1994

Logic Synthesis with High Testability for Cellular Arrays

Andisheh Sarabi
Portland State University

LOGIC SYNTHESIS WITH
HIGH TESTABILITY FOR CELLULAR ARRAYS

by

ANDISHEH SARABI

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
1994

DISSERTATION APPROVAL

The abstract and dissertation of Andisheh Sarabi for the Doctor of Philosophy in Electrical and Computer Engineering were presented on April 29, 1994, and accepted by the dissertation committee and the department.

COMMITTEE APPROVALS:

[redacted]

Marek A. Perkowski, Chair

[redacted]

Malgorzata E. Chrzanowska-Jeske

[redacted]

Laszlo Csanky

[redacted]

W. Robert Daasch

[redacted]

Michael A. Driscoll

[redacted]

F. Rudolf Beyl
Representative of the Office of Graduate Studies

DEPARTMENT APPROVAL:

[redacted]

Rolf Schaumann, Chair
Department of Electrical Engineering

****

ACCEPTED FOR PORTLAND STATE UNIVERSITY BY THE LIBRARY

[redacted] on 14 June 1994

ABSTRACT

An abstract of dissertation of Andisheh Sarabi for the Doctor of Philosophy in Electrical and Computer Engineering presented on April 29, 1994.

Title: Logic Synthesis with High Testability for Cellular Arrays

The new Field Programmable Gate Array (FPGA) technologies and their structures have opened up new approaches to logic design and synthesis. The main feature of an FPGA is an array of logic blocks surrounded by a programmable interconnection structure. Cellular FPGAs are a special class of FPGAs which are distinguished by their fine granularity and their emphasis on local cell interconnects. While these characteristics call for specialized synthesis tools, the availability of logic gates other than Boolean AND, OR and NOT in these architectures opens up new possibilities for synthesis. Among the possible realizations of Boolean functions, XOR logic is shown to be more compact than AND/OR and also highly testable. In this dissertation, the concept of structural regularity and the advantages of XOR logic are used to investigate various synthesis approaches to cellular FPGAs, which up to now have been mostly nonexistent. Universal XOR Canonical Forms, Two-level AND/XOR, restricted factorization, as well as various Directed Acyclic Graph structures are among the proposed approaches. In addition, a new comprehensive methodology for the investigation of all possible XOR canonical forms is introduced. Additionally, a new compact class of XOR-based Decision Diagrams for the representation of Boolean functions, called Kronecker Functional Decision Diagrams (KFDD), is presented. It is shown that for the standard, hard,

benchmark examples, KFDDs are on average 35% more compact than Binary Decision Diagrams, with some reductions of up to 75% being observed.

To My Mother and to the Memory of My Grand Mother (Khanoom jAn)

ACKNOWLEDGEMENT

I would like to thank Dr. Marek A. Perkowski, my advisor, for his free-spirited approach which has provided me with guidance as well as substantial degree of liberty during my research. His vision for XOR logic has provided me with much impetus in my research direction. I would also like to thank Dr. W. Robert Daasch for being a friend as well as a true supporter from the start to this very day. Also my thanks go to Dr. F. Rudolf Beyl for his preciseness and his guidance both on my mathematical endevours as well as this dissertation. I would also like to thank Dr. Laszlo Csanky, Dr. Michael Driscoll, and Dr. Malgorzata Chrzanowska-Jeske for serving in my committee.

I would also like to acknowledge the collaborative work of our colleagues Rolf Drechsler, Prof. Bernd Becker, and Michael Theobald of J. W. Goethe University in Frankfurt, Germany. Through our fruitful collaboration, we have been able to demonstrate the potentials of Kronecker Functional Decision Diagrams as compact canonical representations for Boolean Functions.

My thanks also go to the National Science Foundation for their funding of the work presented in this dissertation. Also, I would like to acknowledge the travel grants I have received from the Academically Controlled Auxiliary Activities Committee at Portland State University. It was through their generous fundings of my participation in DAC'92, ICCD'92, Reed-Muller Workshop'93, and DAC'94 that the direction of this dissertation was made possible. Especially my participation in Reed-Muller Workshop in Hamburg was very crucial for our collaborative efforts with the German colleagues.

And finally my special thanks go to Ms. Shirley Clark, Laura Riddell, and all the staff in Electrical Engineering office for their supports through these years. This

dissertation is dedicated to my mother without whose support, the work and the writing of this dissertation would have not been possible.

Table of Contents

		Page
List of Tables		vi
List of Figures		viii
Chapter		
1	Introduction	1
2	Boolean Functions and their Representations	15
	2.1 Introduction	15
	2.2 Models of Switching Circuit Behavior and Structure	16
	2.2.1 Boolean functions and Model of Behavior 2.2.2 Switching Expressions and Models of Structure 2.2.3 Circuit Realization Based on GF 2 transforms	
	2.3 Functional Completeness	20
	2.4 Universal XOR Canonical Forms and their Number	26
	2.5 Generation of Different Families of AND/OR Bases	29
	2.5.1 Positive Polarity $\alpha\rho$ Family of Bases 2.5.2 Consistent Generalized $\alpha\rho$ Family of Bases 2.5.3 Generalized $\alpha\rho$ Family of Bases 2.5.4 $\alpha\rho\sigma$ Family of Bases 2.5.5 π Operation on $\alpha\rho\sigma$ Family of Bases	
	2.6 AND/XOR Canonical Forms	38
	2.7 Summary	47
3	Minimal Realization of Boolean Functions in Fixed Polarity AND/XOR Forms	49
	3.1 Introduction	49
	3.2 Approaches to Minimization	52

3.3 Improved Techniques for a Given CGRM Realization of a Boolean Function 55

3.3.1 Cube Comparison Method and Spectral Methods
3.3.2 Generation of the Monoterms Representing Each of the Disjoint Cubes
3.3.3 Implementation of the Ring Sum Operation

3.4 Minimal Polarity Vector 69

3.4.1 Subtracting Monoterms of Disjoint Cubes
3.4.2 Expansion Monoterms
3.4.3 The Minimal Polarity

3.5 Minimization Schemes 96

3.5.1 The "Exhaustive" Search Approach
3.5.2 The Heuristic Search Approach
3.5.3 Analysis of the Minimization Methods

3.6 Summary 104

4 Application of CGRMIN in Minimal Realization of Boolean Functions in Generalized AND/XOR Forms 106

4.1 Introduction 106

4.2 Canonical Restricted Mixed Polarity Forms 107

4.3 Algorithm for Quasi-minimal CRMP Synthesis 111

4.4 Experimental Results 113

4.5 Summary 114

5 Complex Maitra Logic Array Approach to CA-Type FPGA Synthesis 116

5.1 Introduction 116

5.2 Maitra terms and Complex terms 118

5.3 Restricted Factorization Theory 120

5.3.1 Term Combinability
5.3.2 Realization of functions in Minimized Restricted Factorized Form

5.4 Technology Folding 125

5.5 Summary 135

6 Minimal Multi-Level Realization of Boolean Functions Based on Kronecker Functional Decision Diagrams 136

6.1 Introduction 136

6.2 Decision Diagrams 139

6.3 Implementation of an OKFDD Package 147

6.3.1 Technical Details
6.3.2 The Construction and Operations on OKFDDs
6.3.3 Optimization of OKFDD-Size

6.4 Experimental Results 153

6.5 Relations Between KFDDs and Two-level AND/XOR Forms 156

6.6 Summary 158

7 Design For Testability Properties of AND/XOR Networks 160

7.1 Introduction 160

7.2 Detection of Stuck-at-Faults in CGRM Networks 162

7.3 Detection of Bridging Faults of CGRM Networks 167

7.4 Detection of Stuck-at-Faults in Mixed-Polarity Networks 169

7.5 Detection of Stuck-at and Bridging Faults in CRMP Networks 170

7.6 Detection of Stuck-at-Faults in Reed-Muller Trees 172

7.7 Summary 174

8 Conclusion 175

References 181

List of Tables

Table		Page
2.1	List of all binary valued functions	18
2.2	Properties of Binary Valued Functions	22
2.3	Minimal Functionally Complete Set of Operations	24
2.4	The only Possible Distributive Properties Among Binary Functions	24
2.5	Examples of Bases for 3-Input Functions	32
2.6	Positive Monoterms of three variables	40
3.1	Cube Literal Intersection	58
3.2	Basis Functions and Basis Function Cubes	59
3.3	Bit-Wise Equivalence Operation Between Two Cubes	61
3.4	Comparison of *CGRM* Realization Programs	69
3.5	Cube Commonality Operator for a Single Bit	77
3.6	Cube Literal Difference	84
3.7	Comparison of the Minimization Programs Against Benchmark Functions	102
3.8	Comparison of the Timings of Minimization Programs Against Benchmark Functions	103
3.9	Difference of Terms for Exact and Heuristic *CGRM*	103
4.1	Two Level AND/OR Compared to Two Level *CRMP* and *ESOP*	114
5.1	Complex Terms for Benchmark Examples	125
5.2	The Effect of Folding	134
6.1	Comparison of optimal OKFDD with optimal OBDD and optimal OFDD with positive Davio-nodes	153
6.2	Comparison of OKFDD with OBDD for certain benchmark functions	154

