bar{x}_4 + \bar{x}_3 \bar{x}_4 x_5 + x_1 \bar{x}_4 x_5$$

For the initial verification of the redundancy of next level modules the *local* T-spectra for each combination of data-select variables, and the *sum* parameter are

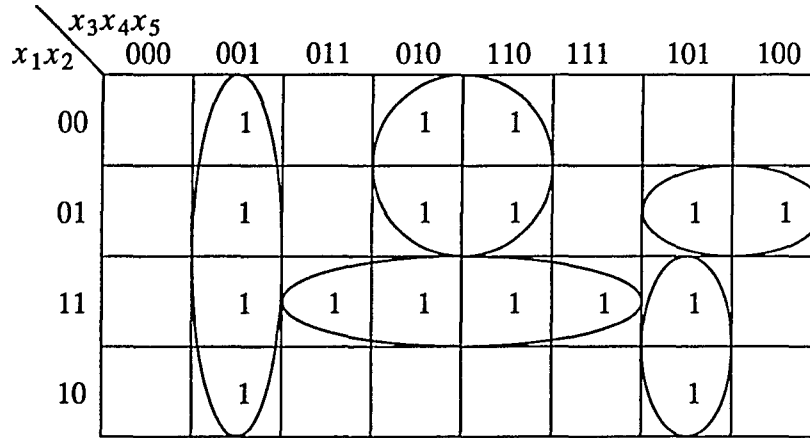


Figure 29. Karnaugh-map of synthesis example.

calculated. The calculation of the initial spectra $T_{x_i x_j}$ will be illustrated for the spectrum of the data-select pair x_1, x_2 . The value of the spectral coefficients of the spectrum $S_{x_i x_j}$ [64] can be obtained by Equation (VII.10). The *trivial functions* p_i for the *local* T-transform of the chosen data-select pair are: $p_0 = \bar{x}_1 \bar{x}_2$, $p_1 = \bar{x}_1 x_2$, $p_2 = x_1 \bar{x}_2$, $p_3 = x_1 x_2$, where each *trivial function* covers $pt = 2^{5-2} = 8$ minterms, Equation (VII.9).

First the intersection of the function $f(X)$ with each *trivial function* has to be calculated to obtain the *local* T-spectrum for the chosen data-select pair, e.g.

$$f_0 = f(X) \cap p_0 = \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 x_5 + \bar{x}_1 \bar{x}_2 x_4 \bar{x}_5$$

However, the data-input functions are now independent from the data-select variables x_1 and x_2 :

$$f_0 = \bar{x}_3 \bar{x}_4 x_5 + x_4 \bar{x}_5$$

Because the function $f(X)$ is completely specified, the spectral coefficients of the *local* T-spectrum are given by $t_i = a_i$. They can be directly computed from the disjoint cube representation of the data-input functions f_i : $a_0 = 3$, $a_1 = 5$, $a_2 = 2$, and $a_3 = 6$. No coefficient $t_i \in \{a_i\}$ fulfills Equation (VII.12), (VII.13) or (VII.17) which verify the possibility of trivial data-input functions. Similarly, the criterion for the identity of two

data-input functions, Equation (VII.20), is not satisfied. However, the necessary condition for complemented data-input functions is true for $a_2 + a_3 = 8$. Therefore, this case has to be verified further. Applying Equation (VII.25), we obtain $f_2 \cap f_3 = x_4 \neq \emptyset$. Because the intersection is not empty, f_2 can not be the complement of f_3 . No data-input of the multiplexer is redundant. Therefore, the number of next level modules Mod for the data-select variables x_1 and x_2 is $Mod = 4$. To be able to check later the Condition 4, the $sum = 48$ is computed by applying Equation (VII.10) and (VII.11).

The spectra for the other data-select pairs are calculated the same way. The results are listed in Table V. The exact minimum number of next level modules Mod can be directly calculated because all possible cases of redundant next level modules can be verified by Conditions (1)-(3), while in the method by Lloyd determines only an upper and lower bound $maxMod$ and $minMod$.

TABLE V
FIRST LEVEL SPECTRA

row	spectrum	t_0	t_1	t_2	t_3	sum	Mod
1	$T_{x_1x_2}$	3	5	2	6	48	4
2	$T_{x_1x_3}$	4	4	4	4	0	4
3	$T_{x_1x_4}$	4	4*	4*	4*	0	1
4	$T_{x_1x_5}$	5	3	2	6	48	4
5	$T_{x_2x_3}$	3	2	5	6	48	4
6	$T_{x_2x_4}$	3	2	5	6	48	4
7	$T_{x_2x_5}$	2	3	5	6	48	4
8	$T_{x_3x_4}$	4*	4	4	4	0	3
9	$T_{x_3x_5}$	3	5	4	4	16	4
10	$T_{x_4x_5}$	1	7	6	2	80	4

In Table V the symbol * indicates that the respective data-input function is either identical to another one, or a trivial one. Because $T_{x_1x_4}$ has the lowest number of next level modules, all other data-select pairs can be discarded. Thus, the best pair of data-select variables has been found in the first step, while for the pure spectral method [64] three

further verification steps are necessary.

The non-trivial data-input functions to the multiplexer module have to be decomposed further for the calculation of the next level of the hierarchical multiplexer circuit. The four data-input functions have been already obtained by the intersection of the *trivial functions* p_i for the data-select pair x_1, x_2 with the initial function for the calculation of the spectral coefficients: $f_0 = \bar{x}_3x_5 + x_2x_3$, $f_1 = \bar{x}_5$, $f_2 = x_5$, and $f_3 = x_2$. The only non-trivial function is f_0 . Therefore, f_0 has to be decomposed further. Because the function f_0 depends on only three variables it can be realized by one M(2) multiplexer and the choice of the data-select variables does not matter [128].

To illustrate the general synthesis procedure, the three spectra $T_{x_i x_j}$ are calculated (Table VI).

TABLE VI
SECOND LEVEL SPECTRA FOR f_0

row	spectrum	t_1	t_2	t_3	t_4	sum	Mod
1	$T_{x_2 x_3}$	4*	5*	4*	3*	16	0
2	$T_{x_2 x_5}$	5*	4*	4*	3*	16	0
3	$T_{x_3 x_5}$	5*	3*	4*	4*	16	0

The property, that no further multiplexer is necessary for any data-select pair is verified by the result given in Table V. The data-select pair x_2, x_3 is chosen for the realization. Thus, the final hierarchical multiplexer circuit has the form shown in Figure 30.

VII.4 MAPPING OF A MULTIPLEXER TREE CIRCUIT TO THE ACT1 MACROCELLS

While the hierarchical multiplexer circuits obtained for M(1) multiplexer can be mapped directly to FPGAs like the TPC10 series of Texas Instruments [50], the CLi6000 of Concurrent Logic [47] or the CAL 1024 of Algotronix [49] a special mapping algo-

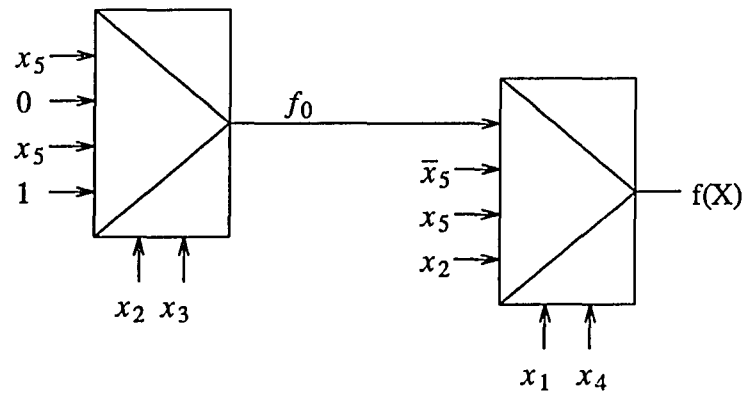


Figure 30. Final multiplexer realization.

rithm is necessary for the ACT^{TM} family of Actel. The macrocells provided by the ACT^{TM} FPGA family from Actel, Figure 21, have a restricted connectivity of M(1) multiplexers. A M(2) multiplexer can be realized with one macrocell both of the $ACT1$ and $ACT2$ family. Therefore, a multiplexer circuit based on M(2) multiplexers can be directly mapped to the Actel FPGAs. However, because of the lack of inverters, for each complemented data-input function an additional macrocell is necessary.

For our implementation we chose the $ACT1$ macrocell. The mapping of an M(1) multiplexer circuit to the $ACT1$ macrocell is restricted by the internal connectivity of the macrocells. The output functions of the two multiplexers at the input of the macrocell, MUX1 and MUX2, are not available as outputs of the macrocell. Therefore, multiplexers with fan-out > 1 have to be realized with the output level multiplexer MUX3 of a macrocell or the respective multiplexer has to be duplicated. As a heuristic we first map a multiplexer with fan-out > 1 to the MUX3 multiplexer of a macrocell. Additionally, input multiplexers to this multiplexer with fan-out $= 1$ are mapped to the same macrocell.

As a second step the multiplexers that have a function g and its complement \bar{g} as input are taken into consideration. Because of the restricted internal connectivity of a macrocell the complementation of a data-input function realized according to Figure 21,

has only one minimal mapping shown in Figure 31.

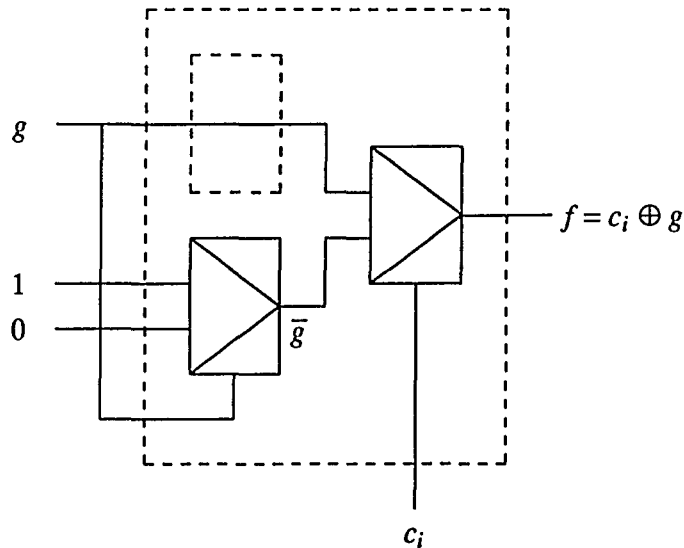


Figure 31. Mapping of complemented data-input functions.

However, if the function \bar{g} is an input to more than 2 other multiplexers, it should be realized with a separate macrocell. Finally the remaining multiplexers are mapped level-by-level to the macrocells.

The next section gives the results obtained by the implementation of the multiplexer synthesis algorithm followed by the mapping to the *ACT* 1 macrocell.

VII.5 BENCHMARK RESULTS

As stated in the previous section, the mapping of *M*(2) multiplexer circuits to the *ACT*TM FPGAs from Actel can be done directly. However, an *M*(1) multiplexer circuit has to be matched to the macrocells.

Table VII lists in column *mux-map* the results of the multiplexer synthesis algorithm for several MCNC benchmarks. For comparison we give the results of *misII* [57,130] with the Actel library of realizable functions as representative for general tech-

nology mappers, and the result of Actel specific mappers *mis-pga* [57,130] and *Prosperine* [59,132].

TABLE VII
BENCHMARK RESULTS FOR MULTIPLEXER SYNTHESIS

function	k = 2		k = 1		mux-map cells (ti.)	misII cells (ti.)	mis-pga cells (ti.)	Amap cells (ti.)	Prosperine	
	mod	depth	mod	depth					act0	act1
bw	61	2	106	4	68 (5.1)	81 (7.1)	60 (11.7)	83 (3.6)	87	63
clip	48	4	91	8	49 (21.5)	57 (4.9)	48 (25.1)	60 (2.5)	86	68
con1	8	3	24	6	9 (0.2)					
f51m	27	4	55	7	27 (4.2)	52 (5.2)	44 (14.4)	56 (2.2)	83	59
inc	35	3	94	6	60 (2.3)					
misex1	17	4	33	5	21 (0.5)	22 (1.9)	18 (6.0)	25 (1.1)	30	24
misex2	90	12	171	22	65 (0.6)	46 (4.3)	40 (3.5)	47 (1.5)	52	42
rd53	7	2	15	4	11 (0.2)					
rd73	14	3	29	6	23 (5.4)	30 (12.1)	32 (2.7)	32 (1.6)		
rd84	21	4	40	7	32 (24.2)	62 (6.5)	50 (30.7)	62 (2.6)	87	65
sao2	41	5	83	9	51 (3.1)	52 (8.0)	51 (24.4)	56 (2.0)		
squar5	19	2	34	4	19 (0.3)					
xor5	2	2	4	4	4 (0.0)					
5xp1	28	3	54	6	31 (2.7)	51 (4.4)	40 (10.4)	42 (1.7)	65	50
9sym	11	4	23	8	19 (3.6)	99 (14.1)	26 (55.8)	106 (4.1)		
	292		550		293 (58.8)	371 (34.3)	300 (101.8)	375 (15.2)	490	371

The results of the multiplexer synthesis given in the columns k=1 and k=2 assume that inverters are available to realize complemented data input functions. The columns *mod* give the total number of necessary M(1) or M(2) modules, and the columns *depth* gives the longest path in the circuit. The columns *cells* give the number of necessary macrocells. The time (*ti.*) is given in seconds. The computation times given for *mux-map* include the multiplexer synthesis and the mapping step. The results were obtained on a Sparc 4/370 (12.5 mips). For *mis-pga* (new) and Amap the time was obtained on a DEC 5500 (28 mips).

It should be stressed, that *mux-map* performs the synthesis and mapping for the *act0* type combination of M(1) multiplexers (Figure 22.b). Additionally, the data-select inputs of the multiplexers are restricted to be input variables. Nevertheless, better results have been obtained than previous presented algorithms which take advantage of the *act2* macrocell possibilities.

In this Chapter the concept of the *local* T-transform has been applied to the synthesis of multiplexer circuits for incompletely specified multi-output Boolean functions. The program has been implemented based on this concept. The results show that that the multiplexer synthesis with following mapping to the *ACT1* family needs usually less macrocells and is generally faster than the conventional multi-level minimization followed mapping [52,57,59,60,130-132]. The obtained results can be even further improved by using a more sophisticated mapping algorithm. The obtained multiplexer circuit can be easily converted to a mixed EXOR multiplexer circuit, where multiplexers as in Figure 31 are replaced by an EXOR. A possible extension is to expand the mapping algorithm to the *ACT2* family. To obtain a significantly improved execution time and smaller memory requirement the algorithm can be implemented using OBDDs.

CHAPTER VIII

MULTIPLE-VALUED INPUT GENERALIZED REED-MULLER FORMS

The concept of a fixed polarity Generalized Reed-Muller (GRM) form [20,135] of an n -input Boolean function has been studied extensively in the literature. Recently, there has been an increased interest in canonic forms over the Galois Field(2) [23-26,29,43,75,92]. One of the reasons to study such forms is that for each of the 2^n polarities they are canonical, which has several applications in both theory and practice. Particularly, there is an interest in finding a canonic form of a logic function for which the circuit realization is close in cost to the minimal Exclusive-Sum-of-Product (ESOP) expansion but requires less computation than finding the minimal ESOP form. They have been intensively studied for the better understanding of canonical representations of switching functions [20,24,84]. As it is well known, the circuits corresponding to Reed-Muller and GRM forms have excellent design-for-test properties [32-37]. Finally, GRMs have applications in signal coding and image processing [136].

In recent years a logic with multiple-valued inputs has been introduced with applications in the synthesis for PLAs with decoders, function generators [2,83,137], multi-level logic synthesis, and factorization [5,138]. Recently, the concept of multiple-valued input ESOP expressions [83,139] has been introduced.

This chapter introduces the counterpart of GRMs for the logic with multiple-valued inputs. Such forms will be called Multiple-Valued Input Generalized Reed-Muller Forms (MIGRMs). The counterpart to RMs the Restricted Multiple-Valued Input Generalized Reed-Muller forms (RMIGRMs) has been introduced in [92].

A motivation for the investigation of such kind of forms is that the concept of the AND-EXOR PLA [39] which is used for the realization of ESOPs, can be extended for MIGRMs. When a GRM is realized in an AND-EXOR PLA, each input is either negated or not, but cannot be in both forms. This decreases the number of columns in the AND plane by 50%. The AND-EXOR PLA with two-input, four-output input decoders [39] which is used for the Multiple-Valued Input, Binary Output Exclusive Sum of Products (MIESOP) expansions can be used for the MIGRMs as well. In the case of RMIGRMs [92] decoders with only three outputs can replace the four-output decoders, which decreases the number of columns in the AND plane by 25% with respect to the AND-EXOR PLA from [39]. However, the number of terms is usually larger in a GRM and a MIGRM than in an ESOP or multiple-valued input ESOP of the same function, respectively. But the GRM and MIGRM have the advantage of excellent testability properties which have been proven for the strict Reed-Muller forms [32] and are extensible for the GRMs and MIGRMs

The existence of new Programmable Devices such as the Xilinx LCA 3000, the 1020 series from Actel, or the LHS501 from Signetics allow the direct implementation of the forms introduced here. For instance in the Xilinx LCA 3000 devices every module realization of a Boolean function of five variables has the same cost and speed. Using EXORs is then reasonable, since they are more powerful than the inclusive gates. It has been proven that the circuits with EXOR gates have lower worst-case complexity than the circuits which use only inclusive gates [137]. Moreover, when an EXOR based circuit such as a MIGRM, GRM or an ESOP is smaller than a SOP of a function, then it should be taken for the implementation in PGAs since it has much better testability properties.