6.3 Comparison of the Number of Nodes for OKFDD vs OBDD . 155

List of Figures

Figure		Page
2.1.	The AND/XOR Canonical Forms	46
3.1.	Example of a "BILBO" barrel shifter	66
3.2.	All Possible Intersections of Three Sets	80
3.3.	The Exact Algorithm	98
3.4.	The Quasi-Minimal Algorithm	102
4.1.	The Calculation of a minimum *CRMP* from a *CGRM*	112
4.2.	The CANNES Algorithm	113
5.1.	An Example of a Complex Maitra Logic Array	117
5.2.	The Restricted Factorization Algorithm	120
5.3.	An Example of a Row Folding	124
5.4.	The Technology Folding Algorithm	130
5.5.	The Input to Cellular_map	131
5.6.	The Output of Cellular_map	132
6.1.	Example for OKFDD	143
6.2.	Algorithm for XOR-operation	149
6.3.	Algorithm for OKFDD-construction	150
7.1	The cascade network for *RMC*	163
7.2.	Augmentation of the network in Figure 7.1 for detection of OR-bridging faults	168
7.3.	The Reed-Muller Tree for a 3-variable function	173

Chapter 1

Introduction

The new Field Programmable Gate Array (FPGA) technologies and their structures have opened up new approaches to logic design and synthesis. The main feature of an FPGA is an array of logic blocks surrounded by a programmable interconnection structure. Cellular FPGAs are a special class of FPGAs which are distinguished by their fine granularity and their emphasis on local cell interconnects. While these characteristics call for specialized synthesis tools, the availability of logic gates other than Boolean AND, OR and NOT in these architectures opens up new possibilities for synthesis and expands the confines of Boolean Algebra as the sole structure for realization of switching circuits.

Based on the logic block types and their interconnections, the existing FPGAs can be separated into four basic types. These are Look up Table(LUT)-based, Row-based, PLA-based, and Cellular Automata(CA) type FPGAs. While the size of programmable switch, type and granularity of block, and programmable routing differentiates the FPGAs, there is a common thread among all of them, namely it is impossible for them to be used without powerful synthesis tools.

A feature that distinguishes the most FPGAs from the previous technologies is their capabilities to incorporate logic gates other than AND, OR, and NOT. In LUT-based FPGAs such as Xilinx 3000 family, the logic blocks can realize any Boolean function of up to 5 input variables. The ACT3 family of Row-based FPGAs from Actel can realize a whole group of logic families including Multiplexers as well as combinations of AND, OR, NAND, NOR, XOR, XNOR, Implication, Inhibition, etc. The cells in

ATMEL6000 CA-type FPGA can realize twenty-five combinational states produced by AND, NAND, XOR, wire, and Multiplexers. Hence, the synthesis methods which are solely based on AND/OR logic would not yield the most compact realization of the functions in these FPGAs.

The dominant realization up to now, however, has been based on AND/OR primitives. Two-level minimizers such as ESPRESSO [111], MINI [67], and PALMINI [93] have been developed to address the AND/OR minimization problem required for utilization in PLAs. Synthesis tools such as MIS [22], BOLD [18], etc. were later based on Boolean and Algebraic methods such as factorization, decomposition, and rectangle covering to result in reduced multi-level representation of the functions. Rule-based systems such as LSS [35] and SOCRATES [59] have also been devised for multi-level representations of the functions. However, the main feature of these systems is their dependency on AND/OR logic constructs. The only general methods come from the transduction method [92] which provides only local minimization capabilities for functions, and functional decomposition.

The logic synthesis systems mentioned above, however, are inadequate for coarse-grain FPGA synthesis. For one, it is not realistic to generate the complete library, obtained by enumeration of all possible configurations of the basic logic blocks. Furthermore, a library based approach does not allow to benefit from the basic structure and therefore leads to less optimized results. Instead, mapping the design directly into logic blocks of the target architecture proved to be the more viable synthesis method for such FPGAs. Synthesis tools such as mis-pga [90, 91], Chortle [47], ASYL [14], etc. were geared towards such an approach. A different approach in such tools as TRADE [150] uses functional decompositions for mapping.

The above methods still would prove insufficient for Cellular Automata type FPGAs where the local cell communications is a distinct restriction. These types of

FPGAs can be found in such architectures as in Motorola MPA10xx, ATMEL6000 (adopted by IBM), Algotronix CAL1024 (Now Xilinx), Plessey, Toshiba, etc. As indicated, the main features of such FPGAs is their fine-granularity and their emphasis on local cell communications. This restriction in cell communications requires for synthesis stage to take the routing restrictions into consideration. With separate stages of logic synthesis and physical design, the mapping would be very inefficient resulting in many cells unused or used solely for routing.

The current techniques used in the industry either do not provide a systematic approach for mapping general-purpose functions or concentrate solely on the technology mapping only. One such approach is the "macro blocks" used by ATMEL. In this approach basic modules are provided in a library and automatic routing techniques are used for connecting the modules. For one, this technique does not provide any means for synthesis of general purpose functions where decompositions into submodules are not known for. Secondly, the modules are irregular in shape and routing will require many cells to be used just for connections. In this approach, on average, about 70% of the area occupied by the design is used either for simple connections or not used at all [31]. The tool set for MPA10xx from Motorola is yet to be announced by Neocad Corporation, but it is unlikely that any new logic synthesis methods will be utilized in these tools since Neocad concentrates on mapping techniques only.

This dissertation addresses the problem of **efficient synthesis and mapping methodologies for CA-type FPGAs.** The methods proposed essentially are based on the concept of regularity to combine the synthesis and physical placement and routing stages. The synthesis methods are mostly based on XOR logic as XOR gate is available in these FPGAs as a basic logic primitive and is known to result in compact realization of Boolean functions and be highly testable.

Among the methods that is of historical significance is the cellular logic approach.

Cellular logic deals with mathematical models as well as synthesis and analysis techniques of digital networks in cellular arrays. "A *cellular array* is a 1-, 2-, or 3-dimensional iterative arrangements of similar or identical logic cells with a uniform interconnection pattern on the cells" [88]. The limited routability of CA-type FPGAs makes these FPGAs to resemble these arrays which were studied in the sixties and seventies. As such, various arrays, universal logic modules, and synthesis methods have been developed and studied without actual mapping to any device [88, 78, 146, 139, 157, 42, 83, 84, 85, 86, 76, 95, 152].

The cellular arrays can be classified into simple one-dimensional cellular arrays, "Multi-rail cascades", and two-dimensional arrays. The simple one-dimensional arrays are also known as *Maitra Cascades* [78]. A *Maitra cascade*, also known as *tributary switching* network, is a one-dimensional array of 2-input, 1-output binary combinational cells. In this cascade, each cell is capable of producing any one of the sixteen possible binary functions of two inputs. It was shown, however, by Stone and Korenjak [147] that even using *redundant cascades*, in which certain vertical inputs are connected to more than one cell, not all functions are realizable in this cellular array. The same deficiency exists for *generalized Maitra cascades* in which inputs are multi-valued.

One attempt to overcome the logical incompleteness of simple one-dimensional arrays is that of the two-rail cascades . Short [139] has shown that every binary function is realizable by means of 3-input, 2-output cells. In the synthesis methods developed by Short, only one of the final outputs is of interest. Yoeli and Turner [157] extended the treatment of the two-rail cascades to both output signals and showed that two-rail cascades are *functionally complete* for realizing an arbitrary pair of Boolean functions of any number of variables. Among the synthesis methods for two-rail cascades, four major approaches can be noted. These approaches are the ones introduced by Short [139], Yoeli [157], Elspas [43] which generalizes Yoeli, and Dvorak [42]. An extension to

two-rail cascades is that of the multi-rail cascades. Here, instead of two rails, the cells are assumed to have more than two horizontal inputs and outputs. The major shortcoming of the multi-rail cascades is their serial structure which makes them slow and the fact that the methods developed were of exponential growth in the number of variables.

Two dimensional arrays provide another attempt at overcomming the limitation of Maitra cascades. These arrays can be classified into general purpose and special purpose ones. General purpose arrays can be essentially classified into those that realize a Two-level representation of functions and those that realize multi-level realizations. While the Two-level representations fit mapping into arrays with regular structures, multi-level representations can utilize both regular and irregular structures. Special purpose arrays utilize special features of their target structures and include Adder, Multiplier, threshold, sorting, coding, interconnection arrays, etc [88].

Sum of Products and positive polarity AND/XOR forms have been among the Two-level representations of functions which have been investigated before. Some multi-level representations geared towards specialized cellular arrays have also been reported. One such example is the functionally complete *cutpoint arrays* [83]. These arrays are composed of columns of Maitra cascades where each cell needs to realize only six possible functions of two input variables. The "cutpoint" in this array refers to the specification bits in each cell to program the type of the operation it will be performing. The main deficiency of this architecture is the large number of cells that do not perform any actual function. Furthermore, there exists no communication between the input horizontal and vertical signals of the cells.