The general concept of a *tabular pattern matching* method to calculate the MIGRM is introduced. A similar method has been used for other spectral transforma-

tions of Boolean functions [92,140]. Chapter VIII.1 presents the theory of the MIGRM form. Chapter VIII.2 gives the algorithm for the MIGRM transform. Finally, Chapter VIII.3 illustrates the transformation of the multiple-valued input, multi-output SOP (sum of products) form of the 2 bit adder to a MIGRM form.

VIII.1 CANONICAL MULTIPLE-VALUED INPUT, BINARY OUTPUT GENERALIZED REED-MULLER FORMS

Canonical forms for multiple-valued input, multiple-valued output functions were already proposed [85,141]. The m-Reed-Muller canonical (m-RMC) forms [85] are obtained from the truth vector or the SOP representation and the generalization of the Boolean difference to multiple-valued logic. The approach presented here to generate a multiple-valued input, binary output GRM expansion, makes use of spectral methods similar to the ones introduced in [92,140].

First the concept of polarity for a multiple-valued literal is introduced. Then it is shown how it is further applied for a product term of multiple-valued literals.

VIII.1.1 The Concept of the Polarity for a Multiple-Valued Variable

The concept of the polarity of a singular multiple-valued literal is given by the following Theorem.

Theorem VIII.1: Multiple-valued literal $X_i^{S_i}$, where $S_i \subseteq R_i [0, 1, \dots, p_i-1]$ and $X_i^{S_i} \subseteq R_i$, can be represented by p_i polarity literals $P_i^{T_i}$ with the set of truth values $T_i \subseteq R_i$. The values of the p_i polarity literals form the row vectors of the $p_i \times p_i$ polarity matrix P_i .

Proof: A property of an orthogonal $m \times m$ matrix O (m is an arbitrary natural number) with the elements $o_{ij} \in (0,1)$ is, that any vector $U(u_0, u_1, \dots, u_{m-1})$ with $u_j \in (0,1)$ can be represented by a superposition (that is, performing of a bit-by-bit EXOR operation) of row vectors O_i of the matrix O . Thus, any set S of truth values $S \subseteq R = \{0,1,\dots,p-1\}$ of an

literal X^S can be represented by a superposition of the values from the orthogonal set of p truth values $T \subseteq R = \{0, 1, \dots, p-1\}$, where T^r is the r^{th} row vector of the orthogonal $p \times p$ matrix P .

A form according to Theorem VIII.1 is canonic because the polarity matrix P is orthogonal.

Example VIII.1: All possible sets of truth values $S \subseteq R = \{0,1,2\}$ of a three-valued literal X^S are calculated from the chosen 3×3 orthogonal matrix shown in Figure 32 to illustrate Theorem VIII.1.

$$\text{PM} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}$$

Figure 32. Example of a 3×3 orthogonal matrix.

The following Table VIII gives all possible sets of truth values S obtained by exoring the rows of the orthogonal matrix P (Figure 32), where the rows are denoted T_1, T_2 and T_3 .

TABLE VIII
ALL POSSIBLE SUPERPOSITIONS OF
THE POLARITY LITERALS

truth values S	binary code	superposition
$\{2\}$	001	$T_1 \oplus T_2$
$\{1\}$	010	T_2
$\{1,2\}$	011	T_1
$\{0\}$	100	T_3
$\{0,2\}$	101	$T_1 \oplus T_2 \oplus T_3$
$\{0,1\}$	110	$T_2 \oplus T_3$
$\{0,1,2\}$	111	$T_1 \oplus T_3$

Definition VIII.1: The matrix P_i introduced in Theorem VIII.1 is called the *polarity matrix* or for short *polarity* of a multiple-valued variable X_i . The r^{th} row vector T_i^r of the

matrix P_i is the binary representation of the *polarity literals* $P_i^{T_i}$ (P_i^r for short).

For the representation of the polarity literals P_i^r non-orthogonal matrices can also be used. Thus, Theorem VIII.1 can be generalized to the following Lemma.

Lemma VIII.1: Instead of the orthogonal matrix P defined in Theorem VIII.1 any set of vectors T^r can be used for the polarity literals P^r , if by exoring of those literals every possible set $S \subseteq R = \{0,1,...,p-1\}$ of a variable X^S can be generated. Thus, there is more than one way to create an mv-literal X_i^S out of the polarity literals given by the non-orthogonal matrix. AND-EXOR expressions created with such polarity literals are no longer canonical forms.

The multiple-valued ESOP expressions [39,139] are examples of expressions generated by polarity literals as described in Lemma VIII.1.

Table IX summarizes the notation presented in Definition VIII.1 and Theorem VIII.1:

TABLE IX
THE NOTATION FOR THE MIGRM

mv-literal	$X_1^{S_1}$	$X_2^{S_2} \dots$	$X_i^{S_i} \dots$	$X_n^{S_n}$
set of truth values	$R_1=(0,1,...,p_1-1)$		$R_i=(0,1,...,p_i-1)$	$R_n=(0,1,...,p_n-1)$
polarity literals	$P_1^1 \dots P_1^r \dots P_1^{p_1}$	$P_n^1 \dots P_n^r \dots P_n^{p_n}$
polarity matrix	P_1		P_i	P_n

The first row of Table IX shows the multiple-valued literals $X_i^{S_i}$ of a function $F(X_1, X_2, \dots, X_n)$. The second row gives the sets of truth values P_i for those literals. Next the corresponding polarity literals P_i^r are shown, where the r stands for the values represented by the T_i^r vector of the matrix P_i . Finally, Figure 33 illustrates two polarity matrices P_i consisting of the value vectors T_i^r .

$$P_1 = \begin{bmatrix} T_1^1 \\ \dots \\ T_1^r \\ \dots \\ T_1^{p_n} \end{bmatrix} \quad \dots \quad P_n = \begin{bmatrix} T_n^1 \\ \dots \\ T_n^r \\ \dots \\ T_n^{p_n} \end{bmatrix}$$

Figure 33. Description of polarity matrices.

VIII.1.2 The MIGRM Forms

Before the MIGRM forms will be formally defined let us consider a simple example of such a form.

Example VIII.2: The two variable mv-function $F(X_1, X_2) = X_1^{023} X_2^{01}$, where X_1 is four-valued and X_2 is three-valued, is used to show how to calculate the MIGRM form for the polarity given in Figure 34. The variable X_2^{01} can be represented by the superposition of the polarity literals $P_2^1 \oplus P_2^3$, where for P_2^1 the subscript 2 indicates the corresponding mv-literal X_2 and the superscript 1 denotes the values T_2^1 for the mv-literal X_2 .

$$A_1 = \begin{bmatrix} 1111 \\ 0101 \\ 0011 \\ 0111 \end{bmatrix} = \begin{bmatrix} T_1^1 \\ T_1^2 \\ T_1^3 \\ T_1^4 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 111 \\ 100 \\ 001 \end{bmatrix} = \begin{bmatrix} T_2^1 \\ T_2^2 \\ T_2^3 \end{bmatrix}$$

Figure 34. Polarity matrices.

The natural method to perform the transformation seems to be an exor-term multiplication:

$$\begin{aligned} X_1^{023} X_2^{01} &= (P_1^1 \oplus P_1^3 \oplus P_1^4) (P_2^1 \oplus P_2^3) \\ &= (1 \oplus P_1^3 \oplus P_1^4) (1 \oplus P_2^3) \\ &= 1 \oplus P_1^3 \oplus P_1^4 \oplus P_2^3 \oplus P_1^3 P_2^3 \oplus P_1^4 P_2^3 \end{aligned}$$

In Example VIII.2 the multiplication is applied to obtain the MIGRM form from the polarity literals. To perform the multiplication of the polarity literals shown there, the MIGRM form will be described as a spectrum M . A *tabular pattern matching* method (see Chapter VIII.2) between the indices of the spectral coefficients and a multiple-valued term will be introduced to calculate the final MIGRM from the polarity representation of the literals, $X_i^{S_i}$.

Next the definition of the Generalized Reed-Muller form for Boolean functions is extended to the MIGRM form.

Definition VIII.2: The Multiple-valued Input, binary output Generalized Reed-Muller (MIGRM) form of a single output function $F(X_1, X_2, \dots, X_n)$, where $X_i, i=1, 2, \dots, n$ are the mv-literals, is defined as follows:

$$\begin{aligned} F(X_1, X_2, \dots, X_n) &= a_0 P_1^1 P_2^1 \dots P_n^1 \oplus a_1 P_1^1 P_2^1 \dots P_n^2 \oplus \dots \\ &\dots \oplus a_{p_n} P_1^1 P_2^1 \dots P_n^{p_n} \oplus \dots \oplus a_t P_1^{p_1} P_2^{p_2} \dots P_n^{p_n} \end{aligned} \quad (\text{VIII.1})$$

where, $a_i \in \{0, 1\}$ and $t = \prod_{i=0}^{n-1} p_i$, the other notation follows the one in Table IX.

For a set of functions $F_j(X_1, X_2, \dots, X_n)$ we obtain

$$\begin{aligned} F_j(X_1, X_2, \dots, X_n) &= a_{0,j} P_1^1 P_2^1 \dots P_n^1 \oplus a_{1,j} P_1^1 P_2^1 \dots P_n^2 \oplus \dots \\ &\dots \oplus a_{p_n,j} P_1^1 P_2^1 \dots P_n^{p_n} \oplus \dots \oplus a_{t,j} P_1^{p_1} P_2^{p_2} \dots P_n^{p_n} \end{aligned} \quad (\text{VIII.2})$$

Definition VIII.3: The *polarity* of a MIGRM form is the vector of polarity matrices describing the polarity for each multiple valued literal in the form.

The above definitions can be described with the terminology of spectral techniques as shown [75] for the GRM form.

Definition VIII.4: The MIGRM given by Definition VIII.2 can be represented by the spectrum M , where the index of a spectral coefficient $M_{P_1^r \dots P_n^r}$ corresponds to the terms of polarity literals $P_1^r \dots P_n^r$ in Equation (VIII.1) and (VIII.2). Those terms represent the *standard trivial functions* [140] of the MIGRM spectrum.

Example VIII.3: The MIGRM of the function $F(X_1, X_2) = X_1^{023} X_2^{01}$ (see Example VIII.2) can be represented by its spectrum M . Figure 35 gives the standard trivial functions of the above product term for the polarity from Figure 34. For a comparison of the product $X_1^{023} X_2^{01}$ with the standard trivial functions in Figure 35, the map of this product is shown in Figure 36.

The new expression now can be represented in the form of spectral coefficients (see Table X), where the indices correspond to the standard trivial functions. The same result as in Example VIII.2 has been obtained:

$$X_1^{023} X_2^{01} = 1 \oplus P_1^3 \oplus P_1^4 \oplus P_2^3 \oplus P_1^3 P_2^3 \oplus P_1^4 P_2^3.$$

The reader may wish to verify this form by exoring the corresponding maps from Figure 35 to obtain the map from Figure 36. The algorithm to calculate this form will be given in Chapter VIII.2.

The first rows of Table X give all possible spectral coefficients, where the MIGRM terms represent all the indices for a general function $F(X_1, X_2)$ with a four-valued literal X_1 and a three-valued literal X_2 . In the second rows a '1' indicates that the term represented by the index of the spectral coefficient in the same column is present in the MIGRM form. In Table X the terms $M_{P_1^1 P_2^1}$, $M_{P_1^1 P_2^3}$, $M_{P_1^3 P_2^1}$, and $M_{P_1^4 P_2^1}$ are identical to 1, P_2^3 , P_1^3 , and P_1^4 because $P_1^1 = P_2^1 = 1$.

The final MIGRM form is now obtained by the substitution of the polarity literals for their multiple-valued literal representation according to the polarity matrices. Thus,

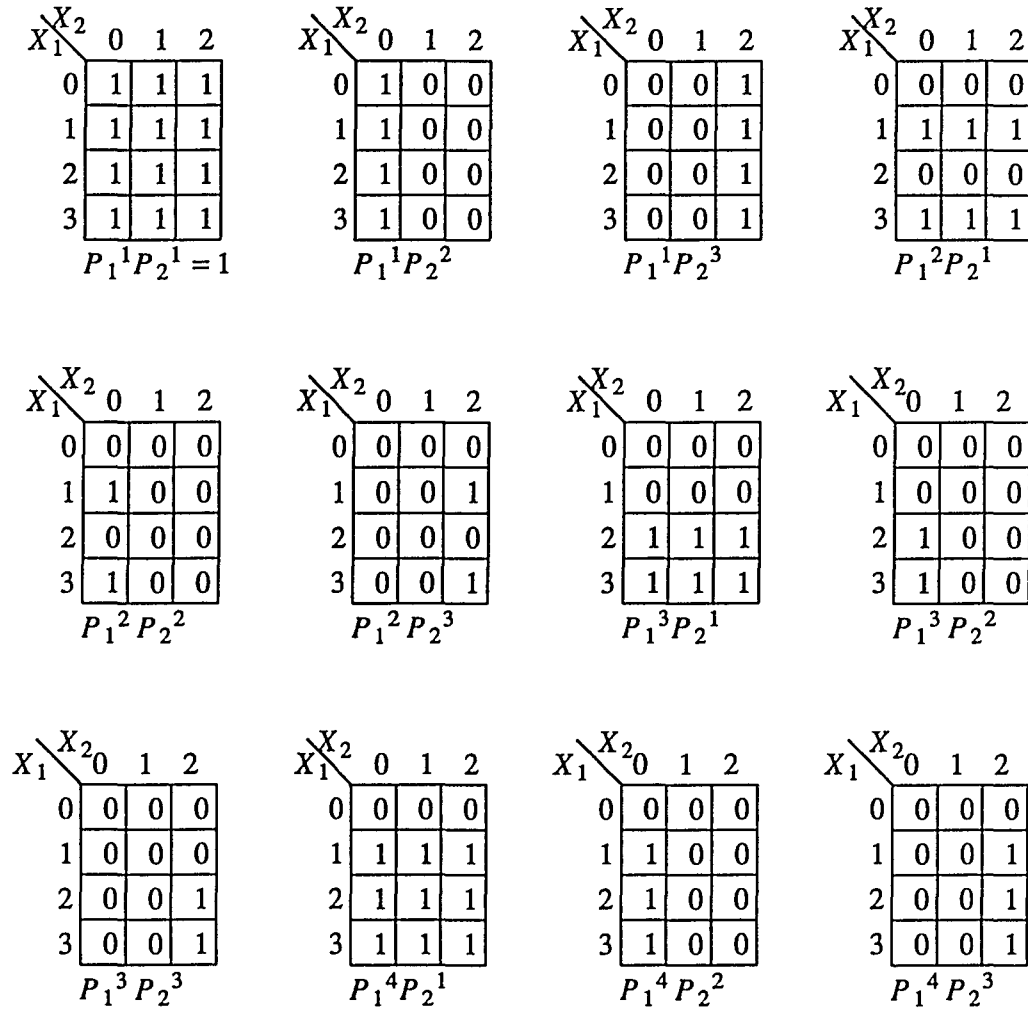


Figure 35. Standard trivial functions for the polarities of variables X_1 and X_2 specified in Figure 34.

$$\begin{aligned}
 X_1^{023} X_2^{01} &= 1 \oplus P_2^3 \oplus P_1^3 \oplus P_1^3 P_2^3 \oplus P_1^4 \oplus P_1^4 P_2^3 \\
 &= 1 \oplus X_2^2 \oplus X_1^{23} \oplus X_1^{23} X_2^2 \oplus X_1^{123} \oplus X_1^{123} X_2^2
 \end{aligned}$$

The transformation of multi-output functions consisting of more than one product term is illustrated with Example VIII.4. The transformation is based on the input function being in the ESOP form or disjoint SOP form.

Example VIII.4: The MIGRM form of the set of two functions: $F_1(X_1, X_2) = X_1^{023} X_2^{01}$, and $F_2(X_1, X_2) = X_1^{023} X_2^{01} + X_1^0 X_2^2 = X_1^{023} X_2^{01} \oplus X_1^0 X_2^2$ for the

$X_1 \backslash X_2$	0	1	2
0	1	1	0
1	0	0	0
2	1	1	0
3	1	1	0

Figure 36. The map of function $F(X_1, X_2) = X_1^{023} X_2^{01}$ from Examples VIII.2 and VIII.3.

TABLE X
THE SPECTRUM OF THE FUNCTION $F(X_1, X_2)$

coefficient	$M_{P_1^1 P_2^1}$	$M_{P_1^1 P_2^2}$	$M_{P_1^1 P_2^3}$	$M_{P_1^2 P_2^1}$	$M_{P_1^2 P_2^2}$	$M_{P_1^2 P_2^3}$
value	1	0	1	0	0	0

coefficient	$M_{P_1^3 P_2^1}$	$M_{P_1^3 P_2^2}$	$M_{P_1^3 P_2^3}$	$M_{P_1^4 P_2^1}$	$M_{P_1^4 P_2^2}$	$M_{P_1^4 P_2^3}$
value	1	0	1	1	0	1

polarity used in Examples VIII.1, and VIII.2 is

$$F_1 = X_1^{023} X_2^{01} = 1 \oplus P_2^3 \oplus P_1^3 \oplus P_1^3 P_2^3 \oplus P_1^4 \oplus P_1^4 P_2^3$$

as calculated in Example VIII.3, where the map of F_1 is shown in Figure 37. The second function F_2 is composed of the following polarity terms:

$$F_2 = X_1^{023} X_2^{01} \oplus X_1^0 X_2^2 = 1 \oplus P_1^3 \oplus P_1^3 P_2^3 \oplus P_1^4$$

For a comparison of the function F_2 with the standard trivial functions for the polarities of variables X_1 and X_2 shown in Figure 36, the map of F_2 is given in Figure 37. The notation in Table X is used to apply the spectral techniques as in Example VIII.4. The output functions F_1 , and F_2 are represented by an output term $f_1 f_2$ for each product term. The output term $f_1 f_2$ is given in the right column of Table VII. Now the spectrum for each term is calculated separately, similar to Example VIII.4. The output term for the

		X_2		
		0	1	2
X_1	0	1	1	1
	1	0	0	0
	2	1	1	0
	3	1	1	0

Figure 37. The map of function $F_2(X_1, X_2)$

TABLE XI
REPRESENTATION OF THE MULTI-OUTPUT
FUNCTION $F(X_1, X_2)$

product term	f_1, f_2
$X_1^{023} X_2^{01}$	11
$X_1^0 X_2^2$	01

product term given in Table XI is taken instead of using '1' as an entry in the spectrum table. As it will be shown in Chapter VIII.2.2 the calculation of the spectrum can be performed in one step for all output functions. However, for the ease of explanation we assume that in Table XII the spectrum for each output function is calculated separately. Thus, the spectrum for each term in each output function can be obtained as shown in Example VIII.3, Table X. Because the product term $X_1^{023} X_2^{01}$ which is present in F_1 and F_2 is the same as in Example VIII.3, the results can be taken directly for the spectrum shown in Table XII. The spectrum for the term $X_1^0 X_2^2$ can be calculated analogously. The results are given in Table XII. The first bit of the entries 11 and 10 in the table stands for the output function F_1 and the second one for the output function F_2 .

TABLE XII
THE SPECTRUM OF THE FUNCTION $F(X_1, X_2)$

term	$M_{P_1^1 P_2^1}$	$M_{P_1^1 P_2^2}$	$M_{P_1^1 P_2^3}$	$M_{P_1^2 P_2^1}$	$M_{P_1^2 P_2^2}$	$M_{P_1^2 P_2^3}$
$X_1^{023} X_2^{01}$	11	-	11	-	-	-
$X_1^0 X_2^2$	-	-	01	-	-	-
EXOR	11	-	10	-	-	-
result	1		$P_1^1 P_2^3$			

term	$M_{P_1^3 P_2^1}$	$M_{P_1^3 P_2^2}$	$M_{P_1^3 P_2^3}$	$M_{P_1^4 P_2^1}$	$M_{P_1^4 P_2^2}$	$M_{P_1^4 P_2^3}$
$X_1^{023} X_2^{01}$	11	-	11	11	-	11
$X_1^0 X_2^2$	-	-	-	-	-	01
EXOR	11	-	11	11	-	10
result	P_1^3		$P_1^3 P_2^3$	P_1^4		$P_1^4 P_2^3$

The calculation of the entries in the above table can be performed in two different ways:

- (1) Each product term is compared with the standard trivial function of each spectral coefficient by the *tabular pattern matching* method.
- (2) The spectral coefficients are determined directly from the product term. Thus, the entry '-' in Table XII indicates that no calculation has to be performed for these cells.

The calculation according to (1) is used in our implementation. Both approaches for the calculation of the complete spectrum are further explained in Chapter VIII.2.2. $P_1^1 P_2^1, P_1^1 P_2^3, P_1^3 P_2^1$, and $P_1^4 P_2^1$ are identical to 1, P_2^3, P_1^3 , and P_1^4 because $P_1^1 = P_2^1 = 1$. The row 'EXOR' in Table XII is obtained by exoring the entries (output functions) in every column. Finally the last row gives the polarity literals for the spectral coefficients having nonzero entry in the 'EXOR' row. The polarity terms in the *result* row are the same as obtained for the functions F_1 and F_2 at the beginning of this example.

Now the polarity literals have to be replaced by their multiple-valued literals as shown in

Example VIII.3. Thus, we obtain

$$F_1 = X_1^{023} X_2^{01} = 1 \oplus X_2^2 \oplus X_1^{23} \oplus X_1^{23} X_2^2 \oplus X_1^{123} \oplus X_1^{123} X_2^2$$

as calculated in Example VIII.3 and

$$F_2 = X_1^{023} X_2^{01} + X_1^0 X_2^2 = 1 \oplus X_1^{23} \oplus X_1^{123} \oplus X_1^{23} X_2^2$$

VIII.2 ALGORITHM FOR THE CALCULATION OF THE MIGRM FORM

The method for the generation of the MIGRM form consists of two basic stages. First, described in Chapter VIII.2.1, each multiple-valued literal of the mv-function has to be transformed to a polarity specified by the chosen orthogonal polarity matrix. In the second stage, presented in Chapter VIII.2.2, the terms consisting of transformed literals are used to calculate the final MIGRM form. The code for the transformation of a multiple-valued literal is chosen in such a way that the second stage of the transformation is not dependent on the chosen polarity for the literal. This approach requires the introduction of the concept of *normalized codes*.

VIII.2.1 Transformation for One Multiple-Valued Literal

The basic steps of the algorithm for the transformation of a multiple-valued literal to its representation of polarity literals in the *normalized code* will be illustrated on an example. Table XIII shows the transformations for some possible four-valued literals to the polarity used in the previous examples. The first row gives all the possible combinations of the polarity literals. Let us observe that the first four polarity literals are the rows of matrix P_1 from Example VIII.2. In the first column some possible literals of the variable X are given. The representation by its polarity literals is given in the respective row for each of these literals, where the binary representation for the value has to be calculated by performing the EXOR operation among the code words that are determined by '1' in the table (i.e. $X^0 = P^1 \oplus P^4 = 1 \oplus P^4$, which is denoted by $1111 \oplus 0111 = 1000$).

TABLE XIII
TRANSFORMATIONS FOR SOME
FOUR VALUED LITERALS

literal	P^1	P^2	P^3	P^4	$P^1 \oplus P^2$	$P^1 \oplus P^3$	$P^1 \oplus P^4$	$P^2 \oplus P^3$	$P^2 \oplus P^4$	$P^3 \oplus P^4$
	1111	0101	0011	0111	1010	1100	1000	0110	0010	0100
X^0							1			
X^1										1
X^{01}						1				
X^{12}								1		
X^{012}										
norm. code	1000	0100	0010	0001	1100	1010	1001	0110	0101	0011

literal	$P^1 \oplus P^2 \oplus P^3$	$P^1 \oplus P^2 \oplus P^4$	$P^1 \oplus P^3 \oplus P^4$	$P^2 \oplus P^3 \oplus P^4$	$P^1 \oplus P^2 \oplus P^3 \oplus P^4$
	1001	1101	1011	0001	1110
X^0					
X^1					
X^{01}					
X^{12}					
X^{012}					1
norm. code	1110	1101	1011	0111	1111

The second row is then created as follows. Its entries for columns P^1 , P^2 , and P^3 are the binary representations of the polarity literals from the polarity matrix. All other entries in the second row are created by the bit-by-bit EXOR operation on the P^i literals from the column header. Finally the last row of the table gives the *normalized code* of the spectral coefficients. It represents the combination of the polarity literals P^r . For instance 0111 means that variables P^2 , P^3 , and P^4 are used. This row is just a binary encoding of the first row.

As mentioned above, the code for the MIGRM representation of one literal is created in such a way, that this *normalized code* can be used directly to perform the transformation of the total mv-function. This can be done by using the same

representation of the polarity literals P^1, P^2, \dots for every chosen polarity. The code determines then what polarity literal/s the initial literal is composed of.

Example VIII.5: The literal X^0 with the internal representation 1000 can be represented by $P^1 \oplus P^4$ which has the *normalized code* 1001. The new code representing this combination of polarity literals can be derived by the bit-by-bit OR-operation of the code representation of the polarity literals shown in Table XIV.

TABLE XIV
THE CODE REPRESENTATION FOR THE
POLARITY LITERALS

normalized code	polarity literals P^r
1000	P^1
0100	P^2
0010	P^3
0001	P^4

The *normalized code* has a "1" in its bit representation corresponding to the index of the polarity literal P^r .

The procedure implemented to perform the transformation of a multiple-valued literal to the *normalized code* can be described by Algorithm 7:

Algorithm 7:

- Step 1: Generate all possible EXOR combinations of the polarity literals P^r (the first row of Table XIII) for the later comparison with the original mv-literal.
- Step 2: Compare the binary representation of the mv-literal with the binary representation (row 2, Table XIII) of the EXOR combination (row 1, Table XIII) of Step 1. If these two binary representations are equal then assign to the mv-literal its *normalized code* (shown in the last row of Table XIII).

VIII.2.2 Transformation of a Multiple-Valued Function

After the transformation of each original mv-literal the whole set of multiple-valued terms of the function has to be changed to the MIGRM form. As mentioned in Chapter VIII.1.2, there are two possible approaches to calculate the complete spectrum. Both methods will be illustrated subsequently. The first method, which is used in our implementation, is based on a *tabular pattern matching* method where every product term of the input function is compared with the *normalized code* of the indices of all spectral coefficients. The second method is based on calculating the spectral coefficients directly from the *normalized code* of the product terms. This is shown for completeness in Chapter VIII.2.2.2.

VIII.2.2.1 Calculation by tabular pattern matching. The basic steps of the algorithm for the *tabular pattern matching* will be explained on an example.

Example VIII.6: Let us take any function $G(X_1, X_2, X_3)$ where the literal X_1 being three-valued is represented by three polarity literals P_1^1 , P_1^2 and P_1^3 . The second literal X_2 being four-valued is represented by the polarity literals P_2^1 , P_2^2 , P_2^3 and P_2^4 , and the three-valued literal X_3 is represented by P_3^1 , P_3^2 , and P_3^3 . Table XV shows the spectral coefficients for a spectrum representing a function $G(X_1, X_2, X_3)$.

TABLE XV
THE COMPLETE MIGRM SPECTRUM OF THE FUNCTION
 $G(X_1, X_2, X_3)$

M_l	$M_{P_1^1 P_2^1 P_3^1}$	$M_{P_1^1 P_2^1 P_3^2}$	$M_{P_1^1 P_2^1 P_3^3}$	$M_{P_1^1 P_2^2 P_3^1}$	$M_{P_1^1 P_2^2 P_3^2}$...	$M_{P_1^3 P_2^4 P_3^3}$
code	100 1000 100	100 1000 010	100 1000 001	100 0100 100	100 0100 010	...	001 0001 001

The code shown in Table XV is obtained by generating all possible combinations of polarity literals from the distinct original mv-literals. This means that not more than one polarity literal per original mv-literal can occur in the index of a spectral coefficient (i.e.

$M_{V_1^2 V_1^3}$ can not occur because both polarity literals are from the original mv-literal X_1). As one can observe from the binary representation of the index (second row in Table XV), called *code*, each spectral coefficient consists basically of a combination of three polarity literals, where each polarity literal is taken from a different original mv-literal. Because Table XV has been created for normalized mv-literals, the *normalized code* has to be used for the *code* in Table XV.

The final value of the spectral coefficient M_s , where M_s is any of the possible spectral coefficients, determined by a column in Table XV, is obtained by comparing the *normalized codes* of all product terms, further called $term_x$, of the Boolean function ($x = 0, \dots, m-1$; where m is the total number of terms) including variables X_1, X_2, \dots, X_n with the representation of the spectral coefficients $code_s$ from Table XV. The value M_s for $term_x$ is the representation of the output functions f_x of $term_x$, if the intersection of the $code_s$ and the $term_x$ is not empty. The EXOR operation among all m values M_s has to be performed to obtain the final value M_s for the whole function (all its terms). The calculation of the values of coefficients M_s is described by the following formula:

$$M_s^0 = 0$$

$$M_s^{x+1} = M_s^x \oplus (((code_s \& term_x) \neq \phi) \& f_x), \quad (\text{VIII.3})$$

for $x = 0, \dots, m-1$

where the value of the spectral coefficient is a product term, $code_s$ and $term_x$ are the binary representation of the respective literals, where $((code_s \& term_x) \neq \phi)$ has to be true for the intersection of each literal of $code_s$ and $term_x$. By ϕ we denote a vector which contains at least one zero. The output product term for $term_x$ is denoted by f_x .

The final MIGRM form consists of the terms obtained by replacing the polarity literals in the indices of the spectral coefficients M_s (row code in Table XV) which have a value that is not '0' with their binary representation.

To summarize, the procedure to obtain the MIGRM of a multi-output Boolean

function can be described by Algorithm 8:

Algorithm 8:

- Step 1: Transform all the multiple-valued input literals of the function to their *normalized codes* according to the chosen polarities for the variables (Chapter VIII.2.1, Algorithm 7).
- Step 2: Calculate the MIGRM spectrum for the *normalized codes* as presented in Chapter VIII.2.2.
- Step 3: Replace the polarity literals of non-zero spectral coefficients by their original mv-literal. The output functions for the terms are given by the values of the spectral coefficients.

As one can observe, each spectral coefficient can be calculated in turn. Thus, it can be stored directly on hard disk. The necessity to keep the whole spectrum in the computer memory has been overcome with this approach. This feature can not be used directly by the method described in the sequel.

VIII.2.2.2 Calculation by direct transformation of a product term. A second approach to obtain the MIGRM form is to calculate the spectrum for the product terms represented in their *normalized code* based on the same properties as in the previous described algorithm. In the *tabular pattern matching* method each product term in its *normalized code* is intersected with the index of each spectral coefficient, where the result of the intersection determines the value of the spectral coefficient (Equation (VIII.3)). For the direct transformation of a product term to its MIGRM form, the same property can be applied to calculate the nonzero spectral coefficients directly from the *normalized code*.

First let us observe some properties of Equation (VIII.3). The index of each spectral coefficient has only one '1' in the *normalized code* of each literal. Thus, the resultant product term of the intersection of the index with the *normalized code* of a product term

has non-zero literals only if the *normalized code* of the product term covers each literal of the index.

By reformulating the above property one obtains a procedure to calculate the spectrum directly from the *normalized code* of a product term. This procedure is a direct translation of the multiplication of the polarity literals in Example VIII.2 to a calculation from the *normalized code* representation.

It should be observed that the *normalized code* for a polarity literal contains only a single '1', the *normalized code* for an input literal has at least one '1'. The '1's of the *normalized codes* of the spectral coefficients for a product term are present at the same position in the *normalized code* for the product term. Thus, the *normalized codes* of the spectral coefficients for a product term can be obtained by generating all *normalized codes* with single '1's per literal, where those '1's are present in the same position of the *normalized code* for the product term.

The calculation of the MIGRM form from a SOP form is illustrated in the following section.

VIII.3 A PRACTICAL EXAMPLE

A real life example is included to illustrate the complete MIGRM transformation. The function used is a 2 bit adder (Table XVI, where the two four-valued literals X_1 and X_2 represent two binary values each ($X_1 = (x_1, x_2)$, $X_2 = (x_3, x_4)$)).

The input variable assignment is not unique. Thus, there exist many different assignments. In this example the mv-literal X_1 is obtained by changing the first two bits of the binary function (x_1, x_2) to a four-valued literal ($X_1^0 = 00$, $X_1^1 = 01$, $X_1^2 = 10$, $X_1^3 = 11$). The second two bits are taken to obtain X_2 .