Variants to cutpoint arrays were introduced to alleviate the difficulty of the synthesis due to the limited interconnections between the cells. One such variance was to add a collector row of Inclusive (or Exclusive) OR gates to the Maitra cascade columns. This approach was mostly used for Two-level Sum of Products and positive polarity

AND/XOR representations rather than the more general multi-level ones. Another attempt at multi-level representation of the functions is that of the *Unate Cellular Logic* by Mukhopadhyay [86]. In this approach the cells are assumed to be unate two-input functions, i.e. all functions except XOR and XNOR. Each cascade in this array can realize a unate function and the whole array is considered to be a two-dimensional arrangement of the unate cascades. In the synthesis method, a test for unate cascade realizability is provided.

Other modifications to the basic structure of the cutpoint arrays have also been introduced to allow more interconnection among the cells. Minnick introduced the *cobweb* array [84] where the cutpoint array is augmented with certain interconnections allowing communication between more cells. In the same line, Akers [2] introduced the "Rectangular Logic Array" where each cell in the cutpoint array receives an additional input from a non-immediate neighboring cell. This in practice makes the array to resemble a three-dimensional structure.

The Directed Acyclic Graph (DAG) structures have also been used as nonrectangular approaches for mapping. The major issue with DAG structures is the number of cells which are wasted due to their shape. They, however, provide a formidable approach which by efficient mapping techniques can prove to be quite useful.

In summary, although the cellular logic approaches described above provide useful techniques that can be incorporated into CA-type FPGA synthesis, they are not adequate by themselves. Many of the techniques were developed for particular architectures which did not have all the features of the current CA-type FPGAs. Among these are local and global busses which add more flexibility to the current arrays. The synthesis methods also did not utilize the modern techniques which make handling of large functions possible.

In this dissertation, in the realm of the current logic synthesis techniques, four basic approaches to CA-type FPGA synthesis are investigated. These approaches are:

i. Two-level Realizations

ii. Universal XOR Forms

iii. Restricted Factorization Technique

iv. DAG structures

The first three are rectangular array shaped and fall under the *Complex Maitra Logic Array (CMLA)* approach. This array is the generalization of the features common to CA-Type FPGAs and can be modified to each specific architecture accordingly. The complex Maitra terms realized include AND, OR, and XOR of literals and are more general than PLAs. In general, a CMLA is a Two-Dimensional Logic Array comprised of two distinct planes:

- The Complex (input) Plane;
- The Collector (output) Plane.

The input variables run through the vertical buses in the Complex plane and the appropriate terms are realized in the rows of the Complex Plane. The terms generated are then put on the horizontal buses and the appropriate terms are XORed (or ORed) together in the Collector Plane. The approach used includes two stages:

1. **Logic optimization** which takes the geometry and layout constraints into account to create a CMLA in which every output function is an OR or XOR of Maitra terms.
2. **Technology-folding** which maps CMLA representation of the function to the target architecture, such that the area of the layout is minimized.

In this dissertation, the logic optimization stage based on Two-level AND/XOR representations as well as two multi-level approaches of Universal XOR Forms and

Restricted Factorization [143] is introduced. The technology folding stage is introduced with the emphasis on the special architecture of ATMEL6000.

The main focus of the Two-level approaches here is, as stated before, on AND/XOR realizations of Boolean functions. The Boolean AND/OR representations have long been the subject of investigation and efficient minimization schemes have been developed for this representation. The AND/XOR representations on the other hand have only received more attention lately due to the existence of technologies which makes their use more practical. They are, however, known to possess special characteristics which makes them of major advantage in circuit representations. In many applications, the AND/XOR realizations of the circuits require less layout area than their AND/OR counterparts [101, 142, 122]. Many such applications can be found in arithmetic, encoding, telecommunication, and linear systems. It has been shown also that the AND/XOR PLAs often require fewer products than AND/OR PLAs [120]. The major advantages of this logic stem from the information processing capabilities of XOR gate and what is termed the *computational work* [62]. These studies have shown that XOR gate has the highest efficiency of all gates in terms of the useful work.

Another major advantage of AND/XOR logic is its high testability properties [109, 105, 118]. A major characteristic of the XOR gate is that any change at its inputs is reflected on the output. This characteristic is already used in many testing schemes. This testability is inherent in the AND/XOR networks. For certain class of these networks, called Reed-Muller networks [110, 89], it has been shown that there exist universal test sets, independent of the function, that can detect both single and multiple stuck-at-faults as well as various bridging faults. Other universal test sets also exist for various other AND/XOR networks which are generally of higher cardinality than the Reed-Muller networks. With respect to the NP-completeness of the test generation scheme for AND/OR networks [52], this is an especially important trait of this logic which is described in

more detail in chapter 7.

As Two-level realizations perfectly fit the array structure of the CA-type FPGAs, these realizations are first investigated in the dissertation. In particular, fixed polarity AND/XOR canonical forms and Generalized AND/XOR canonical forms are studied. The fixed polarity forms, including the positive polarity Reed-Muller forms, are the most basic of the AND/XOR forms. Due to their very high testability properties and the fact that they can be used in the minimization of the functions in other more general forms, the fixed polarity forms are first studied. Specifically, fast techniques for the identification of a minimal realization of Boolean functions in these forms are introduced. This scheme is then used in a Generalized AND/XOR canonical form minimization technique to identify even more compact representations of functions.

In addition to the Two-level realizations, two multi-level approaches are introduced and presented. The first approach introduced is that of the Universal XOR forms as Boolean techniques which provide more global minimized realizations at the expense of more processing requirements. The second approach is that of the Algebraic Factorization techniques which are more local but can be performed more efficiently.

Universal XOR forms (UXF) are nothing but all possible canonical realizations of Boolean functions which are based on Exclusive sum of Maitra terms. In this thesis, a new comprehensive methodology for the investigation of all possible XOR canonical forms is introduced. The methodology is based on the fact that the set of n-variable Boolean functions under addition mod-2 forms a 2^n-dimensional vector space over the Galois field of two elements, GF(2). It is then possible to represent any XOR canonical form as a basis in this vector space. In the following chapter, the basic traits of UXF as an approach for CA-type FPGA synthesis and mapping are provided along with the introduction of several new AND/OR/XOR canonical forms.

The product terms of of Two-level AND/XOR forms and Sum of Products, as well as the terms in the UXF can be directly mapped to the Complex plane of the CMLA without any changes necessary. While the UXF can result in more reduced realizations than the Two-level representations, the identification of a minimal UXF requires a large search space. For this reason, Algebraic factorization techniques are often preferred for fast multi-level realizations. The limited interconnection of the cells in CA-type FPGAs, however, would make general factorization techniques to be extremely inefficient.

To overcome this restriction, certain factorization techniques have been proposed which take the architecture restrictions into consideration. One such technique is that of the Lexicographical ordering technique [125]. This technique is solely based on AND/OR paradigm and also requires a separate place and route stage afterwards which can be done through COMPASS design automation tool as an example. A different technique which is based on the more general complex terms [143] is that of the restricted factorization. In this dissertation, a fast restricted multi-level realization of Boolean functions is devised based on the latter approach. By identifying an ordering for the input variables, it will be then possible to directly map the factorized terms to the CA-type FPGA of interest.

Once the Logic optimization stage has been performed, the technology folding is applied on the Complex plane to further economize the cell utilization in that plane. The folding stage is general for CA-type FPGAs while taking the peculiarities of each architecture into consideration. In chapter 5, the method is shown for the ATMEL6000 series of FGPAs. It is demonstrated that this stage can reduce the number of cells in the plane by up to 33%.

The DAG approaches to CA-type FPGA synthesis have been applied previously. While Hurst [65] mentions the approach, possible synthesis tools have been reported in [155] and [131]. In this dissertation the new class of Decision Diagrams, called

Kronecker Functional Decision Diagrams (KFDD) is presented. These Decision Diagrams are the generalization of the popular Binary Decision Diagrams [23] and Functional Decision Diagrams [72] and are more compact than both of the former Decision Diagrams. While FDDs have been used in [155] and KFDDs in [131], the method introduced here is much more efficient and can be applied to very large functions given as multi-level netlists.

In addition, KFDDs, similar to BDDs and FDDs, provide a canonical representation of the functions and can be applied to areas far broader than the CA-type FPGA synthesis. Currently, BDDs have been used in many applications in logic synthesis, verification, testing, modeling and simulation. KFDDs while being more compact, can also be utilized in many of such applications and thus can provide a major improvement over the current techniques in these areas. They can also drastically cut on the number of nodes in the Decision Diagram for very large functions that up to now have not been able to be presented by BDDs.

For any Decision Diagram to prove to be useful, the compactness of the representation has to be compared with the ease of construction and manipulation. Here, a package for representation and manipulation of functions is presented - a joint project [40, 41] with colleagues in J. W. Goethe University of Frankfurt - which shows the compactness of the KFDDs together with ease of manipulation and construction. It is shown that for the standard, hard, benchmark examples, KFDDs are on average 35% more compact than Binary Decision Diagrams, with some reductions of up to 75% being observed. The minimization scheme is based on the state of the art minimization schemes for BDDs, namely, dynamic variable ordering with sifting algorithm [112]. Here the sifting is performed for both the order of variables as well as the type of decompositions.