To make it easier to follow the steps of the transformation, the same polarity is assumed for both literals. The polarity literals P_i^f and the binary representations of their

TABLE XVI
THE TRUTH TABLE OF THE
2 BIT ADDER

binary x_1, x_2, x_3, x_4	multiple valued $X_1 \quad X_2$		normalized $X_1 \quad X_2$		output $f_c f_0 f_1$
0000	1000	1000	1110	1110	000
0001	1000	0100	1110	1111	001
0010	1000	0010	1110	0010	010
0011	1000	0001	1110	1011	011
0100	0100	1000	1111	1110	001
0101	0100	0100	1111	1111	010
0110	0100	0010	1111	0010	011
0111	0100	0001	1111	1011	100
1000	0010	1000	0010	1110	010
1001	0010	0100	0010	1111	011
1010	0010	0010	0010	0010	100
1011	0010	0001	0010	1011	101
1100	0001	1000	1011	1110	011
1101	0001	0100	1011	1111	100
1110	0001	0010	1011	0010	101
1111	0001	0001	1011	1011	110

values (T_i^r) are given in Table XVII.

TABLE XVII
THE POLARITY FOR THE 2 BIT ADDER

polarity for X_1	polarity for X_2
$P_1^1 : 1111$	$P_2^1 : 1111$
$P_1^2 : 0101$	$P_2^2 : 0101$
$P_1^3 : 0010$	$P_2^3 : 0010$
$P_1^4 : 1100$	$P_2^4 : 1100$

Now the binary representations of the literals X_1 and X_2 in Table XIII are replaced by their *normalized codes* (Algorithm 7). Because only four different values occur in the literals X_1 and X_2 , the *normalized codes* for those values are shown in Table XVIII,

where $P' = P_1' = P_2'$.

TABLE XVIII
THE NORMALIZED CODES FOR THE
FOUR OCCURRING VALUES

the binary value of a literal	the normalized code	the cube representation of the normalized code
0001	$P^1 \oplus P^3 \oplus P^4$	1011
0010	P^3	0010
0100	$P^1 \oplus P^2 \oplus P^3 \oplus P^4$	1111
1000	$P^1 \oplus P^2 \oplus P^3$	1110

The multiple-valued literals X_1 and X_2 shown in Table XVI are now substituted with the cube representations of their *normalized codes* given in Table XVIII. The result is given in Table XVI.

Then, the *normalized code* obtained in this procedure is compared with all the indices of the spectral coefficients of the general spectrum for a function $G(X_1, X_2)$, where X_1 and X_2 are four-valued literals (Algorithm 8, Step 2). Table XIX illustrates the calculation of the spectrum for the first three terms from Table XVI. If for each term the intersection of the term and the index of the coefficient is not empty (Equation (VIII.3)) the Table has the output functions of the terms as entries. Otherwise the entry is "-". For instance, the cell for the intersection of row 1110 1011 and column 0100 1000 is "011" since the intersection of those indices is 0100 1000 where both literals are not empty. The intersection of row 1110 0010 and column 0100 1000 is 0100 0000 so "-" is placed in the corresponding cell. The final coefficients for the partial function in Table XIX are obtained by exoring the entries of the columns. The last row gives the result of this operation.

The result of the complete function is shown in Table XX. The first column of Table XX lists all non-zero spectral coefficients M_s that occur in any of its component functions. These coefficients have been obtained by comparing (as in Table XIX) the

TABLE XIX
THE SPECTRUM FOR THE FIRST THREE
TERMS FROM TABLE XVI

term	$M_{P_1^1 P_2^1}$	$M_{P_1^1 P_2^2}$	$M_{P_1^1 P_2^3}$	$M_{P_1^1 P_2^4}$	$M_{P_1^2 P_2^1}$	$M_{P_1^2 P_2^2}$	$M_{P_1^2 P_2^3}$	$M_{P_1^2 P_2^4}$
	1000 1000	1000 0100	1000 0010	1000 0001	0100 1000	0100 0100	0100 0010	0100 0001
1110 1111	001	001	001	001	001	001	001	001
1110 0010	-	-	010	-	-	-	-	010
1110 1011	011	-	011	011	011	-	011	011
result	010	001	000	010	010	001	010	000

term	$M_{P_1^3 P_2^1}$	$M_{P_1^3 P_2^2}$	$M_{P_1^3 P_2^3}$	$M_{P_1^3 P_2^4}$	$M_{P_1^4 P_2^1}$	$M_{P_1^4 P_2^2}$	$M_{P_1^4 P_2^3}$	$M_{P_1^4 P_2^4}$
	0010 1000	0010 0100	0010 0010	0010 0001	0001 1000	0001 0100	0001 0010	0001 0001
1110 1111	001	001	001	001	-	-	-	-
1110 0010	-	-	010	-	-	-	-	-
1110 1011	011	-	011	011	-	-	-	-
result	010	001	000	010	000	000	000	000

normalized binary representations of their indices (listed in the second column of Table XX) with the *normalized code* obtained according to Tables XVII and XVIII. Finally, the binary representation for the literals of variables X_1 and X_2 is obtained by replacing the polarity literals of the indices of spectral coefficients from the first row by their binary representations (Algorithm 8, Step 3). According to Table XVII the conversion is: 1000 to 1111, 0100 to 0101, 0010 to 0010, and 0001 to 1100. The output terms $f_c f_0 f_1$ are the values of the spectral coefficients.

The implementation of the above result as an AND-EXOR PLA with 2-input, 3-output decoders for the chosen polarities is shown in Figure 38.

The input decoders are the same because the same polarity has been chosen for both literals, The input decoder for both variables X_1 , and X_2 is shown in Figure 39. The above method was implemented in the program GRM-MV (Generalized Reed-Muller synthesizer, Multiple-Valued version). The calculation time for this expansion using this

TABLE XX
THE MIGRM OF THE 2 BIT ADDER

spectral coefficient	index	X_1	X_2	$f_c f_0 f_1$
$M_{V_1^1 V_2^1}$	1000 1000	1111	1111	100
$M_{V_1^1 V_2^2}$	1000 0100	1111	0101	101
$M_{V_1^1 V_2^4}$	1000 0001	1111	1100	110
$M_{V_1^2 V_2^1}$	0100 1000	0101	1111	101
$M_{V_1^2 V_2^2}$	0100 0100	0101	0101	010
$M_{V_1^2 V_2^3}$	0100 0010	0101	0010	100
$M_{V_1^2 V_2^4}$	0100 0001	0101	1100	100
$M_{V_1^3 V_2^2}$	0010 0100	0010	0101	100
$M_{V_1^4 V_2^1}$	0001 1000	1100	1111	110
$M_{V_1^4 V_2^2}$	0001 0100	1100	0101	100
$M_{V_1^4 V_2^4}$	0001 0001	1100	1100	100

program took a 1/10 of a second on a Sun 3/50.

The extension of the General Reed-Muller expansion to multiple-valued input, binary multi-output functions has been shown. For this, the concept of code normalization of single multiple-valued literals to perform a final transformation has been developed. The code normalization is applied to make the transformation of the complete function independent of the chosen polarity. This simplifies and speeds up the main transformation step to the final MIGRM form for the transformed single mv-literal. For further investigations of the properties of such forms the MIGRM transformation has been implemented as a computer algorithm. Because an exhaustive search of all $n-1$

$\prod_{i=0}^{p(X_i)} p(X_i)$ MIGRM forms, where $p(X_i)$ is the number of possible polarities for variable $i=0$

X_i with $i=1, \dots, n$, would be too time consuming. Further research is necessary to avoid a complete search to find good polarities such as done for GRM forms by Almaini et al [23-25]. Therefore, the next chapter introduces the concept of the Kronecker product to the computation of the MIGRM form. This allows the determination of the minimal

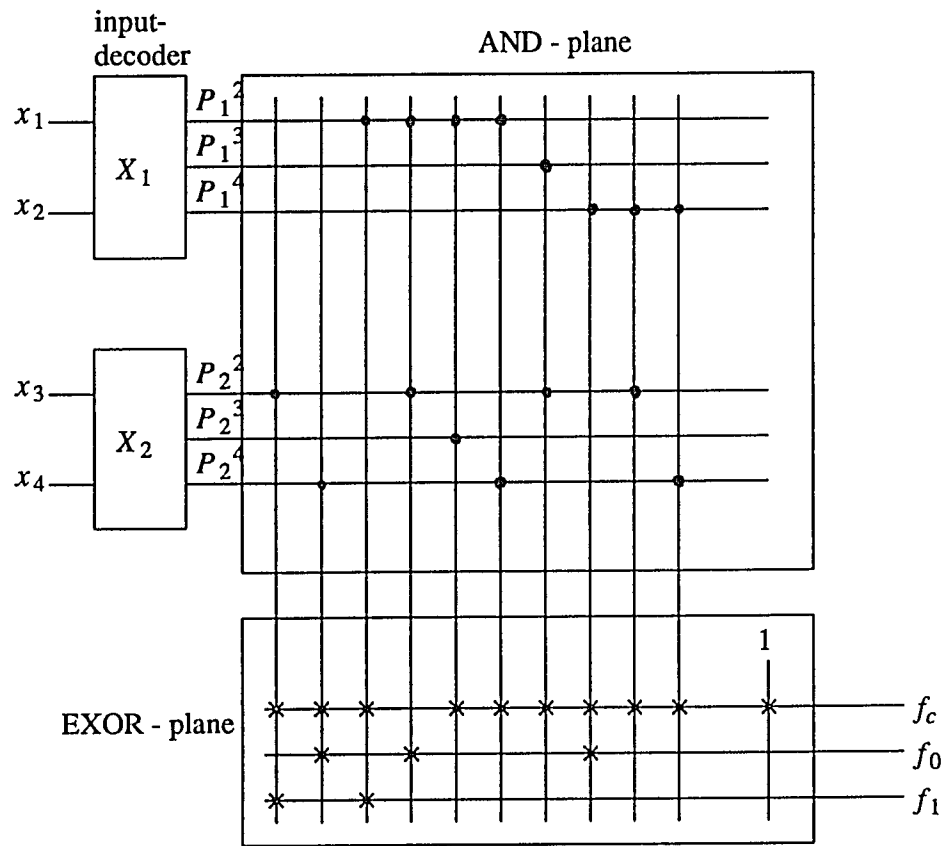


Figure 38. AND-EXOR PLA implementation with input decoders of the MIGRM form of the 2 bit adder.

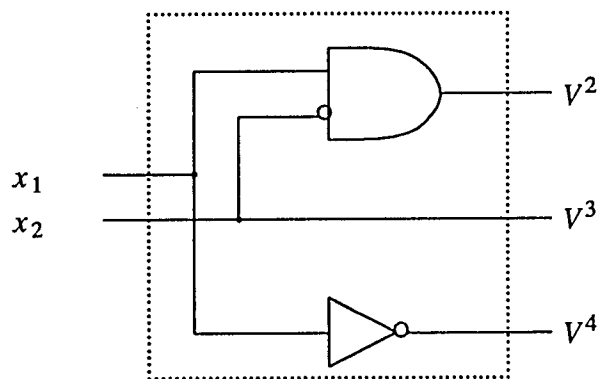


Figure 39. Input decoder for variables X_1 , and X_2 .

MIGRM form without computing all forms. Additionally, the concept of the MIGRM is extended to multiple-valued input, incompletely specified binary output functions.

CHAPTER IX

ON THE MINIMAL MULTIPLE-VALUED INPUT KRONECKER REED-MULLER FORM FOR INCOMPLETELY SPECIFIED MULTIPLE-VALUED INPUT FUNCTIONS

In the previous Chapter a *tabular pattern matching* method has been developed for the calculation of the Multiple-Valued Input Generalized Reed-Muller (MIGRM) form [29,92]. This Chapter adopts the concept of the Kronecker matrix product [23,24] for the calculation of the transform matrix of the MIGRM forms for multiple-valued input logic. Because these forms can be obtained by the Kronecker matrix product they are called Multiple-valued Input Kronecker Reed-Muller (MIKRM) forms. Thus, it is possible to develop Fast Transforms, based on the Kronecker product, for the calculation of the MIKRM form. To overcome the memory requirements for a matrix calculation, we investigate here the calculation of the MIKRM form from a disjoint representation, in particular from an Ordered Multiple-valued Decision Diagram (OMDD) [80]. Up to now the Binary Decision Diagrams (BDD) [79] have been applied in spectral methods only for the case of the Walsh [94] and the Reed-Muller [97] transforms. The advantage of Decision Diagrams is that they require less memory, especially for large functions. Another advantage is the possibility of both the fast tautology verification and the fast intersection of two functions [79,80], in comparison to the cube calculus methods [100]. The concept of the *extended truth vector* e was presented in [23] to obtain the minimal KRM form for completely specified Boolean functions. This concept is generalized here for incompletely specified multiple-valued input, binary multi-output functions (*mv*-functions) and is used to obtain the minimal MIKRM form.

It can be observed that although for Boolean logic the Generalized Reed-Muller forms are a subset of Kronecker Reed-Muller forms, for multiple-valued input logic the MIGRMs and the MIKRM, introduced in two different ways, describe the same family of forms which will be referred to as MIKRM.

The Chapter IX.1 gives a short review of the Boolean KRM forms. Chapter IX.2 introduces the calculation of the transform matrix for the MIKRM form by a Kronecker matrix product. Chapter IX.3 presents the method for finding the forms having the minimal terms. The following section presents the generalization of the developed methods to incompletely specified multi-output *mv* functions. The computation of the MIKRM form from the disjoint representation OMDD is introduced in Chapter IX.5.

IX.1 KRONECKER REED-MULLER FORM

The Kronecker Reed-Muller (KRM) forms for Boolean functions have been presented in [23,24,84]. The Reed-Muller (RM) form, which is included in the KRM forms, for an n variable function can be described by the transform matrix T_n over the Galois Field (2) [84]

$$a = T_n \times d \quad (\text{IX.1})$$

where vector d is the truth vector representation of the Boolean function [65], and a is the coefficient vector of the RM form, the multiplication " \times " is over the Galois Field (2). Both vectors are in the straight binary order [27].

The transform matrix exhibits the recursive structure given in Equation (IX.2)

$$T_n = \begin{bmatrix} T_{n-1} & 0 \\ T_{n-1} & T_{n-1} \end{bmatrix} \quad (\text{IX.2})$$

where $T_0 = [1]$. Equation (IX.2) can be described by the application of the Kronecker matrix product (*) for n times

$$T_n = T_1 * T_1 * \dots * T_1 \quad (\text{IX.3})$$

with

$$T_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad (\text{IX.4})$$

For a Fixed Polarity Reed-Muller (FPRM) form [23,24] each variable can have a different polarity, positive or negative literal (x or \bar{x}), described by the polarity matrices P_0 and P_1 , respectively.

$$P_0 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad P_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad (\text{IX.5})$$

In a KRM form a variable can occur in both, positive and negative form, which results in polarity matrix P_2 .

$$P_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{IX.6})$$

The three polarity matrices P_0 , P_1 and P_2 generate the three possible canonic forms [23,24] of single variable Boolean functions.

For the Reed-Muller transform there exists only one polarity matrix P_0 . Thus, there is only one possible transform matrix T_1 for each variable. However for the KRM form there exist three different polarity matrices. Therefore, there exist three possible transform matrices K^0 , K^1 and K^2 . They are obtained by inverting the three polarity matrices given in Equations (IX.5) and (IX.6) [23]

$$K^0 = P_0^{-1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad K^1 = P_1^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad K^2 = P_2^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{IX.7})$$

For an n variable Boolean function there are 3^n KRM transform matrices [23], which include the 2^n FPRM transform matrices. The transform matrix T_n for a KRM

transform of a Boolean function can be obtained by Equation (IX.8)

$$T_n = K_1 * K_2 * \dots * K_i * \dots * K_n \quad (\text{IX.8})$$

where $K_i \in \{K^0, K^1, K^2\}$ ($K_i \in \{K^0, K^1\}$ for FPRM) denotes the transform matrix for the corresponding binary variable x_i .

The minimal KRM form is of most interest. However, the naive approach of computing all 3^n forms to determine the minimal form is very complex. Thus, special algorithms have been developed based on the *extended truth vector* e [23,84,142].

The next section introduces the Multiple-valued Input Kronecker Reed-Muller (MIKRM) form, which is an extension of the KRM form to *mv* logic. Then, Chapter IX.3 adapts the concept of the *extended truth vector* e to find the minimal MIKRM form.

IX.2 MULTIPLE-VALUED KRONECKER REED-MULLER FORM

The extension of the Kronecker Reed-Muller forms to *mv* logic is based on the concept of polarity matrices introduced in the previous Chapter by Theorem VIII.1. The MIKRM is defined identically to the MIGRM by Definition VIII.2.

The *tabular pattern matching* method was developed [92] to perform the product multiplication necessary to obtain the MIKRM form. The method has been based on the polarity matrices P_i for the variables X_i . The relation of the polarity matrices P_i for the variables X_i to the polarity matrix P of the complete function follows directly from Equation (VIII.2).