Furthermore, a class of functions is presented for which both BDD and FDD representations are exponential in size but KFDD is of polynomial size [41]. This pro-

perty together with the canonicity and the ease of construction and manipulation distinguishes the major significance of the KFDDs.

A note has to made on the evaluation of methods developed in the dissertation. The developed synthesis and mapping tools are evaluated on the basis of the standard benchmarks from Microelectronic Center of North Carolina (MCNC). This is the current state of evaluation in logic synthesis community and as such the tools developed have been evaluated based on their performance on these benchmarks.

The synthesis tools developed are integrated as part of the POLO - Portland Logic Optimization - package which is geared towards multi-level AND/OR/XOR representation and manipulation of the functions. This package is interactive and is interfaced to the SIS [133], and will be to HSIS [5] in a later phase, packages from UC Berkeley and makes it possible to utilize synthesis capabilities of both systems. While the former is geared more towards XOR logic, the latter is concentrated mostly on AND/OR representation and manipulation of the functions. POLO can also be used as the logic synthesis component of the DIADES [97] high level synthesis package developed at Portland State University.

In the following chapters the above mentioned approaches are presented. In chapters 2 through 5 different logic optimization techniques in CMLA are introduced. Chapter 2 is devoted to the Boolean representation of the functions where the new Universal XOR forms, all possible XOR canonical forms, and their subset of AND/OR/XOR canonical forms have been introduced. In addition, various Two-level AND/XOR realizations are reviewed. In chapter 3 a fast cube-based method for fast minimal realization of functions in fixed polarity AND/XOR canonical forms is described. Following that, an application of the tools developed in chapter 3 is shown for minimal realization of Boolean functions in Generalized AND/XOR canonical forms. The application of these synthesis methods as well as the restricted factorization are

presented in chapter 5. The technology folding stage is also introduced in that chapter. Kronecker Functional Decision Diagrams and their manipulation as well as basic properties are given in chapter 6. In chapter 7 the testability of AND/XOR networks is discussed.

In summary, the dissertation introduces new concepts in synthesis and mapping for CA-type FPGAs based on XOR logic which have applications beyond just the synthesis for these FPGAs. The synthesis methods introduced by this author alleviate the shortcomings of the previous techniques by providing synthesis for any general purpose function. They also take full advantage of these architectures not being confined to AND/OR logic. As these methods incorporate regular structures, they alleviate the need for a separate physical design stage which is a major advantage for the restricted interconnections within these architectures.

The new concept of UXF provides the framework for investigation of a larger possibilities of XOR representation of the functions. In particular, the AND/OR/XOR canonical forms prove to be quite useful and more compact than either AND/OR or AND/XOR representations of the functions. The minimization schemes for the Two-level representations of fixed polarity and generalized AND/XOR forms are also new and provide efficient methods for reduced representation of functions in these highly testable realizations. Also new in the dissertation are the methodologies for the realization of Boolean functions in restricted factorized form and the concept of technology folding.

The popularization of the concept of the Kronecker Functional Decision Diagrams as well as the application of the dynamic variable ordering with sifting, introduced by Rudell, to KFDDs are the contributions of the author. The iterative generation of the Davio nodes were independently developed by the author and the colleagues at J. W. Goethe University. The generation of the package is due to these colleagues.

Also new in this dissertation is the testability analysis and algorithms for fixed polarity and the Generalized AND/XOR forms as well as the Reed-Muller trees.

Chapter 2

Boolean Functions and their Representations

2.1. Introduction

A distinguishing feature of the new Field Programmable Gate Array technologies is that they break the confines of Boolean Algebra as the sole representation for the Boolean functions. Not only the traditional primitives of AND, OR, and NOT can be utilized in these technologies, but for some FPGAs the cost of utilizing other gates is the same as the above mentioned primitives. Hence, a study and actual practice of incorporation of other gates into the synthesis is of more importance now than ever before.

A certain criterion for a set of logic operations to be used in representation of the functions is for the set to be functionally complete. That is, every Boolean function should be able to be represented by the operations in the set.

Once an appropriate set of logic operations is identified for usage in the technology, the realization in that set becomes of importance. A given Boolean function can have numerous realizations in a given set. Based on the optimization criteria, one or more realizations can be chosen among different possibilities. An example would be one that would require the least number of a selected type of operation in the representation. Minimal number of products for Programmable Logic Areas has been one such example in the case of Two-level synthesis. A similar problem also exists for the realizations in different sets of logic operations.

Parts of Sections 2.4 and 2.5 have been based on the original paper, M. A. Perkowski, A. Sarabi, F. R. Beyl, Universal XOR Canonical Forms of Switching Functions, IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg, Germany, September 1993.

An important representation of Boolean functions is that of the canonical representations. In this form, the functions can be uniquely represented thus making it possible for fast verification and equivalency checking.

A major possible representation is that of the XOR logic. The advantages of XOR logic in terms of the compactness and high testability were mentioned in chapter 1. Based on these properties and the fact that in Cellular FPGAs, the cost of XOR gate is mostly the same as any other gate, this logic would be of more interest in this dissertation. In sections 2.5 and 2.6, a new methodology to investigate all possible XOR canonical representations of Boolean functions is introduced.

In this chapter, the representation of Boolean functions is mainly discussed. In section 2.2, Boolean function and its different structural representations will be discussed. Section 2.3 is geared towards functional completeness and functionally complete set of operations. In section 2.4, XOR canonical forms of Boolean functions will be described and in section 2.5 different AND/XOR canonical forms will be presented.

2.2. Models of Switching Circuit Behavior and Structure

Boolean functions are a useful mathematical model for representing behavior and structure of switching circuits. In this section a Boolean function is first defined and later on its relevance to switching algebra is demonstrated.

2.2.1. Boolean Functions and Model of Behavior

The behavior of a given system is designated by abstracting the system as a black box where the internal structure is not of concern and only the mapping from the inputs to the outputs of the system is of interest. This mapping, in terms of the switching circuits is a *Boolean function.*

Boolean functions are special cases of discrete functions. Depending on the number of input variables, their values, and the number of outputs, Boolean functions are

classified in different ways. The following definitions, mainly from Davio et al [38], put binary multi-variable Boolean functions - the functions of interest in this dissertation - into perspective:

Definition 2.1. Let S and R denote two finite non-empty sets. Then

$$f : S \rightarrow R \tag{2.1}$$

is a *discrete function.*

The function $f : S \rightarrow R$ will be denoted as $f(x)$ where the variable x takes its values from the set S and $f(x)$ takes its values from the set R.

When the domain of the function, S, is the Cartesian product of n finite sets, S_i, the function

$$f : \underset{i=0,n-1}{\times} S_i \rightarrow R$$

can be denoted by $f(\mathbf{x})$, where $\mathbf{x} = (x_0, x_1, ..., x_{n-1})$, each x_i taking its value from the set S_i.

If the elements of the sets S_i and R are integers, the above function will be termed an *integer function.* It is then the mapping:

$$f : \underset{i=0,n-1}{\times} [0, 1, ..., s_i - 1] \rightarrow [0, 1, ..., r - 1],$$

where s_i and r are the *cardinalities* of the sets S_i and R respectively.

Definition 2.2. An integer function where the sets S_i and R have the same cardinality is a *logic function.*

Then the logic function is the mapping:

$$f : [0, 1, ..., r - 1]^n \rightarrow [0, 1, ..., r - 1].$$

where r is the cardinality of the sets S_i and R.

Definition 2.3. A logic function where the cardinality of the sets S_i and R is 2 is a *Boolean function.*

A Boolean function is then the mapping:

$$f : [0, 1]^n \rightarrow [0, 1]. \tag{2.2}$$

At times generalizations of the term "Boolean function" is used in the literature. A *multi-valued* Boolean function is a logic function where the cardinality of the domain and range sets is more than two. A *multi-output* Boolean function is one that the range of the logic function is a Cartesian product of the set R. That is:

$$f : [0, 1]^n \rightarrow [0, 1]^m.$$

The most general logic function is the multi-valued, multi-output, and multi-variable one. The logic functions mostly considered in this dissertation are, however, that of the multi-variable Boolean functions. These functions can be related to the propositional logic in formal logic.

For n variables, each being binary, there are 2^n possible states. Since there are also 2 possible mappings for each of these states, the total number of Boolean functions is 2^{2^n}. Table 2.1 shows all the possible 16 binary valued functions of two variables. For reference, Boolean representations of these functions are also included.

f_n	$f(x_1x_2)$	Name of function	Symbol
f_0	$\underline{0}$	Inconsistency	0
f_1	$x_1 \vee x_2$	NOR	$x_1 \downarrow x_2$
f_2	$\bar{x}_1 x_2$	Inhibition by x_2	$x_1 < x_2$
f_3	$\bar{x}_1$	NOT x_1	$\bar{x}_1$
f_4	$x_1 \bar{x}_2$	Inhibition by x_1	$x_1 > x_2$
f_5	$\bar{x}_2$	NOT x_2	$\bar{x}_2$
f_6	$\bar{x}_1 x_2 \vee x_1 \bar{x}_2$	Exclusive OR(XOR)	$x_1 \oplus x_2$
f_7	$x_1 x_2$	NAND	$x_1 \uparrow x_2$
f_8	$x_1 x_2$	AND	$x_1 x_2$
f_9	$x_1 x_2 \vee \bar{x}_1 \bar{x}_2$	Equivalence(XNOR)	$x_1 \equiv x_2$
f_{10}	x_2	Assertion of x_2	x_2
f_{11}	$\bar{x}_1 \vee x_2$	Implication from x_2	$x_2 \Rightarrow x_1$
f_{12}	x_1	Assertion of x_1	x_1
f_{13}	$x_1 \vee \bar{x}_2$	Implication from x_1	$x_1 \Rightarrow x_2$
f_{14}	$x_1 \vee x_2$	OR	$x_1 + x_2$
f_{15}	1	Tautology	1

Table 2.1 List of all binary valued functions

The *complement* of a Boolean function is also of importance. The complement of a Boolean function $f(x_1, x_2, ..., x_n)$, denoted by $\overline{f(x_1, x_2, ..., x_n)}$ is a function which takes a value of 0 whenever $f(x_1, x_2, ..., x_n)$ takes a value of 1 and takes a value of 1 when the former takes a value of 0.

2.2.2. Switching Expressions and Models of Structure

Boolean function is generated by a *switching expression.* This is an expression that represents each output as a function of a set of inputs. The switching expression can not only represent a Boolean function - the model of the behavior of the switching circuits - but can also be used as a suitable model for the *structure* of the switching circuits. A note of distinction is that while the behavior for a given value of the input set is unique, there can be different structures displaying the same behavior. However, a given structure will result in one and only one behavior given the same input set. Then, for every output variable, there will be a corresponding switching expression for a given structure. A formal definition of a switching expression is:

(a) The constants 0, 1, ..., m - 1, where m refers to the maximum value a variable can take.

(b) All the variables and the specific logic functions applied to the variables in the algebra.

(c) For A and B being any switching expression, $A\ f_i\ B$, f_i representing any of the logic functions described in (b).

(d) No other form will be a switching expression.

As it can be noticed, the switching expression can result in multi-valued and multi-level representations of the function. A special case of these expressions is for binary, Two-level representations which will be discussed in detail later on.

Switching algebra is one method of representing the structure of switching circuits. Development of this algebra is generally attributed to Claude Shannon. In his paper [136], he developed four postulates as the calculus of switching circuits for series-parallel, two terminal circuits. Comparing these postulates with Huntington's set of postulates for symbolic logic, he showed the analogy between the calculus of switching circuits and the symbolic logic (or the calculus of propositions). Calculus of propositions was thus shown to be another method for representing switching circuits. As the calculus of propositions is one interpretation of Boolean algebra, this algebra provides another possible representation for the structure of the switching circuits. Yet another representation is through Boolean rings with unit. It has been known [145] that Boolean algebras and Boolean rings with unit can be transformed to one another. In the language of formal logic, AND and OR together with NOT are the operators in Boolean algebra while XOR and AND are the operators in Boolean rings with unit,

These two algebras are not the only possible algebras that can represent the behavior of a switching circuit. The choice of operators for a structural representation of a circuit also depends on the exact type of gates that are used in realization of the circuit. NAND, NOR as well as AND, OR, and XOR have been the most often used gates for realizing switching circuits. With the new FPGA technologies it is possible to utilize other gates too. The criteria for a given algebra to be used as a switching expression is reviewed in the following section.

2.3. Functional Completeness

In general, for any algebra to be used as a switching expression, its operations have to be able to generate all possible functions. Otherwise, the algebra would be incomplete.

Definition 2.4. A set of operations is said to be *functionally complete*, or *universal*, if and only if every Boolean function can be expressed entirely by means of operations

from this set.

The complete set of operations can be further distinguished as strong and weak.

Definition 2.5. A set of operations is said to be *strong functionally complete* if any arbitrary Boolean function $f(x_1, x_2, ..., x_n)$ can be realized entirely by these operations. If the set of operations can realize all Boolean functions except constants then the set of operations is said to be *weak functionally complete.*

As summarized by Klir [73], the necessary and sufficient conditions for functionally complete set of operations were derived by Yablonskii [156]. Before the main conditions are presented in terms of Theorem 2.1, a few properties of the binary operations are reviewed.

Definition 2.6. [Monotonicity] Let $A = <a_1, a_2, ..., a_n>$ and $B = <b_1, b_2, ..., b_n>$ be two n-tuples where $a_i, b_i = 0$ or $1 \; \forall \; i = 1, 2, ..., n$. If $a_i \leq b_i \; \forall \; i$, then $A \leq B$. A function $f(x_1, x_2, ..., x_n)$ is said to be *monotonic* if and only if $f(A) \leq f(B) \; \forall \; A \leq B$.

Definition 2.7. [Linearity] A function $f(x_1, x_2, ..., x_n)$ is said to be *linear* if it can be expanded in the form $f(x_1, x_2, ..., x_n) = a_0 \oplus a_1 x_1 \oplus ... \oplus a_n x_n$, where $\oplus$ stands for the ring sum and a_i is either 0 or 1; $0 \leq i \leq n$.

Definition 2.8. [Self-Duality] A function $f(x_1, x_2, ..., x_n)$ is said to be *self-dual* if

$$f(x_1, x_2, ..., x_n) = \overline{f(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n)} \tag{2.3}$$

Definition 2.9. [Zero-Preservation] A function $f(x_1, x_2, ..., x_n)$ is said to be a function *preserving zero* if

$$f(0, 0, ..., 0) = 0. \tag{2.4}$$

Definition 2.10. [One-Preservation] A function $f(x_1, x_2, ..., x_n)$ is said to be a function *preserving one* if

$$f(1, 1, ..., 1) = 1. \tag{2.5}$$

Table 2.2 shows all binary-valued functions and their properties. Among the six-

teen operations, a set would be of interest that is functionally complete.

The following theorems give the criteria for a set of operations to be functionally complete:

Theorem 2.1. [Yablonskii] A set of operations is strong functionally complete if and only if it contains

(1) at least one non-monotonic operation,

(2) at least one nonlinear operation,

(3) at least one non-self-dual operation,

(4) at least one non-zero-preserving operation,

(5) at least one non-one-preserving operation.

For proof see [156].

f_n	Name of function	S	M	L	SD	Z	O	C	A
f_0	Inconsistency	0	1	1	0	1	0	–	–
f_1	NOR	$x_1 \downarrow x_2$	0	0	0	0	0	1	0
f_2	Inhibition	$x_1 < x_2$	0	0	0	1	0	0	0
f_3	NOT	$\bar{x}_1$	0	1	1	0	0	–	–
f_4	Inhibition	$x_1 > x_2$	0	0	0	1	0	0	0
f_5	NOT	$\bar{x}_2$	1	0	0	1	1	–	–
f_6	XOR	$x_1 \oplus x_2$	0	1	0	1	0	1	1
f_7	NAND	$x_1 \uparrow x_2$	0	0	0	0	0	1	0
f_8	AND	$x_1 x_2$	1	0	0	1	1	1	1
f_9	XNOR	$x_1 \equiv x_2$	0	1	0	0	1	1	1
f_{10}	Assertion	x_2	1	1	1	1	1	–	–
f_{11}	Implication	$x_1 \Rightarrow x_2$	0	0	0	0	1	0	0
f_{12}	Assertion	x_1	1	1	1	1	1	–	–
f_{13}	Implication	$x_2 \Rightarrow x_1$	0	0	0	0	1	0	0
f_{14}	OR	$x_1 + x_2$	1	0	0	1	1	1	1
f_{15}	Tautology	1	0	1	0	0	1	–	–

1 stands for the function possessing the property and 0 otherwise. M stands for Monotone, L for Linear, SD for Self-Dual, Z for Zero-Preserving, O for One-Preserving, C for Commutative, and A for Associative. – denotes Don't care as the properties are of significance for more than one variable.

Table 2.2 Properties of Binary Valued Functions

Theorem 2.2. A set of operations is weak functionally complete if and only if it contains:

(1) at least one non-monotonic operation,

(2) at least one nonlinear operation.

Definition 2.11. A complete set of operations is a *minimal complete set of logic operations* if by omitting any one of its operations it becomes incomplete.

The operation with smallest number of arguments is the unary operation which takes only one variable as its argument. Among all the unary operations, the only non-trivial one on the set (0, 1) is the NOT which by itself is not functionally complete. It is known that unary operations are linear and by Theorem 2.1, they are not functionally complete. Therefore, any functionally complete set of operations should include some binary operations.

It can be seen from Table 2.2 that each of the two binary operations NAND, and NOR, is by Theorem 2.1 a minimal complete operation. Table 2.3 shows all combinations of logic operations involving one or two arguments which result in a minimal complete set of operations.