Property IX.1: The polarity matrix P describing the *polarity* of an *mv*-function $f(X_1, X_2, \dots, X_n)$ is given by

$$P = P_1 * \dots * P_i * \dots * P_n \quad (\text{IX.9})$$

Theorem IX.1: The truth vector representation d of an *mv* function defined by Equation (VIII.2) can be expressed by the Galois Field (2) matrix multiplication

$$d = P \times a \quad (\text{IX.10})$$

where P is the polarity matrix [92] for the mv function.

Proof: As shown in [92] the form given by Equation (VIII.2) is canonic. The Kronecker product has the property that it preserves the orthogonality. Therefore, because each matrix P_i , describing the *polarity* for a variable X_i , is orthogonal, the matrix P obtained by Equation (IX.10) is also orthogonal. Thus, the MIGRM forms can be obtained by the matrix multiplication given by Equation (IX.10). \square

For the calculation of the coefficient vector a of the MIGRM form for an n variable mv function we obtain

$$a = P^{-1} \times d = T_n \times d \quad (\text{IX.11})$$

where T_n is the inverse of polarity matrix P over Galois Field (2). Because matrix P is orthogonal, T_n always exists. It follows from the properties of the Kronecker product and Equation (IX.11), that the transform matrix $T_1 = K_i$ for a single-variable mv function is obtained by inverting the polarity matrix P_i over the Galois Field (2).

$$T_1 = K_i = P_i^{-1} \quad (\text{IX.12})$$

The inversion of a matrix over the Galois Field (2) is illustrated in Example IX.1.

Example IX.1: Let us assume the following polarity matrix P for a single variable mv function, which represents one possible orthogonal representation for a three-valued variable.

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

The first column of P represents the value X^0 , the second X^1 and the third X^2 , while the rows correspond to the literals. Therefore, polarity matrix P consists of the three polarity

literals X^{012} , X^0 and X^2 . Any literal of a three-valued variable can be obtained by an EXOR combination of a subset of polarity literals. To obtain the inverted matrix $K = P^{-1}$ the single-valued literals of the variable X have to be expressed as the EXOR function of the polarity literals:

$$\begin{array}{rcl} X^0 & = & \\ X^1 & = & X^{012} \oplus X^0 \oplus X^2 \\ X^2 & = & \end{array}$$

The polarity literals necessary to obtain the single-valued literals X^0 , X^1 , and X^2 determine the column vectors of the transform matrix K . The literal X^0 is composed of the second polarity literal. Thus, the second component of the first column vector of the matrix K is 1. Similarly, the literal X^1 is composed of all three polarity literals. Thus, the elements of the second column vector are 1. The third column vector is obtained analogously.

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

Property IX.2: The MIKRM form for an mv function can be calculated by the Kronecker matrix product given in Equation (IX.13)

$$T_n = K_1 * K_2 * \dots * K_i * \dots * K_n \quad (\text{IX.13})$$

where K_i is one of the possible transform matrices for variable X_i . For an mv function having only two-valued variables the possible orthogonal transform matrices K_i are the same as in Equation (IX.7). Thus, the KRM forms are special cases of the MIKRM forms.

Example IX.2: The computation of an MIKRM form is illustrated on the function $F(X_1, X_2) = X_1^{023} X_2^{01}$, given in Figure 40 in its truth vector representation, and the polarity matrices P_1, P_2

$$P_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} P_1^1 \\ P_1^2 \\ P_1^3 \\ P_1^4 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} P_2^1 \\ P_2^2 \\ P_2^3 \end{bmatrix}$$

where $P_2 = P_i$ from Example IX.1 and P_i^k describes the k^{th} polarity literal, represented by the k^{th} row of the polarity matrix P_i , for the i^{th} variable. The transform matrices K_1 and K_2 are obtained by inverting the polarity matrices P_1 and P_2 over the Galois Field (2).

$$K_1 = P_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad K_2 = P_2^{-1} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Now the transform matrix T_2 can be calculated.

$$T_2 = K_1 * K_2 = \begin{bmatrix} K_2 & 0 & 0 & 0 \\ 0 & 0 & K_2 & K_2 \\ 0 & K_2 & 0 & K_2 \\ K_2 & K_2 & K_2 & K_2 \end{bmatrix}$$

Next the MIKRM coefficients for the function $f(X_1, X_2)$ are obtained by the multiplication of the transform matrix T_2 and the function vector d , shown in Figure 40. The product terms $P_i^k P_j^l$, are obtained by the intersection of the respective function of polarity literal P_i^k with the function of polarity literal P_j^l , where the ordering follows from the Kronecker matrix product of K_1 with K_2 . The values of the coefficients a_i in Equation (IX.12), corresponding to the terms $P_i^k P_j^l$ in Figure 40, are substituted by the respective polarity literal to obtain the final MIKRM form.

In general one is interested in the MIKRM form with the least number of product terms. Therefore, the next section adopts the concept of the *extended truth vector* e for

$$a = T \times d = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} V_1^1 V_2^1 \\ V_1^1 V_2^2 \\ V_1^1 V_2^3 \\ V_1^2 V_2^1 \\ V_1^2 V_2^2 \\ V_1^2 V_2^3 \\ V_1^3 V_2^1 \\ V_1^3 V_2^2 \\ V_1^3 V_2^3 \\ V_1^4 V_2^1 \\ V_1^4 V_2^2 \\ V_1^4 V_2^3 \end{bmatrix}$$

Figure 40. Transform of the function $F(X_1, X_2) = X_1^{023} X_2^{01}$

$$\begin{aligned} f(X_1, X_2) &= P_1^1 P_2^1 \oplus P_1^1 P_2^3 \oplus P_1^3 P_2^1 \oplus P_1^3 P_2^3 \oplus P_1^4 P_2^1 \oplus P_1^4 P_2^3 \\ &= 1 \oplus X_2^2 \oplus X_1^{23} \oplus X_1^{23} X_2^2 \oplus X_1^{123} \oplus X_1^{123} X_2^2 \end{aligned}$$

obtaining the minimal KRM forms [23].

IX.3 EXTENDED TRUTH VECTOR

The concept of the *extended truth vector* e has been introduced to obtain the minimal KRM form [23]. As for Boolean functions, the generalization of the *extended truth vector* e can be defined for *mv* logic.

Definition IX.1: The *extended truth vector* e for *mv* logic consists of all possible coefficients a_i of all MIKRM forms, where the number of coefficients a_i is given by

$$\prod_{i=1}^n (2^{p_i} - 1) \quad (\text{IX.14})$$

where p_i is the number of values of the i^{th} variable.

In the case of a Boolean function one obtains the 3^n components. An approach to determine the KRM forms with the minimal number of product terms by the calculation of the *extended weight vector* w has been introduced in [23,24]. We will investigate here a similar approach to obtain the minimal MIKRM forms.

Definition IX.2: The *extended truth vector* e for mv logic is given by

$$e = M_n \times d \quad (\text{IX.15})$$

where \times denotes the matrix multiplication over Galois Field (2).

To obtain all coefficients a_i of all MIKRM forms, the transform matrix M_n has to consist of all the different row vectors of the transform matrices T_n for all MIKRM forms. For an n -variable logic function with all variables having the same number of values, the matrix M_n is described by Equation (IX.16)

$$M_n = M_{n-1} * M_1 \quad (\text{IX.16})$$

Otherwise, the matrix M_n is obtained by the Kronecker product of the transform matrices M^i , where the matrix M_p is determined by the number of values of the variable X_i .

$$M_n = M^1 * M^2 * \dots * M^n \quad (\text{IX.17})$$

Example IX.3: The *extended truth vector* e for a single variable function X_i consists of all possible product terms of single-value literals of X_i . It can also be described by the row vectors of the matrix M^i in the straight binary order, $r = 1, 2, \dots, 2^p - 1$, where p is the number of values of the respective multiple-valued variable. Thus, the transform matrix M^1 for a three-valued variable is given by

$$M^1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} X^2 \\ X^1 \\ X^{12} \\ X^0 \\ X^{02} \\ X^{01} \\ X^{012} \end{bmatrix} \quad (\text{IX.18})$$

The transform matrix M^1 obtained is related to the transform matrices $T_1 = K_i$ as follows: all possible row vectors in any transform matrix K_i exist in the set of row vectors of M^1 .

The next step is to calculate the number of product terms for an mv function in each MIKRM form. The transform matrix M_n of an mv function calculated according to Equation (IX.17) is composed of all row vectors that occur in the transform matrices T_n describing the different MIKRM forms. Thus, the number of terms for a certain MIKRM form is given by the summation of the respective components of the *extended truth vector* e .

Definition IX.3: The *extended weight vector* is defined as

$$w = W_n \times e \quad (\text{IX.19})$$

where \times denotes the integer matrix multiplication, and each row vector $W_n[i]$ of the transform matrix W_n is determined by the incidences of the row vectors of the transform matrix T_n in the transform matrix M_n . Therefore, the matrix W_n is called the *incidence* or *weight matrix* [23].

For the case that all variables have the same number of values, W_n can be calculated by Equation (IX.20).

$$W_n = W_{n-1} * W_1 \quad (\text{IX.20})$$

If the mv variables X_i have different numbers of values, W_n has the following property

$$W_n = W^1 * W^2 * \dots * W^n \quad (\text{IX.21})$$

where W^i is the *weight matrix* for the i^{th} variable. The calculation of the row vectors $W_n[i]$ of the weight matrix W_n is illustrated in Example IX.4.

Example IX.4: There are 28 polarities for a three-valued variable. Therefore, only the calculation of one row vector is illustrated for the weight matrix $W_1 = W^1$. The chosen row vector is determined by the polarity matrix P_1 from Example IX.1.

$$P_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad K_1 = P_1^{-1} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

The row vectors of the matrix K_1 describing the MIKRM form for the polarity matrix P_1 determine incidence vector $W_1[1]$ of the transform matrix M^1 . As one can observe, the first row vector of the matrix K_1 is the same as the second in matrix M^1 , Equation (IX.19). The second row in matrix K_1 is the same as the sixth row in matrix M^1 and the third row in matrix K_1 is the same as the third row in M^1 . The identical rows are marked with a '*' in matrix M^1 . Thus, the incidence row vector $W_1[1]$ for the polarity described by matrix $P_1 = K_1^{-1}$ is given by the transposed column vector of symbols '*':

$$M^1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 & * \\ 0 & 1 & 1 & * \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 & * \\ 1 & 1 & 1 \end{bmatrix} \rightarrow [0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0] = W_1[1]$$

The *extended weight vector* w gives the number of necessary terms for the different MIKRM forms. Thus, the minimal MIKRM form can be obtained by calculating the *extended weight vector* w , where a row vector $W_n[i]$ of the *weight matrix* W_n describes the incidences of a particular MIKRM form. The following example illustrates the steps

to find the minimal MIKRM form.

Example IX.5: First the *extended truth vector* e has to be computed by Equation (IX.15) to find the minimal MIKRM form for the mv function $f(X) = X^{02}$, where variable X is three-valued.

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

where \times denotes the Galois Field (2) matrix multiplication. Next the *extended weight vector* w has to be calculated. Only the calculation of w for a subset of four MIKRM forms is illustrated because there are 28 polarities for a three-valued variable. The first row of the *weight matrix* $W_1 = W^1$ is the vector obtained for the transform matrix K_1 as calculated in Example IX.4,

$$w = W_1 \times e = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 2 \end{bmatrix}$$

where \times denotes the integer matrix multiplication. As one can observe, two of the four selected MIKRM forms, the second and the third row of the *weight matrix* W_1 , consist of a single product term. Thus, they are the two minimal MIKRM forms among the four chosen MIKRM forms. The transform matrix K_1 for the MIKRM form determined by the second incidence vector of the *weight matrix* W_1 is described by the respective row

vectors of the transform matrix M^1 .

$$K_1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Then the respective polarity of variable X_1 to be selected to obtain the minimal MIKRM form is given by

$$P_1 = K_1^{-1} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The introduced methods based on the *extended weight vector* w allow to compute the minimal MIKRM form for a given *mv* function without calculating all MIKRM forms.

In the next section the method introduced above to compute the minimal MIKRM form for a single output, completely specified *mv* function is generalized to multi-output incompletely specified *mv* functions.

IX.4 MINIMAL MIKRM FORM FOR MULTI-OUTPUT INCOMPLETELY SPECIFIED *MV* FUNCTIONS

IX.4.1 Extension for incompletely specified *mv* functions

The problem to obtain the minimal MIKRM form for an incompletely specified *mv* function is to assign the not specified part of the *mv* function in such a way that an MIKRM form has the minimum number of terms. In this section we introduce a new approach to the minimization of incompletely specified *mv* functions that makes use of *symbolic, mixed Boolean-Arithmetic equation solving*.

The variable c_i is introduced as the entry for the i^{th} not specified minterm for the

truth vector d of an incompletely specified mv function. Thus, the *extended truth vector* e calculated according to Equation (IX.15) has symbolic equations dependent on c_i as its elements.

Example IX.6: The incompletely specified mv function $f(X)$, where X is a four-valued variable is given by the specified part $f_s = X^0$ and the not specified part $f_n = X^{23}$. Therefore, the truth vector d for the mv function $f(X)$ is given by

$$d = [1 \ 0 \ c_1 \ c_2]$$

The *extended truth vector* e computed according to Equation (IX.15) is obtained by the following matrix multiplication

$$e = M_4 \times d = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} c_2 \\ c_1 \\ c_1 \oplus c_2 \\ 0 \\ c_2 \\ c_1 \\ c_1 \oplus c_2 \\ 1 \\ 1 \oplus c_2 \\ 1 \oplus c_1 \\ 1 \oplus c_1 \oplus c_2 \\ 1 \\ 1 \oplus c_2 \\ 1 \oplus c_1 \\ 1 \oplus c_1 \oplus c_2 \end{bmatrix}$$

The *extended weight vector* w can be obtained similarly to the computation of the *extended truth vector* e . The elements of the *extended weight vector* w for an incompletely specified mv function $f(X)$ are determined by equations dependent on the variables c_i describing the don't care minterms. To determine the MIKRM with the minimal

number of terms, each equation in vector w has to be solved in such a way that the obtained value is minimal.

Example IX.7: The *extended weight vector* w for the Polarity P_1 from Example IX.2 and the Polarity P_2 , given below, is calculated for the *extended truth vector* e obtained in Example IX.6.

$$P_2 = K_2^{-1} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The *extended weight vector* w is calculated according to Equation (IX.19).

$$w = W \times e = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} c_2 \\ c_1 \\ c_1 \oplus c_2 \\ 0 \\ c_2 \\ c_1 \\ c_1 \oplus c_2 \\ 1 \\ 1 \oplus c_2 \\ 1 \oplus c_1 \\ 1 \oplus c_1 \oplus c_2 \\ 1 \\ 1 \oplus c_2 \\ 1 \oplus c_1 \\ 1 \oplus c_1 \oplus c_2 \end{bmatrix}$$

Thus, we obtain

$$w = \begin{bmatrix} 2 + c_2 \\ 2 + (1 \oplus c_1) + (1 \oplus c_1 \oplus c_2) \end{bmatrix}$$

Now the Equations describing the *extended weight vector* w have to be solved. This can be done for example by checking all possible combinations of assignments for c_1 and c_2 .

c_1	c_2	w_0	w_1
0	0	2	4
0	1	3	3
1	0	2	2
1	1	3	3

As one can observe from the above table, there are three possible assignments for the don't care minterms which lead to a minimal MIKRM form with two terms. For the selection $c_1 = 1$ and $c_2 = 0$ substituted to vector d we obtain according to Equations (IX.11) and (IX.12)

$$K_1 \times d = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

The set of obtained coefficients inserted in Equation (VIII.2) gives

$$f(X) = P_1^1 \oplus P_1^2 = X^{0123} \oplus X^{13}$$

Next the introduced methods are generalized for systems of incompletely specified *mv* functions.

IX.4.2 Extension to systems of incompletely specified *mv* functions

To obtain the *extended weight vector* w for a system of incompletely specified *mv* functions $F(f_0, \dots, f_{m-1})$, we redefine the concept of the *extended truth vector* e .

Definition IX.4: The *extended truth vector* e for a system of incompletely specified *mv* functions $F(f_0, \dots, f_{m-1})$ is given by

$$e = \bigcup_{i=0}^{m-1} (M_n \times d_i) \quad (\text{IX.22})$$

where \bigcup is the Boolean OR operator [100], $M_n \times d_i$ is a Boolean expression, m is the number of mv functions in $F(f_0, \dots, f_{m-1})$, d_i is the truth vector of f_i , and the multiplication has to be performed over Galois Field (2).

With the definition of the *extended truth vector* e for a system of incompletely specified mv functions according to Equation (IX.22), the *extended weight vector* w can be obtained by Equation (IX.19), where the *incidence matrix* W_n is created in the same way as for the completely specified mv function.