In Table 2.3, the set of operations 3, 4, 12, 13, 14, and 15 are weak functionally complete while the rest are strong functionally complete. Furthermore, for Boolean functions, there are no minimal complete sets of operations with more than three functions.

The order of operations in each row of Table 2.3 is also important. In other words, if the order of two operations is interchanged, the result would no longer be necessarily a complete set of operations. This can be seen as an example in the case of AND, XOR, and Tautology. XOR of ANDs of variables is functionally complete while AND of Exclusive OR of variables is not.

No.	Minimal Complete Set of Operations
1	NAND
2	NOR
3	Implication-Inconsistency
4	Inhibition-Tautology
5	Implication-Inhibition
6	Inhibition-NOT
7	AND-NOT
8	OR-NOT
9	Implication-NOT
10	Inhibition-XNOR
11	Implication-XOR
12	AND-XNOR-Inconsistency
13	OR-XNOR-Inconsistency
14	AND-XOR-Tautology
15	OR-XOR-Tautology
16	AND-XNOR-XOR
17	OR-XNOR-XOR

Table 2.3 Minimal Functionally Complete Set of Operations

In practice, only those complete sets of operations are used to represent Boolean functions that are easy to work with. These would be the operations which have properties that can be utilized in the simplification of functions. Referring back to Table 2.2, it can be seen among all possible binary operations, there are only six which are commutative. However, two of them - NOR and NAND - are not associative. This makes them not as useful in actual process of minimization and manipulation of Boolean functions although they are minimal functionally complete and the dominant gates used in VLSI. There remains four operations, namely AND, OR, XOR, and XNOR which are both commutative and associative. Moreover, there exist only four possible distributions possible which again involves above four operations. These distributions, as noted by Calingaert [29] are shown in Table 2.4.

No.	Distribution
1	$x_1 \cdot (x_2 + x_3) = (x_1 \cdot x_2) + (x_1 \cdot x_3)$
2	$x_1 + (x_2 \cdot x_3) = (x_1 + x_2) \cdot (x_1 + x_3)$
3	$x_1 \cdot (x_2 \oplus x_3) = (x_1 \cdot x_2) \oplus (x_1 \cdot x_3)$
4	$x_1 + (x_2 \equiv x_3) = (x_1 + x_2) \equiv (x_1 + x_3)$

Table 2.4. The only Possible Distributive Properties Among Binary Functions

AND, OR and NOT are the Boolean operations and the bulk of switching theory deals with Boolean algebra. AND and XOR are operations of Boolean ring with unit and are more investigated in this dissertation. The XNOR and OR have been studied by Mukhopadhyay and Schmitz [87]. Sheffer [137] gave a description of NAND , which he called "stroke" and its dual NOR. As it is noted in [153], Whitehead and Russell used NOT and OR in their treatment of *Principia mathematica.* Hillbert and Ackerman used the same two operations as well as the pair NOT and AND, or NOT and Implication. Some studies have also been reported on other functionally complete sets of operations. For example see [140] for Implication and Inhibition, and [141] for Implication and XOR.

Representing a certain Boolean function by a functionally complete set of operations can take different forms. One standard form of representation is called a *canonical form.* In order to give the appropriate definition of a canonical form, a few definitions are in order.

Definition 2.12. A *literal* is a variable or, in the case that negation is present as one of the operations, its complement.

Definition 2.13. A *term* is a sequence of literals which are related by one and only one operation.

Definition 2.14. A *normal form* is a sequence of terms which are related by one and only one operation.

A special normal form is the canonical form.

Definition 2.15. A *canonical form* is a normal form in which other forms can be reduced to.

The bulk of the work on canonical forms has been concentrated on Boolean algebra, and more recently on Boolean rings with unit. The latter is mostly known as AND/XOR canonical forms or Reed-Muller forms. One canonical form has been

reported for Implication and NOT.

The uniqueness provided by these forms supports many useful applications. One use of the canonical forms is in the enumeration and comparison of functions. As each function has one and only one unique representation, canonical forms find many useful applications in verification.

In Boolean algebra sum of minterms and product of maxterms provide canonical representations of functions. In addition, for each of these two Boolean canonical forms, new canonical forms can be defined which are equivalent to them and are derived by application of the DeMorgan's theorem to each form. In the next section, a framework for generation of all possible XOR canonical forms will be introduced.

2.4. Universal XOR Canonical Forms and their Number

It has long been known that the set of n-variable Boolean functions under addition mod-2 forms a 2^n-dimensional vector space over the Galois field of two elements, *GF(2)* [134, 106, 29, 147]. The vector space representation makes it possible to investigate all possible XOR canonical forms of Boolean functions.

Each basis in the vector space Ψ over $GF(2)$ formed by the set of n-variable Boolean functions under addition mod-2 is comprised of 2^n vectors. Once a basis has been chosen, its vectors are called *basis functions*. Thus every Boolean function can be represented uniquely as a linear combination of the basis functions. The task of the identification of all canonical forms of the Boolean functions in this field is same as the identification of all possible bases of the 2^n-dimensional vector space Ψ. In the following, a systematic method of identifying all of these bases will be presented.

First, as an example, one can start with the minterm canonical form presented in the previous section. The 2^n minterms of the function provide a basis for this vector space and each minterm will be a basis function. Any Boolean function can then be

uniquely represented as a linear combination of these minterms.

Furthermore, it is possible to define a transition from one basis to another through an appropriate transition matrix. In general, in the space Ψ of Boolean functions, any nonsingular matrix of dimension 2^n provides the transition matrix to a new basis. All bases can be identified through the utilization of appropriate nonsingular matrices. Each basis in this vector space is a *Universal XOR form (UXF)* defined in the following:

Definition 2.16. Let Ψ be a vector space of n-variable Boolean functions over *GF* (2). A *Universal XOR form (UXF)* is a basis in this vector space. If a basis function in a *UXF* can be realized as a product of literals, it is called a *monoterm.*

In *GF* (2), the linear combination of the basis functions is the same as XORing of these terms. Hence, identification of all possible bases in this vector space translates into identification of all possible XOR canonical representations of a Boolean function.

Although the vector space representation of Boolean functions has long been known, only the basis functions which can be realized by product of literals have received attention. Here, this concept is generalized to its logical conclusion by extending it to all possible XOR canonical forms. As there exist certain basis functions which can not be realized only by product of literals, the *UXF* is a much broader concept than AND/XOR canonical forms.

The unique representation of a function in a given basis is through the exact basis functions that will appear with a coefficient of 1 in the linear combination. These basis functions are distinguished from all basis functions in the given basis by the term "uxf-term" defined below:

Definition 2.17. Given a *UXF* and a Boolean function f, all the basis functions which appear with a coefficient 1 are called the *uxf-terms* of f.

In order to identify the number of all possible XOR canonical forms for a given Boolean function of n-variables, the transition matrices, given in Appendix A, can be

used. As indicated in Appendix A, one basis can be related to another through the transition matrix. Starting from the minterm basis, every possible nonsingular matrix will define a transition matrix from the minterm basis to a new basis. These matrices form a group called the *general linear group* and it is possible to use this group to identify all possible bases of Ψ for an n-variable Boolean function.

Definition 2.18. The group of all nonsingular m-by-m matrices with entries in the field k is called the *general linear group* and denoted by $GL_{m(k)}$ [4].

The number of such matrices is given in the following Lemma:

Lemma 1 Let $k=GF(q)$ be the Galois field with q elements. The order of $GL_{m(k)}$ is

$$q^{m(m-1)/2}\prod_{i=1}^{m}(q^i-1). \tag{2.6}$$

Proof: See Theorem 4.11 in [4].

Theorem 2.3. Let $f(x_1, x_2, ..., x_n)$ be a Boolean function of n variables. The number of all possible XOR canonical representations of the function is given by:

$$\frac{2^{(2^n-1)(2^n-1)}}{2^n!}\prod_{i=1}^{2^n}(2^i-1). \tag{2.7}$$

Proof: Substituting $q=2$ and $m=2^n$ in Lemma 1, the number of such matrices would be:

$$2^{(2^n-1)(2^n-1)}\prod_{i=1}^{2^n}(2^i-1)$$

for this special case. This is the number of all ordered bases. As the order of basis functions is not relevant to the canonicity of the expansion, it is the number of unordered bases that is of interest here. Hence, the number of canonical forms is given by the number of unordered bases which is the above quantity divided by $m!=2^n!$. *QED*

By Theorem 2.3, there exist 20160/4! = 840 different XOR canonical forms for a 2-variable function alone. This number for a 3-variable function is around 1.326×10^{14}.

As it is evident, the number of canonical forms grows astronomically with the number of variables in this field. Up until now, only AND/XOR canonical forms have been studied in the literature. As UXF in general can include gates other than just AND and NOT and encompasses much larger number of forms, it is more likely to find a minimal circuit among them.

There are two problems of practical interest here. The first is to find such families of forms which have direct mapping to a given technology. The second is to find the form(s) among these families which require a minimal number of uxf terms for a given function.