Example IX.8: The *extended weight vector* w for the system of two incompletely specified functions $f(X)$ and $g(X)$ will be calculated. The function $f(X)$ was given in Example IX.7. The function $g(X)$ is given by the specified part $g_s = X^2$ and the not specified part $g_n = X^0$. The truth vector representations are given by

$$f(X) = d_1 = \begin{bmatrix} 1 \\ 0 \\ c_1 \\ c_2 \end{bmatrix}, \quad g(X) = d_2 = \begin{bmatrix} c_3 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

First the *extended truth vector* e is computed according to Equation (IX.22)

$$e = M_4 \times d_1 \bigcup M_4 \times d_2 =$$

$$\begin{bmatrix}
 c_2 \cup 0 \\
 c_1 \cup 1 \\
 (c_1 \oplus c_2) \cup 1 \\
 0 \cup 0 \\
 c_2 \cup 0 \\
 c_1 \cup 1 \\
 (c_1 \oplus c_2) \cup 1 \\
 1 \cup c_3 \\
 (1 \oplus c_2) \cup c_3 \\
 (1 \oplus c_1) \cup (1 \oplus c_3) \\
 (1 \oplus c_1 \oplus c_2) \cup (1 \oplus c_3) \\
 1 \cup c_3 \\
 (1 \oplus c_2) \cup c_3 \\
 (1 \oplus c_1) \cup (1 \oplus c_3) \\
 (1 \oplus c_1 \oplus c_2) \cup (1 \oplus c_3)
 \end{bmatrix} = \begin{bmatrix}
 c_2 \\
 1 \\
 1 \\
 0 \\
 c_2 \\
 1 \\
 1 \\
 1 \\
 (1 \oplus c_2) \cup c_3 \\
 (1 \oplus c_1) \cup (1 \oplus c_3) \\
 (1 \oplus c_1 \oplus c_2) \cup (1 \oplus c_3) \\
 1 \\
 (1 \oplus c_2) \cup c_3 \\
 (1 \oplus c_1) \cup (1 \oplus c_3) \\
 (1 \oplus c_1 \oplus c_2) \cup (1 \oplus c_3)
 \end{bmatrix}$$

The Boolean properties $a \cup 1 = 1$ and $a \cup 0 = a$ have been applied to simplify the equations in the *extended truth vector* e . For the two polarities given by the *incidence matrix* W of Example IX.7 we obtain

$$w = W \times e = \begin{bmatrix}
 2 + c_2 + \left[(1 \oplus c_1 \oplus c_2) \cup (1 \oplus c_3) \right] \\
 2 + \left[(1 \oplus c_1) \cup (1 \oplus c_3) \right] + \left[(1 \oplus c_1 \oplus c_2) \cup (1 \oplus c_3) \right]
 \end{bmatrix}$$

Because c_3 occurs only in the term $1 \oplus c_3$, c_3 has to be 1 to obtain the minimal solution.

Thus,

$$w_{c_3=1} = \begin{bmatrix} 2 + c_2 + \left[1 \oplus c_1 \oplus c_2 \right] \\ 2 + \left[1 \oplus c_1 \right] + \left[1 \oplus c_1 \oplus c_2 \right] \end{bmatrix}$$

Now, only the not specified variables c_1 and c_2 have to be assigned. By checking all possible assignments we obtain

c_1	c_2	c_3	w_0	w_1
0	0	1	3	4
0	1	1	3	3
1	0	1	2	2
1	1	1	3	3

It can be observed, that the minimal solution is the same as obtained in Example IX.7, because the not specified part of function $g(X)$ can be assigned in such a way that it is equal to the completely specified function $f(X)$ obtained in Example IX.7.

In the previous sections we introduced a method to find the minimal MIKRM form based on matrix operations. However, the computation of the MIKRM form based on a Fast Algorithm still has the disadvantage of a large memory requirement for the function represented as minterms. Therefore, the next section introduces a method for the calculation of the MIKRM form from OMDDs.

IX.5 CALCULATION OF THE MIKRM FORM FROM OMDDS

One can understand each row vector of the transform matrix T_n , Equation (IX.13) as a *positive positive standard trivial function* [73,81]. Thus, the value a_i of a coefficient for the MIKRM expansion can be calculated by the intersection of the mv function with the respective *positive standard trivial function*. The mv function and the *positive standard trivial function* have to be in a disjoint representation, i.e. an array of disjoint cubes or an OMDD [80].

The row vectors of the transform matrix T_n obtained by the Kronecker matrix product according to Equation (IX.13), can also be derived differently by Equation (IX.23). K_i^j is the *positive standard trivial function* representation of the j^{th} row vector of the transform matrix K_i for the i^{th} variable X_i . A row vector of the transform matrix T_n in Equation (IX.23) is the truth-table representation of the result of the intersection $K_1^i \cap K_2^i \cap \dots \cap K_n^i$.

$$T_n = \begin{bmatrix} K_1^1 \cap K_2^1 \cap \dots \cap K_n^1 \\ K_1^1 \cap K_2^1 \cap \dots \cap K_n^2 \\ \dots \\ K_1^1 \cap K_2^1 \cap \dots \cap K_n^{p_n} \\ \dots \\ \dots \\ K_1^{p_1} \cap K_2^{p_2} \cap \dots \cap K_n^{p_n} \end{bmatrix} \quad (IX.23)$$

Because the intersection of OMDDs can be performed relatively fast [79,80], the OMDDs are an ideal representation to calculate the *positive standard trivial functions* by applying Equation (IX.23).

The coefficients a_i of the MIGRM form, Equation (IX.12), can now be easily determined by the intersection of the *mv* function with the corresponding *positive standard trivial function* u_i . Thus, OMDDs are again superior to the disjoint cube representation [73] because of the ease of performing the intersection of two functions.

Theorem IX.2: The coefficient a_i of the MIKRM form for an *mv* function f is given by

$$a_i = \begin{cases} 1 & \text{the number of minterms covered by } f \cap PSTF_i \text{ is odd} \\ 0 & \text{the number of minterms covered by } f \cap PSTF_i \text{ is even} \end{cases} \quad (IX.24)$$

where $PSTF_i$ is the *positive standard trivial function* corresponding to the i^{th} row of the

transform matrix T_n .

Proof: Trivial. The modulo-2 addition for each coefficient a_i in Equation (IX.11) is equivalent to a summation with determination if the sum is even or odd. \square

To speed up the calculation of the MIKRM form from the OMDD, an additional information is given to each node of the OMDD.

Definition IX.5: The *Node Weight* $NW(i)$ of the node i in the OMDD is given by the number of minterms covered in the subsequent nodes of the OMDD

$$NW(i) = \sum_{k=0}^{p_i-1} NW(C_k) \quad (\text{IX.25})$$

where $NW(C_k)$ is the number of minterms covered by the node C_k which is connected to the node i by the value k .

The calculation of the *Node Weight* and the MIKRM is illustrated in the following complete example.

Example IX.9: The coefficient a_7 ($P_1^3 P_2^1$) for the function $f(X_1, X_2)$ from Example IX.2 is calculated to illustrate the computation of spectral coefficients from an OMDD representation. The OMDD of the function $f(X_1, X_2)$ is shown in Figure 41.

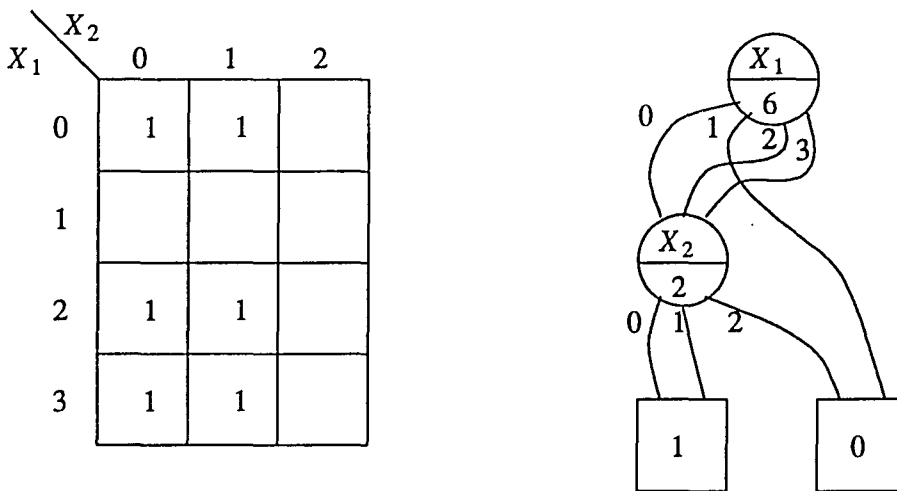


Figure 41. OMDD of $f(X_1, X_2)$.

The *Node Weight* of node X_2 is obtained by the summation of the connections to the 1-terminal. Thus, $NW(X_2)$ is 2. Node X_1 has three connections to node X_2 . Thus, the *Node Weight* of the node X_1 is given by $3 \times NW(X_2) = 6$. The corresponding *positive standard trivial function* u_7 for the calculation of the coefficient a_7 is shown in Figure 42. *Standard trivial function* u_7 is obtained by the intersection of K_1^3 and K_2^1 : $u_7 = K_1^3 \cap K_2^1$.

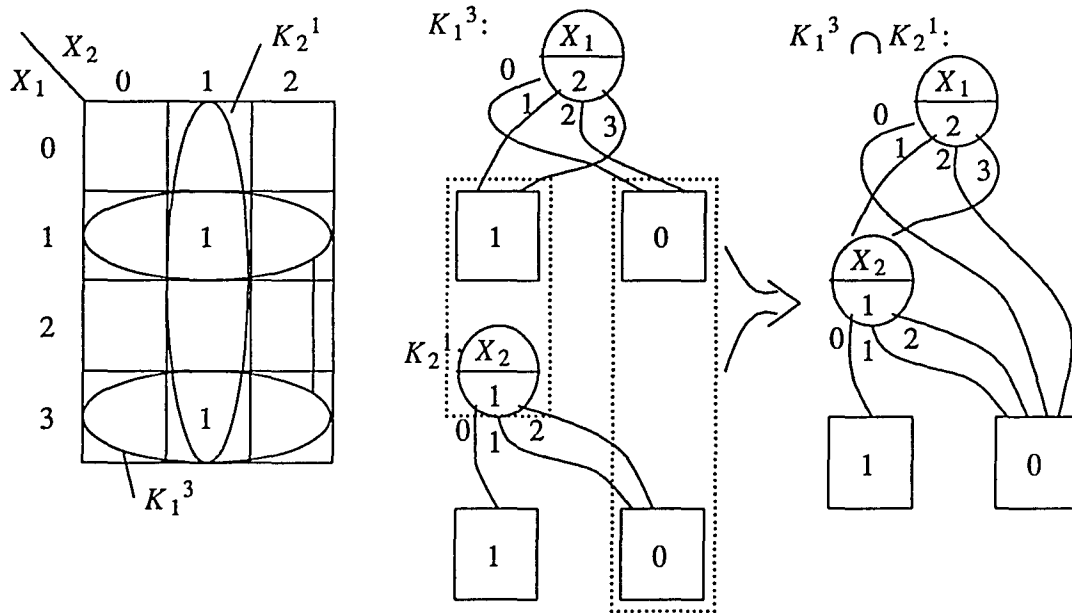


Figure 42. Standard trivial function $u_7 = K_1^3 \cap K_2^1$.

The realization of the intersection of the respective K_i^j with OMDDs can be performed by simple 'chaining' of the single-node OMDDs representing K_i^j . In the example the connections of node X_1 X_1 , representing K_1 , and X_2 , representing K_2 going to a 0-terminal are merged together. The 1-terminal of node X_1 is replaced by the node X_2 , as illustrated in Figure 42. The result of the intersection of the *positive standard trivial function* u_7 with $f(X_1, X_2)$ is shown in Figure 43. The number of minterms covered by the result of the intersection, Figure 43, is 1 as was obtained in Example IX.2. As stated in Theorem IX.2, the intersection of the *positive standard trivial function* with the

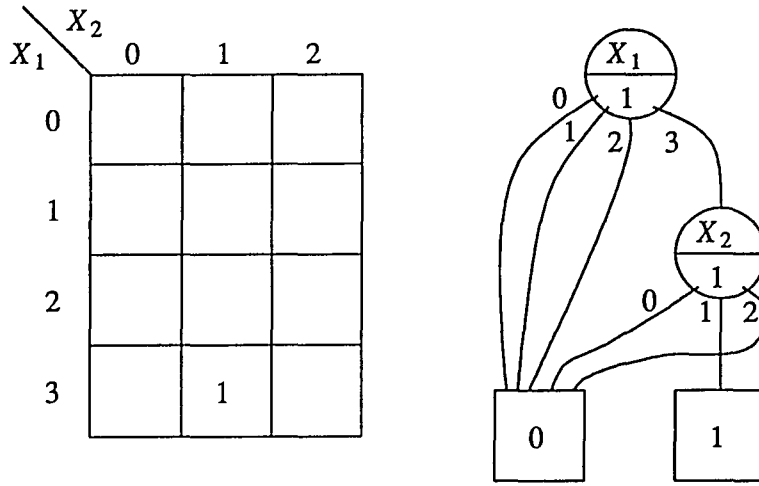


Figure 43. $u_7 \cap f(X_1, X_2)$.

function $f(X_1, X_2)$ covering an odd number of minterms determines that the *positive standard trivial function* is a term in the MIKRM form.

IX.6 EVALUATION AND RESULTS

For evaluation purposes the transforms to obtain the *extended truth vector* e and the *extended weight vector* have been implemented by the fast transforms that can be directly obtained from the respective Kronecker product. Thus, only the polarity matrices for the different valued variables have to be stored in the computer memory.

There are 3 possible polarities for a single two-valued variable, 27 for a three-valued variable, and 840 for a four-valued variable. Therefore, for a function having only two-valued variables the *extended weight vector* w has the dimension 3^n , for three-valued variables 27^n , and for four-valued variables 840^n . It can be easily seen that the dimension of the *extended weight vector* w becomes prohibitive for its computation in case of four-valued variable functions even if the function depends only on a few variables.

For the evaluation of the MIKRM in comparison to the KRM, the number of polar-

ities for a four-valued variable has been restricted to 60 randomly selected polarities. Table XXI gives the results obtained for some MCNC benchmark functions.

TABLE XXI
BENCHMARK RESULTS FOR MIKRM

	ESPRESSO	KRM	MIKRM 2-bit decoder
adr2	11	8	5
adr4	75	34	34
bw	22	22	22
con1	9	14	17
inc	30	34	40
rd53	31	20	15
rd73	127	63	40
rd84	255	107	54
squar5	25	23	25
xor5	16	5	4
5xp1	65	61	64

The column ESPRESSO lists the number of product terms obtained by the two-level minizer ESPRESSO. The following column gives the minimal number of product terms in a KRM form for the function being in a Boolean representation. For the computation of the MIKRM based on four-valued variables, pairs of Boolean variables have been taken to obtain a four-valued one (see Chapter VIII). The results obtained are given in the last column of Table XXI.

One can observe from the obtained results, that the MIKRM gives an up to 50% reduction of the number of product terms over the KRM. Moreover, the number of product terms of the KRM is the absolute minimum while only a local minimum is obtained for the MIKRM. This is due to the restriction of the number of polarities per variable to 60 instead of 840.

The results obtained give motivation to further investigate in MIKRM forms. However, methods to avoid the computation of all possible polarities similar to the ones introduced for GRMs [25,143] have to be developed to overcome the high computational

complexity of MIKRM forms. Additionally, it has to be investigated which multiple-valued decoders are practical for circuit realizations. Then, the search for the minimal MIKRM form can be restricted to the one which is optimal with respect to a certain set of multiple-valued decoders.

This Chapter introduced the concept of the canonic Multiple-valued Input Kronecker Reed-Muller form for systems of incompletely specified *mv* functions. This form is an extension of the KRM form to multiple-valued input, binary output logic. A method based on the *extended weight vector* w has been presented to determine the MIKRM forms having the minimal number of terms.

CHAPTER X

CONCLUSION

This dissertation introduced an extended theory of spectral methods for the application in logic synthesis. While the methods presented are also suitable for the logic synthesis for ASICs the concentration has been laid on their application to FPGAs.

The major contribution of this dissertation is the extension of spectral methods to allow the specification of spectra that describe the correlation to user defined gate structures. Therefore, it is possible to define spectra which give the correlation to a set of functions. Previous approaches taking advantage of spectral methods have been limited to the use of the known spectra like Walsh-type, Arithmetic, and Adding [62,65,89,90,108]. The extension presented is especially advantageous for the logic synthesis for FPGAs. There, the basic architecture consists of macrocells which can realize only a very limited set of functions. Therefore, a spectrum can be introduced that gives the correlation of the underlying Boolean function to the set of functions realizable with the macrocell of a certain FPGA. As an application of this method a multilevel multiplexer synthesis algorithm has been introduced. It overcomes the computational complexity of the known methods based on the computation of the complete Rademacher-Walsh spectrum [64,65]. Such a multiplexer circuit can be directly realized with the CLi 6000 series from Concurrent Logic [47] and the TPC 10 from Texas Instruments [50]. The *ACTTM* series from Actel [46] has a restricted connectivity between the multiplexers inside its macrocells. Therefore, a simple mapping algorithm has been developed that maps the obtained multiplexer circuit to the Actel *ACTTM* architecture. The results obtained by the computer implementation of these algorithms compared to existing

technology mapping algorithms [57,59,60] show that the application of the presented spectral methods in conjunction with cube calculus methods give better results in terms of computation time and required number of macrocells.