The uxf-terms can be classified according to their realization requirements. In the existing technologies, some terms are more interesting than the others due to their possibility of direct mapping. These terms will be separately defined in the following:

Definition 2.19. A *Sum* term is a term which can be realized by only OR gates.

Definition 2.20. An *AND/OR* term is a term which can be realized by only AND and OR gates.

In the next section, two operations that generate different AND/OR terms will be introduced.

2.5. Generation of Different Families of AND/OR Bases

AND and OR gates occur in many technologies; hence, developing methods for generation of various bases which can be realized with AND and OR terms would be most useful in this regard. In the following, certain operational transforms on matrices to generate different monoterms and AND/OR terms will be described. The terms with positive polarities will be discussed first with more general terms, incorporating NOT gates, following. The utilization of these forms for cellular arrays will be discussed in chapter 5.

2.5.1. Positive Polarity αρ Family of Bases

Different positive polarity AND/OR bases can be generated by application of two basic operations in various orders. These two operations are called the Reed-Muller and the AND/OR operators. From now on, the basis of reference consists of the minterms in reverse binary order with reversed bits.

Definition 2.21. Let R be a nonsingular matrix. The *Reed-Muller Operator*, ρ, on R is:

$$\rho(R) = \begin{bmatrix} R & 0 \\ R & R \end{bmatrix} \tag{2.8}$$

Here **0** stands for a square matrix of the size of R with all entries 0 .

Definition 2.22. Let R be a nonsingular matrix. The *AND-OR Operator* α on R is:

$$\alpha(R) = \begin{bmatrix} R & 0 \\ 1 & R \end{bmatrix} \tag{2.9}$$

Here **1** stands for a square matrix of the size of R with all entries 1 and **0** has the same meaning as above.

Theorem 2.4. Reed-Muller and AND-OR operators result in nonsingular matrices of a higher dimension.

Proof: This follows from the fact that both ρ (R) and α (R) are block triangular matrices of the form

$$\begin{bmatrix} R & 0 \\ * & R \end{bmatrix} \tag{2.10}$$

with determinant $det(R)^2$. The value of the matrix denoted by * is irrelevant. *QED*

The starting transition matrix for a single variable, used in the generation of the positive polarity αρ family of bases, is:

$$T_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{2.11}$$

This matrix essentially gives the basis

$$\begin{bmatrix} a \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ \bar{a} \end{bmatrix}. \tag{2.12}$$

A special case of applying the ρ operator is the generation of the Reed-Muller Transform. In this case, the repetitive application of the ρ operator is the same as the Kronecker product of the generated nonsingular matrices.

Table 2.5 shows the basis functions of the Reed-Muller, AND/OR, Reed-Muller/AND/OR, and AND/OR/Reed-Muller expansions for three input variables.

The Reed-Muller/AND/OR expansion is constructed by $\alpha(\rho(T_1))$. Other similar constructs are possible - incorporating different orders of application of α and ρ operators which give rise to various AND/OR/XOR canonical forms. While the order of variables is irrelevant for Reed-Muller basis, for AND/OR and all other combinations of the α and ρ operators, it gives rise to a new basis.

Definition 2.23. The family of bases generated by applications of α and ρ operators in all possible orders and all possible permutations of the variables is the *positive polarity αρ family of bases.*

Example 2.1. The nonsingular transition matrix of AND/OR expansion for a 2-input function is given as:

$$\alpha[T_1] = \begin{bmatrix} T_1 & 0 \\ 1 & T_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

The transition matrix above results in the following basis functions:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} ab \\ \bar{a}b \\ a\bar{b} \\ \bar{a}\bar{b} \end{bmatrix} = \begin{bmatrix} ab \\ b \\ a + b \\ 1 \end{bmatrix} \cdot \square$$

Reed-Muller	AND/OR	Reed-Muller/AND/OR	AND/OR/Reed-Muller
abc	abc	abc	abc
bc	bc	bc	bc
ac	$(a+b)c$	ac	$(a+b)c$
c	c	c	c
ab	$ab+c$	$ab+c$	ab
b	$b+c$	$b+c$	b
a	$a+b+c$	$a+c$	$a+b$
1	1	1	1

Table 2.5 Examples of Bases for 3-Input Functions

As it can be seen, by different order of application and also choice between Reed-Muller and AND/OR operators, it is possible to generate many different positive polarity AND/OR/XOR canonical forms. This notion can be further expanded by additionally introducing the negative polarities.

2.5.2. Consistent Generalized αρ Family of Bases

It is possible to generalize all members of the αρ family of bases to 2^n different fixed polarities. This family will be called the *Consistent Generalized* αρ (CGαρ) *family of forms.*

In order to introduce negation of variables, two negation operations and a new starting transformation matrix need to be introduced. Notice that similar to Equation (2.11), it is possible to define a negative polarity basis of a single element. This is given as T_2 below:

$$T_2 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \tag{2.13}$$

which gives essentially the basis

$$\begin{bmatrix} \bar{a} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ \bar{a} \end{bmatrix}. \tag{2.14}$$

Now, corresponding to the ρ and α operators, $\bar{\rho}$ and $\bar{\alpha}$ operators are defined as the following:

Definition 2.24. Let R be a nonsingular matrix. The *Negative Reed-Muller Operator* $\bar{\rho}$ on R is:

$$\bar{\rho}(R) = \begin{bmatrix} 0 & R \\ R & R \end{bmatrix} \tag{2.15}$$

Definition 2.25. Let R be a nonsingular matrix. The *Negative AND-OR Operator* $\bar{\alpha}$ on R is:

$$\bar{\alpha}(R) = \begin{bmatrix} 0 & R \\ R & 1 \end{bmatrix} \tag{2.16}$$

Theorem 2.5. The $\bar{\rho}$ and $\bar{\alpha}$ operators result in nonsingular matrices of a higher dimension.

Proof: As in the proof of Theorem 2.4, the determinants of both $\bar{\rho}(R)$ and $\bar{\alpha}(R)$ are $det(R)^2$, thus nonzero. To this end, column exchanges will transform each of the above matrices into a block triangular matrix of the form (2.10). Moreover, since $-1 = +1$ in $GF(2)$, column exchanges do not alter the determinant. *QED*

Now, each variable can take either a positive or a negative operation and thus there exist 2^n possible Consistent Generalized forms for each positive $\alpha\rho$ family of bases. The changing of the order of application of the operators and the order of variables transcends into column swaps in the transition matrix, this is shown in the following example:

Example 2.2. In the following, a CG$\alpha\rho$ basis of three variables will be shown. Here, the order and the polarity of the variables is given as: $\bar{b}c\bar{a}$, where the "natural" order of variables is assumed to be abc. First, the transition matrix for the natural order is generated and then the corresponding transition matrix for the given order will be shown.

$$\rho\bar{\alpha}[T_2] = \begin{bmatrix} \bar{\alpha}[T_2] & 0 \\ \bar{\alpha}[T_2] & \bar{\alpha}[T_2] \end{bmatrix}; \textit{where } \bar{\alpha}[T_2] = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

The transition matrix above results in the following basis functions:

$$\begin{bmatrix} 0&0&0&1&0&0&0&0\\ 0&0&1&1&0&0&0&0\\ 0&1&1&1&0&0&0&0\\ 1&1&1&1&0&0&0&0\\ 0&0&0&1&0&0&0&1\\ 0&0&1&1&0&0&1&1\\ 0&1&1&1&0&1&1&1\\ 1&1&1&1&1&1&1&1 \end{bmatrix} \cdot \begin{bmatrix} abc\\ \bar{a}bc\\ a\bar{b}c\\ \bar{a}\bar{b}c\\ ab\bar{c}\\ \bar{a}b\bar{c}\\ a\bar{b}\bar{c}\\ \bar{a}\bar{b}\bar{c} \end{bmatrix} = \begin{bmatrix} \bar{a}\bar{b}c\\ \bar{b}c\\ (\bar{a}+\bar{b})c\\ \bar{c}\\ \bar{a}\bar{b}\\ \bar{b}\\ \bar{a}+\bar{b}\\ 1 \end{bmatrix}$$

The corresponding transition matrix for the "bca" ordering will be:

$$\begin{bmatrix} 0&0&0&0&0&0&1&0\\ 0&0&0&0&1&0&1&0\\ 0&0&1&0&1&0&1&0\\ 1&0&1&0&1&0&1&0\\ 0&0&0&0&0&0&1&1\\ 0&0&0&0&1&1&1&1\\ 0&0&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1 \end{bmatrix} \cdot \begin{bmatrix} abc\\ \bar{a}bc\\ a\bar{b}c\\ \bar{a}\bar{b}c\\ ab\bar{c}\\ \bar{a}b\bar{c}\\ a\bar{b}\bar{c}\\ \bar{a}\bar{b}\bar{c} \end{bmatrix} = \begin{bmatrix} \bar{b}\bar{c}a\\ \bar{c}a\\ (\bar{b}+\bar{c})a\\ a\\ b\bar{c}\\ \bar{c}\\ \bar{b}+\bar{c}\\ 1 \end{bmatrix}$$

It can be observed that the order of columns has changed to 1, 5, 2, 6, 3, 7, 4, and 8 in the two transition matrices. □

The change in the order of columns and the permutations in order of the variables have the same result. This can be justified by the fact that changing the order of variables is same as reassigning minterms - changing minterm rows. This change can be done by maintaining the original minterm orders but changing the corresponding columns in the transition matrix.