It is known, that the Binary Decision Diagram (BDD) can be directly realized by an $M(1)$ multiplexer circuit. The size of a BDD for a Boolean function depends on its variable ordering [78]. To take efficiently advantage of BDDs, one depends on finding a good variable ordering [133,134,144,145] to obtain a small BDD. The multiplexer synthesis algorithm introduced here can be applied to construct a quasi-minimal Shared Ordered Binary Decision Diagram (SOBDD) with attributed edges [104].

The main reason that spectral methods have not been popular in logic synthesis is their computational complexity and their high memory requirement. This is due to the computation of the spectra from a minterm representation. To overcome these deficiencies this dissertation introduces the application of the disjoint cube representation and the ODDs representations for the computation of the introduced extension of spectral methods. The computation time for a spectrum increases linearly with the number of cubes in the disjoint representation. Therefore, Chapter III introduced an algorithm to compute the quasi-minimal set of disjoint cubes for a Boolean function. It has been shown that on the average the obtained number of disjoint cubes is by factor 2 smaller than the one obtained by previous presented algorithms. Thus, the computation time for a spectrum from such a disjoint cube representation is by factor two faster. Additionally, the application of OBDDs has been introduced to the computation of spectra. Because they allow the fast computation of the intersection and tautology of two Boolean functions [79] they are especially suited for the application for spectral methods. Thus, their modification to allow their use for the computation of spectra has been presented.

In recent years more attention has been paid to take advantage of incompletely specified Boolean functions. However, current spectral methods do not allow the efficient

incorporation of incompletely specified Boolean functions. To overcome this deficiency the T spectrum has been introduced. The T spectrum allows to determine the contribution of the completely specified and the not specified part of the underlying Boolean function to each spectral coefficient, while in the previous spectral methods the information is spread over the complete spectrum. The linearization method [4,12,62] based on the Walsh spectrum and the autocorrelation of a Boolean function has been extended to take advantage of the T spectrum. To decrease the complexity of the obtained linear preprocessor σ the Rademacher order has been adapted to select the appropriate high value coefficients. The results obtained by the implementation verify the assumptions for the applicability of the linearization. For functions having highly correlated output functions, the autocorrelation and Walsh spectrum can be applied on the set of function to improve the complexity of the remaining core function. However, there is a trade of between the computational complexity of the autocorrelation versus the Walsh spectrum and the complexity of the obtained core function. For future research on the reduction of the complexity for the computation of the complete autocorrelation or Walsh spectrum, the linearization procedure can be based on the computation of second order coefficients only. Such an approach has the property, that a preprocessor of two-input EXOR gates is generated.

As an example for the application of spectral methods to multiple valued logic Chapter VIII and Chapter IX introduced the concept of Multiple-Valued Input Kronecker Reed Muller (MIKRM) forms. To overcome the memory requirements to store an m -valued function in its minterm representation in the computer memory, as would be necessary for the matrix or the Fast Transform calculation, a method has been introduced to calculate the MIKRM form from the OMDD representation. For large functions the memory requirement for the OMDD representation was found to be much smaller than for the cube representation [79], while it is also known that the cube representation usually has a

much smaller memory requirement than the minterm representation. Therefore, the approach proposed should be computationally more efficient than the approaches introduced for Boolean functions [23,24]. Moreover, solutions are proposed for incompletely specified functions which problem has not been solved even for the Boolean case: the KRMs. Additionally, the approach introduced is expanded to multi-output functions. The advantage of the MIKRM forms for *mv* functions over the multiple-valued input, multiple-valued output forms [85,141] is that they do not require special realization technologies but can be implemented using standard digital circuits. For instance multiple-valued literals are realized as decoders [83], function generators [138], or PLAs [146]. The circuit for the MIKRM additionally consists of AND and EXOR gates. Circuits of the above components can be easily mapped to new Field Programmable Gate Arrays (FPGAs) such as lookup-table based [45], multiplexer-based [46], or small granularity FPGAs [47,49,147]. Such mapping can lead to circuits with less logic blocks than obtained by standard mapping methods [56,57,59,60].

This dissertation introduced a general framework of spectral and logic synthesis methods that can be applied to the logic synthesis for FPGAs. The presented general top-down, breadth-first multilevel decomposition algorithm, where the decomposition is based on spectral methods, proved to be a powerful synthesis method superior to standard algebraic factorization concepts [4]. Moreover, the circuits obtained by the presented synthesis algorithm exhibit the property that their connectivity is limited two adjacent levels in the circuit. Thus, such a circuit is ideally suited for the realization with fine grain FPGAs like the CLi 6000 series, where the connectivity of a macrocell is limited to its four neighbors and only few global routing resources are available.

REFERENCES

1. R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, in *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
2. R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-Valued Minimization for PLA Optimization," *IEEE Trans. on CAD*, vol. 6, no. 5, pp. 727-750, September 1987.
3. L. B. Nguyen, M. A. Perkowski, and N. B. Goldstein, "PALMINI - Fast Boolean Minimizer for Personal Computers," *24th ACM/IEEE Design Automation Conference*, pp. 615-621, 1987.
4. R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proc of the IEEE*, vol. 78, no. 2, pp. 264-300, February 1990.
5. R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: Multi-Level Interactive Logic Optimization System," *IEEE Trans. on CAD*, vol. 6, no. 6, pp. 1062-1082, November 1989.
6. D. Bostick, G. D. Hachtel, R. Jacoby, M. R. Lightner, P. Moceyunas, C. R. Morrison, and D. Ravenscroft, "The Boulder Optimal Logic Design System," *IEEE Proc. of Int. Conf. on CAD, ICCAD-87*, pp. 62-65, November 1987.
7. K.-C. Chen and S. Muroga, "SYLON-DREAM: A Multi-Level Network Synthesizer," *Proc. ICCAD*, pp. 552-555, 1989.
8. K.-C. Chen, Y. Matsunaga, M. Fujita, and S. Muroga, "A Resynthesis Approach for Network Optimization," *28th ACM/IEEE Design Automation Conference*, pp. 458-463, San Francisco, CA, June 1991.
9. R. McNaughton, "Unate truth functions," *Trans. on IRE*, vol. 10, pp. 1-6, 1961.
10. W. G. Schneeweiss, in *Boolean Functions with Engineering Applications and Computer Programs*, Springer-Verlag, 1989.
11. C. Jay, "XOR PLDs Simplify Design of Counters and Other Devices," *EDN*, May, 1987.

12. D. Varma and E. A. Trachtenberg, "Design Automation Tools for Efficient Implementation of Logic Functions by Decomposition," *IEEE Trans. on CAD*, vol. 8, pp. 901-916, August 1989.
13. H. Landman, "Logic Synthesis at Sun," *IEEE Conference Paper, CH 2686-4/89/0000/0469*, 1989.
14. P. Calingaert, "Switching Function Canonical Forms Based on Commutative and Associative Binary Operations," *AIEE Trans.*, vol. 79, pp. 808-814, January 1961.
15. M. Cohn, "Inconsistent Canonical Forms of Switching Functions," *IRE Trans. Electron. Comput.*, vol. 11, p. 284, April 1962.
16. D. H. Green and L. S. Taylor, "Multiple-Valued Switching Circuit Design by Means of Generalised Reed-Muller Expansions," *Digital Processes*, vol. 2, pp. 63-81, 1976.
17. X. Wu, X. Chen, and S. L. Hurst, "Mapping of Reed-Muller Coefficients and the Minimization of Exclusive-OR Switching Functions," *IEE Proc. Pt. E*, vol. 129, no. 1, pp. 15-20, January 1982.
18. Ph. W. Besslich, "Efficient Computer Method for EXOR Logic Design," *Proc. of IEE Pt. E*, vol. 130, no. 6, pp. 203-206, 1983.
19. L. Csanky, "On the Generalized Reed-Muller Canonical Form of Boolean Functions," *Master of Science Thesis, University of California, Berkeley*, December 1972.
20. D. H. Green, in *Modern Logic Design*, Electronic Systems Engineering Series, 1986.
21. H. Fleisher, M. Tavel, and J. Yeager, "A Computer Algorithm for Minimizing Reed-Muller Canonical Forms," *IEEE Trans. on Comput.*, vol. 36, no. 2, pp. 247-250, February 1987.
22. P. K. Lui and J. C. Muzio, "Boolean Matrix Transforms for the Minimization of Modulo-2 Canonical Expansions," *University of Victoria technical report DCS-117-IR*, May 1989.
23. D. H. Green, "Reed-Muller Canonical Forms With Mixed Polarity and Their Manipulations," *Proc. of IEE Pt. E*, vol. 137, no. 1, January 1990.
24. D. H. Green, "Families of Reed-Muller Canonical Forms," *Int. J. of Electronics*, vol. 63, no. 2, pp. 259-280, January 1991.
25. A. E. A. Almaini, "Tabular Techniques for Reed-Muller Logic," *Int. J. of Electronics*, vol. 70, no. 1, pp. 23-34, January 1991.

26. D. Varma and E. A. Trachtenberg, "Computation of Reed-Muller Expansions of Incompletely Specified Boolean Functions From Reduced Representations," *IEE Proc. Pt. E*, vol. 138, no. 2, pp. 85-92, March 1991.
27. B. J. Falkowski and M. A. Perkowski, "One more Way to Calculate Generalized Reed-Muller Expansions of Boolean Functions," *Int. J. Electronics*, vol. 71, no. 3, pp. 383-396, 1991.
28. L. Csanky, M. A. Perkowski, and I. Schäfer, "Canonical Restricted Mixed-Polarity Exclusive Sums of Products and the Efficient Algorithm for Their Minimization," *accepted for the publication in Proc. of IEE Pt. E*, May 1992.
29. I. Schäfer and M. A. Perkowski, "Multiple-Valued Input Generalized Reed-Muller Forms," *accepted for the publication in Proc. of IEE Pt. E*, May 1992.
30. I. Schäfer and M. A. Perkowski, "On the Minimal Multiple-Valued Input Kronecker Reed-Muller Form for Incompletely Specified Multiple-Valued Input Functions," *submitted to IEEE Trans. on Computers*, April 1992.
31. M. A. Perkowski, L. Csanky, A. Sarabi, and I. Schäfer, "Minimization of Mixed-Polarity Canonical AND/EXOR Forms," *to appear in Proc. ICCD'92*, October, 1992.
32. S. M. Reddy, "Easily Testable Realizations for Logic Functions," *IEEE Trans. on Comput.*, vol. 21, no. 11, pp. 1183-1188, November 1972.
33. K. L. Kodandapani, "A Note on Easily Testable Realizations for Logic Functions," *IEEE Trans. on Comput.*, pp. 332-333, March 74.
34. K. K. Saluja and S. M. Reddy, "Fault Detecting Test Sets for Reed-Muller Canonical Networks," *Trans. on Comput.*, vol. 24, no. 10, pp. 995-998, October 1975.
35. D. K. Pradhan, "Universal Test Sets for Multiple Fault Detection in AND-EXOR Arrays," *IEEE Trans. on Comput.*, vol. 27, no. 2, pp. 181-187, February 1978.
36. B. B. Bhattacharya, B. Gupta, S. Sarkar, and A. K. Choudhury, "Testable Design of RMC Networks with Universal Tests for Detecting Stuck-At and Bridging Faults," *IEE Proc. Pt. E*, vol. 132, no. 3, pp. 155-161, May 1985.
37. T. Damarla and M. Karpowsky, "Detection of Stuck-At and Bridging Faults in Reed-Muller Canonical (RMC) Networks," *IEE Proc. Pt. E*, vol. 136, no. 5, pp. 430-433, September 1989.
38. M. Helliwell and M. A. Perkowski, "A Fast Algorithm to Minimize Multi-Output Mixed-Polarity Generalized Reed-Muller Forms," *25th ACM/IEEE Design Automation Conference*, pp. 427-432, Anaheim, CA, June 1988.

39. T. Sasao, "EXMIN: A Simplification Algorithm for Exclusive-OR-Sum-of-Products Expressions for Multiple-Valued Input Two-Valued Output Functions," *20th Int. Symp. on Multiple-Valued Logic*, pp. 128-135, May 1990.
40. T. Sasao, "A Transformation of Multiple-Valued Input Two-Valued Output Functions and its Application to Simplification of Exclusive-or-Sum-of-Products Expressions," *21th Int. Symp. on Multiple-Valued Logic*, pp. 270-279, Victoria, BC, May 1991.
41. J. M. Saul, "An Improved Algorithm for the Minimization of Mixed Polarity Reed-Muller Representations," *Int. Conf. on Comput. Design: VLSI in Comput. and Processors*, pp. 372-375, September 1990.
42. Ph. W. Besslich and M. W. Riege, *A New Approach to Mod-2 Sum Synthesis*, March 1991.
43. Ph. W. Besslich and M. W. Riege, "An Efficient Program for Logic Synthesis of Mod-2 Sum Expressions," *Euro ASIC*, pp. 136-141, Paris, 1991.
44. E. B. Pitty, "A Critique of the GATEMAP Logic Synthesis System," *The Plessy Company, Report*, 1988.
45. Xilinx, in *The Programmable Gate Array Data Book*, 1989.
46. Actel, in *ACT Family Field Programmable Gate Array Data Book*, March 1991.
47. Inc. Concurrent Logic, "CLi6000 Series Field Programmable Gate Array," *Preliminary Information*, December 1991.
48. A. Auer, in *PLD Handbuch: Tabellen und Daten*, Hl thig Buch Verlag, 1990.
49. Algotronix Ltd., *Configurable Array Logic User Manual*, Edinburgh, UK, 1991.
50. Texas Instruments, *TPC10 Series 1.2-um Field-Programmable Gate Arrays*, July 1991.
51. K. Karplus, "Xmap: A Technology Mapper for Table-Lookup Field-Programmable Gate Arrays," *28th ACM/IEEE Design Automation Conference*, pp. 240-243, San Francisco, CA, June 1991.
52. K. Karplus, "Amap: A Technology Mapper for Selector-Based Field-Programmable Gate Arrays," *28th ACM/IEEE Design Automation Conference*, pp. 244-247, San Francisco, CA, June 1991.
53. R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *28th ACM/IEEE Design Automation Conference*, pp. 227-233, San Francisco, CA, June 1991.

54. R. J. Francis, J. Rose, and Z. Vranesic, "Technology Mapping of Lookup Table-Based FPGAs for Performance," *ICCAD-91*, pp. 568-571, Santa Clara, CA, November 1991.
55. R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *ICCAD-91*, pp. 564-567, Santa Clara, CA, November 1991.
56. R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays," *ICCAD-91*, pp. 572-575, Santa Clara, CA, November 1991.
57. R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "An Improved Synthesis Algorithm for Multiplexor-based PGAs," *1st ACM Workshop on FPGAs*, pp. 97-102, Berkeley, CA, February 1992.
58. P. Sawkar and D. Thomas, "Technology Mapping for Table-Look-Up based Field Programmable Gate Arrays," *1st ACM Workshop on FPGAs*, pp. 83-88, Berkeley, CA, February 1992.
59. A. Bedarida, S. Ercolani, and G. DeMicheli, "A New Technology Mapping Algorithm for the Design and Evaluation of Fuse/Antifuse-based Field-Programmable Gate Arrays," *1st ACM Workshop on FPGAs*, pp. 103-108, Berkeley, CA, February 1992.
60. K.-C. Chen, "Logic Minimization of Lookup-Table Based FPGAs," *1st ACM Workshop on FPGAs*, pp. 71-76, Berkeley, CA, February 1992.
61. R. J. Lechner, "Harmonic Analysis of Switching Functions," in *Recent Developments in Switching Theory*, ed. A. Mukhopadhyay, pp. 121-228, Academic, 1971.
62. M. F. Karpovsky, in *Finite Orthogonal Series in the Design of Digital Devices*, John Wiley, 1976.
63. S. L. Hurst, in *The Logical Processing of Digital Signals*, Crane Russak, 1978.
64. A. M. Lloyd, "Design of Multiplexer Universal-Logic-Module Networks Using Spectral Techniques," *IEE Proc. Pt. E.*, vol. 127, no. 1, pp. 31-36, January 1980.
65. S. L. Hurst, D. M. Miller, and J. C. Muzio, in *Spectral Techniques in Digital Logic*, Academic Press, 1985.
66. J. Poswig, "Disjoint Decomposition of Boolean Functions," *IEE Proc. Pt. E.*, vol. 138, no. 1, January 1991.
67. E. Eris and J. C. Muzio, "Spectral Testing of Circuit Realization Based on Linearizations," *IEE Proc. Pt. E.*, vol. 133, pp. 73-78, March 1986.

68. P. K. Lui and J. C. Muzio, "Spectral Signature Testing of Multiple Stuck-at Faults in Irredundant Combinational Networks," *IEEE Trans. on Comput.*, vol. 35, no. 12, pp. 1088-1092, December 1986.
69. S. L. Hurst, "Use of Linearisation and Spectral Techniques in Input and Output Compaction Testing of Digital Networks," *IEE Proc. Pt. E.*, vol. 136, no. 1, pp. 48-56, January 1989.
70. Ph. W. Besslich and T. Lu, in *Diskrete Orthogonaltransformationen*, Springer Verlag, 1990.
71. 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 Int. Symp. Circ. & Systems (IS-CAS)*, May 1990.
72. I. Schäfer, "An Efficient Cube Comparison Method for Discrete Spectral Transformations of Logic Functions," *Master of Science Thesis, Portland State University*, June 1990.
73. B. J. Falkowski, I. Schäfer, and M. A. Perkowski, "Effective Computer Methods for the Calculation of the Rademacher-Walsh Spectrum for Boolean Functions," *to appear in IEEE Trans. on CAD.*, July 1992.
74. B. J. Falkowski, "Spectral Methods for Boolean and Multiple-Valued Input Logic Functions," *Ph.D. dissertation, Portland State University*, May 1991.
75. B. J. Falkowski and M. A. Perkowski, "On the Calculation of Generalized Reed-Muller Canonical Expansions from Disjoint Cube Representation of Boolean Functions," *IEEE 33rd Midwest Symp. on Circuits & Systems*, pp. 1131-1133, August 1990.
76. C. Y. Lee, "Representation of Switching Circuits by Binary-Decision Programs," *Bell System Technical Journal*, vol. 38, pp. 985-999, July 1959.
77. S. B. Akers, "Binary Decision Diagrams," *IEEE Trans. on Comput.*, vol. 27, no. 6, pp. 509-516, June 1978.
78. R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Comput.*, vol. 35, no. 8, pp. 667-691, August 1986.
79. K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," *27th ACM/IEEE Design Automation Conference*, pp. 40-45, June 1990.
80. A. Srinivasan, T. Kam, S. Malik, and R.K. Brayton, "Algorithms for Discrete Function Manipulation," *IEEE Proc. of Int. Conf. on CAD, ICCAD-90*, pp. 92-95, 1990.

81. S. L. Hurst, "The Application of Chow Parameters and Rademacher-Walsh Matrices in the Synthesis of Binary Functions," *Comput. J.*, vol. 16, no. 2, 1973.
82. I. Schäfer and M. A. Perkowski, "Synthesis of Multi-Level Multiplexer Circuits for Incompletely Specified Boolean Functions with Mapping to Multiplexer Based FPGAs," *submitted to IEEE Trans. on CAD*, April 1992.
83. T. Sasao, "On the Optimal Design of Multiple-Valued PLA's," *IEEE Trans. on Comput.*, vol. 38, no. 4, pp. 582-592, April 1989.
84. M. Davio, J. P. Deschamps, and A. Thayse, in *Discrete and Switching Functions*, McGraw-Hill, 1978.
85. K. L. Kodandapani and R. V. Setlur, "Reed-Muller Canonical Forms in Multivalued Logic," *IEEE Trans. on Comput.*, pp. 628-636, June 1975.
86. Bo-Gwan Kim and D. L. Dietmeyer, "Multilevel Logic Synthesis with Extended Arrays," *IEEE Trans. on Comput.*, vol. 11, no. 2, pp. 142-157, February 1992.
87. A. K. Jain, in *Fundamentals of Digital Image Processing*, Prentice Hall, 1989.
88. A. V. Oppenheim and R. W. Schaffer, in *Discrete-Time Signal Processing*, Prentice Hall, 1990.
89. A. M. Lloyd, "A Consideration of Orthogonal Matrices, Other Than the Rademacher-Walsh types for the Synthesis of Digital Networks," *Int. J. Electronics*, vol. 47, no. 3, pp. 205-212.
90. I. Schäfer, B. J. Falkowski, and M. A. Perkowski, "A Fast Computer Implementation of Adding and Arithmetic Multi Polarity Transforms for Logic Design," *IEEE 34th Midwest Symp. on Circuit & Systems*, Monterey, CA, May 1991.
91. R. Rudell, "Multiple-Valued Logic Minimization for PLA Synthesis," *Master of Science Thesis, University of California, Berkeley*, June 1986.
92. I. Schäfer and M. A. Perkowski, "Multiple Valued Generalized Reed-Muller Forms," *21st Int. Symp. on Multiple-Valued Logic*, pp. 40-48, May 1991.
93. M. A. Perkowski and P. D. Johnson, "Canonical Multi-Valued Input Reed-Muller Trees and Forms," *3rd NASA Symposium on VLSI Design*, 1991.
94. S. Purwar and A. K. Susskind, "Computation of Walsh Spectrum from Binary Decision Diagram and Binary Decision Diagram from Walsh Spectrum," *Computers & Elect. Engng.*, vol. 15, no. 2, pp. 59-65, 1989.

95. B. J. Falkowski and M. A. Perkowski, "One More Way to Calculate the Hadamard-Walsh Spectrum for Completely and Incompletely Specified Boolean Functions," *Int. J. Electronics*, vol. 69, no. 5, pp. 595-602, 1990.
96. 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," *IEEE 34th Midwest Symp. on Circuits & Systems*, Monterey, CA, May 1991.
97. S. Purwar, "An Efficient Method of Computing Generalized Reed-Muller Expansions from Binary Decision Diagram," *IEEE Trans. on Comput.*, vol. 40, no. 11, pp. 1298-1301, November 1991.
98. 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," *IEEE 33rd Midwest Symp. on Circuits & Systems*, Calgary, Alberta, August 1990.
99. B. J. Falkowski and M. A. Perkowski, "Algorithm for the Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions," *Int. J. Electronics*, vol. 70, no. 3, pp. 533-538, 1991.
100. D. L. Dietmayer, in *Logic Design of Digital Systems*, Allyn and Bacon, 1978.
101. W. Pugh, "Skip Lists: a Probabilistic Alternative to Balanced Trees," *Communications of the ACM*, vol. 44, no. 6, pp. 668-676, June 1990.
102. B. M. E. Moret, "Decision Trees and Diagrams," *Computing Surveys*, vol. 14, no. 4, pp. 594-623, December 1982.
103. R. E. Bryant, "Symbolic Manipulation of Boolean Functions Using a Graphical Representation," *22nd ACM/IEEE Design Automation Conference*, pp. 688-694, 1985.
104. S. Minato, N. Ishiura, and S. Yajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," *27th ACM/IEEE Design Automation Conference*, pp. 52-57, 1990.
105. J. C. Muzio, "Nonorthogonal Transforms for Logical Design," *Technical Report CS77006-R, Virginia Polytechnic Institute and State University*, June 1977.
106. B. J. Falkowski and M. A. Perkowski, "Algorithms for the Calculation of Hadamard-Walsh Spectrum for Completely and Incompletely Specified Boolean Functions," *Proc. of Int. Conf. on Comput. and Comm.*, March 1990.
107. 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 Int. Symp. Circ. & Systems (ISCAS)*, May 1990.

108. I. Schäfer, B. J. Falkowski, and M. A. Perkowski, "Generation of Adding and Arithmetic Multi-Polarity Transforms for Incompletely Specified Boolean Functions," *accepted for the publication in the Int. J. of Electronics*, February 1992.
109. C. R. Edwards, "The Application of the Rademacher-Walsh Transform to Boolean Function Classification and Threshold Logic Synthesis," *IEEE Trans. on Comput.*, vol. 24, pp. 48-62, 1975.
110. S. W. Yablonskii, "On Algorithmic Obstacles to the Synthesis of Minimal Contact Networks," *Problemy Kibernetiki*, no. 2, pp. 75-121, 1959.
111. S. L. Hurst, D. M. Miller, and J. C. Muzio, "Spectral Method of Boolean Function Complexity," *Electronic Letters*, vol. 18, pp. 572-574, 1982.
112. D. Varma and E. A. Trachtenberg, "On The Estimation of Logic Complexity for Design Automation Applications," *8th IEEE Int. Conf. on Comput. Design: VLSI in Computers & Processors (ICCD)*, pp. 368-371, September 1990.
113. C. Moraga, "Comments on a Method of Karpovsky," *Inform. Contr.*, vol. 39, no. 3, pp. 243-246, December 1978.
114. L. Lavagno, S. Malik, R. K. Brayton, and A. Sangiovanni-Vincentelli, "MIS-MV: Optimization of Multi-Level Logic with Multiple-Valued Inputs," *ACM/IEEE Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 560-563, Santa Clara, CA, November 1990.
115. S. S. Yau and C. K. Tang, "Universal Logic Modules and Their Applications," *IEEE Trans. on Comp.*, vol. 19, no. 2, pp. 141-149, February 1970.
116. X. Chen and S. L. Hurst, "A Consideration of the Minimum Number of Input Terminals on Universal Logic Gates and Their Realization," *Int. J. Electronics*, vol. 50, pp. 1-13, January 1981.
117. S. Bandyopadhyay, A. Pal, and A. K. Choudhury, "Characterization of Unate Cascade Realizability Using Parameters," *IEEE Trans. on Comput.*, vol. 24, no. 2, pp. 218-219, February 1975.
118. D. G. Whitehead, "Algorithm for Logic-Circuit Synthesis by Using Multiplexers," *Electronic Letters*, vol. 1977, no. 12, pp. 355-356, June 1977.
119. A. E. A. Almaini and M. E. Woodward, "An Approach to the Control Variable Selection Problem for Universal Logic Modules," *Digital Processes*, vol. 3, pp. 189-206, 1977.
120. R.P. Voith, "ULM Implicants for Minimization of Universal Logic Module Circuits," *IEEE Trans. on Comput.*, vol. 26, no. 5, pp. 417-424, May 1977.

121. B. Dormido and D. Canto, "Systematic Synthesis of Combinational Circuits Using Multiplexers," *Electronic Letters*, vol. 14, no. 18, pp. 588-590, August 1978.
122. G. G. Langdon, "A Decomposition Chart Technique to Aid in Realizations with Multiplexers," *IEEE Trans. on Comput.*, vol. 27, no. 2, pp. 157-159, February 1978.
123. L. A. M. Bennett, "The Application of Map-Entered Variables to the Use of Multiplexers in the Synthesis of Logic Functions," *Int. J. Electronics*, vol. 45, no. 4, pp. 373-379, 1978.
124. D. Mange and E. Sanchez, "Synthese des Fonctions Logiques avec des Multiplexeurs," *Digital Processes*, vol. 4, pp. 29-44, 1978.
125. A. J. Tosser and D. Aoulad-Syad, "Cascade Networks of Logic Functions Built in Multiplexer Units," *IEE Proc. Pt. E*, vol. 127, no. 2, pp. 64-68, March 1980.
126. D. H. Green and M. A. Chughtai, "Use of Multiplexers in Direct Synthesis of ASM-based Designs," *IEE Proc. Pt. E*, vol. 133, no. 4, pp. 194-200, July 1986.
127. A. Pal, "An Algorithm for Optimal Logic Design Using Multiplexers," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 755-757, August 1986.
128. R. K. Gorai and A. Pal, "Automated Synthesis of Combinational Circuits by Cascade Networks of Multiplexers," *IEE Proc. Pt. E*, vol. 137, no. 2, pp. 164-170, March 1990.
129. T.F. Tabloski and F.J. Mowle, "A Numerical Expansion Technique and Its Application to Minimal Multiplexer Logic Circuits," *IEEE Trans. on Comput.*, vol. 25, no. 7, pp. 684-702, July 1976.
130. R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays," *27th ACM/IEEE Design Automation Conference*, pp. 620-625, 1990.
131. F. Mailhot and G. DeMicheli, "Technology Mapping Using Boolean Matching and Don't Care Sets," *IEEE Proc. of European Design Automation Conference*, pp. 212-216, Glasgow, March 1990.
132. S. Ercolani and G. DeMicheli, "Technology Mapping for Electrically Programmable Gate Arrays," *28th ACM/IEEE Design Automation Conference*, pp. 234-239, San Francisco, CA, June 1991.
133. M. Fujita, Y. Matsunaga, and T. Kakuda, "On Variable Ordering of Binary Decision Diagrams for the Application of Multilevel Logic Synthesis," *IEEE European Design Automation Conference*, pp. 50-54, Amsterdam, Netherland, February 1991.

134. N. Ishiura, H. Sawada, and S. Yajima, "Minimization of Binary Decision Diagrams Based on Exchanges of Variables," *ICCAD-91*, pp. 472-475, Santa Clara, CA, November 1991.
135. S. B. Akers, "On a Theory of Boolean Functions," *J.SIAM*, vol. 7, pp. 487-498, December 1959.
136. B. R. K. Reddy and A. L. Pai, "Reed-Muller Transform Image Coding," *Computer Vision, Graphics, and Images*, vol. 42, pp. 48-61, 1988.
137. T. Sasao, "Multiple-Valued Decomposition of Generalized Boolean Functions and the Complexity of Programmable Logic Arrays," *IEEE Trans. on Comput.*, vol. 30, no. 9, pp. 636-643, September 1981.
138. T. Sasao, "MACDAS: Multi-Level AND-OR Circuit Synthesis Using Two-Variable Function Generators," *Proc. 23rd Design Automation Conference*, pp. 86-93, June 1986.
139. M. A. Perkowski, M. Helliwell, and P. Wu, "Minimization of Multiple-Valued Input Mutli-Output Mixed-Radix Exclusive Sums of Products for Incompletely Specified Boolean Functions," *19th Int. Symp. on Multiple-Valued Logic*, pp. 256-263, May 1989.
140. B. J. Falkowski and M. A. Perkowski, "Walsh Type Transforms for Completely and Incompletely Specified Multiple-Valued Input Binary Functions," *20th Int. Symp. on Multiple-Valued Logic*, pp. 75-82, May 1990.
141. Z. Hu, "The Simple Methods for Evaluating the Coefficients of the Canonical RM Expansion of Multivalued Functions," *Int. J. of Electronics*, vol. 63, no. 6, pp. 851-856, June 1987.
142. P. Gilliam, "A Practical Parallel Algorithm for the Minimization of Kronecker Reed-Muller Expansions," *Master of Science Thesis, Portland State University*, August 1991.
143. A. Sarabi and M. A. Perkowski, "Fast Exact and Quasi-Minimal Minimization of Highly Testable Fixed-Polarity AND/XOR Canonical Networks," *Proc. 29th Design Automation Conference*, Anaheim, CA, June, 1992.
144. S. J. Friedman and K. J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," *24th ACM/IEEE Design Automation Conference*, pp. 348-356, 1987.
145. K. M. Butler, D. E. Ross, R. Kapur, and M. R. Mercer, "Heuristic to Compute Variable Orderings for Efficient Manipulation of Ordered Binary Decision Diagrams," *28th ACM/IEEE Design Automation Conference*, pp. 417-420, San Francisco, CA, June 1991.

146. S. Yang and M. Ciesielski, "A Generalized PLA Decomposition with Programmable Encoder," *Int. Workshop on Logic Synthesis*, May 1989.
147. E. A. Walkup, S. Hauck, G. Borriello, and C. Ebeling, "Routing-Directed Placement for the TRIPTYCH FPGA," *1st ACM Workshop on FPGAs*, pp. 33-38, Berkeley, CA, February 1992.

APPENDIX

SPECTRA MANUAL PAGE

SYNOPSIS:

spectra <filename> do <option> (<type>)

DESCRIPTION:

act-map - multiplexer synthesis with M(1) multiplexers with following mapping to actel FPGA macrocells.

disjoint - computation of a disjoint cover for the given SOP.

esop2sop - computation of a disjoint cover for the given ESOP.

linearization <type> - linearization of the given disjoint SOP based on the spectrum determined by <type> (autocorrelation, walsh). The resulting EXOR preprocessor is given in the file *filename.sigma.blif* and the remaining core function in the file *filename.core.blif*.

literal_count - computation of the number of literals in given Sum of Product form.

multiplexer <type> - multi-level multiplexer synthesis for multiplexers with <type> data-select variables. The obtained circuit description is given in the file *filename.mblif*.

spectrum <type> <order> - computation of the spectrum given by <type> (autocorrelation, walsh, walsh2d, user). The type *user* is a transform based on user defined standard trivial functions. As a default the Adding transform has been implemented. The parameter <order> allows to specify a certain order to be computed.

INPUT/OUTPUT:

spectra accepts the standard truth table format (.tt). The output for multi-level networks is given in the standard Berkeley exchange format *blif*, for two-level networks in the truth table format.

AUTHOR:

Ingo Schläfer

SEE ALSO:

techmap(1)

Orthogonal and Nonorthogonal Expansions for Multi-Level Logic Synthesis for Nearly Linear Functions and their Application to Field Programmable Gate Array Mapping, Ph.D. dissertation, Ingo Schläfer, June 1992.