2.5.3. Generalized αρ Family of Bases

The members of the αρ family of bases need not be confined to fixed polarities in order to provide new bases. The polarity of the literals can be "inconsistently" varied and still result in a new basis. This will be shown by the following example:

Example 2.3. The basis generated in Example 2.2 can now be inconsistently changed for polarity of literals to give the following new basis:

$$\begin{bmatrix} b\bar{c}a \\ \bar{c}a \\ (\bar{b}+\bar{c})\bar{a} \\ a \\ b\bar{c} \\ \bar{c} \\ b+\bar{c} \\ 1 \end{bmatrix}. \square$$

As it can be observed, the literals a, b, and c take different polarities in different basis functions.

This larger family of bases will be termed *Generalized* αρ (Gαρ).

2.5.4. αρσ Family of Bases

A different generalization of the αρ family of bases is possible through the introduction of a third operator called the Shannon operator, σ. This family will then have 3^n different expansions.

First the starting transformation matrix for this extension will be introduced. Again similar to Equation (2.11), the basis for a single element is given as T_3:

$$T_3 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.17}$$

which gives essentially the basis

$$\begin{bmatrix} a \\ \bar{a} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ \bar{a} \end{bmatrix}. \tag{2.18}$$

The σ operator is then defined as the following:

Definition 2.26. Let R be a nonsingular matrix. The *Shannon Operator* σ on R is:

$$\sigma(R) = \begin{bmatrix} R & 0 \\ 0 & R \end{bmatrix} \tag{2.19}$$

Theorem 2.6. The σ operator results in a nonsingular matrix of a higher dimension.

Proof: As in the proof of Theorem 2.4, the determinant of $\sigma(R)$ is $det(R)^2$, thus nonzero. *QED*

Notice that T_1, T_2, and T_3 are three possible nonsingular matrices for a single variable function. All other nonsingular matrices for a single variable function can be constructed from swapping the rows of these three matrices and would not result in any new bases. As an example, Similar to α and ρ operators, it is possible to define a negative σ operator as shown below:

Definition 2.27. Let R be a nonsingular matrix. The *Negative Shannon Operator* $\overline{\sigma}$ on R is:

$$\overline{\sigma}(R) = \begin{bmatrix} 0 & R \\ R & 0 \end{bmatrix} \tag{2.20}$$

The basis for a single element here is:

$$T_4 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.21}$$

which gives essentially the basis

$$\begin{bmatrix} \overline{a} \\ a \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ \overline{a} \end{bmatrix}. \tag{2.22}$$

It can be observed that T_3 and T_4 define essentially the same basis and no new bases will be generated by negative Shannon operator when the corresponding positive is present.

Theorem 2.7. The $\overline{\sigma}$ operator results in a nonsingular matrix of a higher dimension.

The proof is exactly the same as in Theorem 2.5 and is omitted here.

2.5.5. π Operation on $\alpha\rho\sigma$ Family of Bases

The π operation, introduced by Green [58] and symbolized as $\circ$, can still further allow generalization of the previous family. This operation is defined below and is

shown to result in nonsingular matrices similar to α, ρ, and σ operations before.

Definition 2.28. Let R_1, and R_2 be nonsingular matrices of the same size, and let $T_i \in \{T_1, T_2, T_3\}$. The *pseudo-Kronecker Operator, π, on T_1, R_1, and R_2, symbolized by $\circ$ is:*

$$\pi(T_i, R_1, R_2) = T_i \circ \{R_1, R_2\} = \begin{bmatrix} \tau_{00} & \tau_{10} \\ \tau_{01} & \tau_{11} \end{bmatrix} \circ \{R_1, R_2\} = \begin{bmatrix} \tau_{00}R_2 & \tau_{10}R_2 \\ \tau_{01}R_1 & \tau_{11}R_1 \end{bmatrix} \quad (2.23)$$

where $\tau_{ij}R_k$ refers to a matrix of size R_k where each of its elements is multiplied by the scalar τ_{ij}.

Example 2.4. Let $R_1 = T_1, R_2 = T_3$; Then

$$T_2 \circ \{T_1, T_3\} = \begin{bmatrix} 0\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & 1\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ 1\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} & 1\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \square$$

Theorem 2.8. The π operator results in a nonsingular matrix of a higher dimension.

Proof. The proof of the theorem is similar to the the proofs given for Theorems 2.4 and 2.5. Each of the matrices T_i will result in one block matrix of 0. Then similar to the proof of the above theorems, it can be argued that the determinant of the larger matrix is some multiple of the determinants of the smaller nonsingular submatrices and therefore nonzero. *QED*

Introduction of the σ, and π operations still extends the possibilities of generating new AND/OR bases. Certain other generalizations have been known for the AND/XOR bases in the literature. Those generalizations can also be extended to the AND/OR bases resulting in even larger classes of AND/OR bases. As the AND/XOR bases have been known in the literature and all of them can be used in logic optimization stage for CMLA, the next section is devoted to the review of the AND/XOR bases.

2.6. AND/XOR Canonical Forms

AND/XOR canonical forms have been up to now the only XOR forms that have been studied. These forms are mostly known as Reed-Muller canonical forms. Although the first of these is first attributed to Zhegalkin [158], they are referred to as Reed-Muller forms following Reed [110] and Muller [89]. In the following, they will mostly be referred to as AND/XOR forms in order to put them more into perspective.

A comprehensive presentation of different AND/XOR canonical forms was given by Davio et al [38] and more recently extended by Green [58].

The concept of monoterms was defined in the previous section. Here, different kinds of monoterms will be given. These monoterms then can be used to define various AND/XOR canonical forms. The term "monoterm" has been based on the concept of monomials in algebra to convey both the monomial structure of the terms and also designate the specific characteristics of these monomials.

Definition 2.29. A *monomial* is an algebraic expression consisting of a single term which is a product of numbers and variables.

Definition 2.30. A *monoterm* is a monomial in which all the literals occur linearly only and the numbers are in *GF* (2).

Example 2.5. $x_1x_2\bar{x}_3$ is an example of a monoterm, while $x_1x_1x_2$ (or $x_1^2x_2$) is not. In the first case all the literals occur only once while in the second one x_1 occurs twice. □

In order to make distinctions among various possible monoterms, special monoterms giving rise to different AND/XOR Canonical forms are defined in the following.

Definition 2.31. A *positive monoterm* is a monoterm in which all the literals occur in positive polarity only.

Example 2.6. $x_1x_2x_3$ is an example of a positive monoterm. □

Similar to a positive monoterm, a negative monoterm can be defined.

Definition 2.32. A *negative monoterm* is a monoterm in which all the literals occur in negative polarity only.

Example 2.7. $\bar{x}_1\bar{x}_2\bar{x}_3$ is an example of a negative monoterm while $\bar{x}_1x_2\bar{x}_3$ is not because x_2 occurs in positive polarity. □

The above two monoterms can be seen as special cases of the more general concept of the monoterm. In a monoterm, in general, variables can occur in either positive or negative polarities.

Example 2.8. $\bar{x}_1x_2\bar{x}_3$ as well as $\bar{x}_1\bar{x}_2\bar{x}_3$ and $x_1x_2x_3$ are all examples of monoterms. The last two terms are special cases of negative and positive monoterm accordingly. □

Definition 2.33. *Reed-Muller Canonical form (RMC)* or positive polarity AND/XOR Canonical form is a sum of positive monoterms, the sum being over *GF* (2).

RMC results by $n-1$ successive applications of the Reed-Muller operator, ρ, on T_1 (Equation 2.11), for an n-variable Boolean function. Mathematically, the *RMC* can be shown as:

$$f(x_1, x_2, ..., x_n) = \bigoplus_{i=0}^{2^n-1} a_i \mu_i \tag{2.24}$$

where $a_i \in \{0, 1\}$ and $\mu_i = x_n^{e_n} x_{n-1}^{e_{n-1}} \cdots x_2^{e_2} x_1^{e_1} = \prod_{j=1}^{n} x_j^{e_j}$ where $e_j \in \{0, 1\}$ such that $e_n e_{n-1} \cdots e_2 e_1$ is a binary number which is equal to i. Moreover $x_i^0 = 1$ and $x_i^1 = x_i$. $\oplus$ denotes summation over *GF* (2), the Galois field of two elements.

If Equation (2.24) is expanded, one gets:

$$f(x_1, x_2, ..., x_n) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_1x_2 \oplus a_4x_3 \oplus \cdots \oplus a_{2^n-1}x_1x_2 \cdots x_n.$$

As an example, any function of three variables can be uniquely represented in

RMC form as:

$$a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_1x_2 \oplus a_4x_3 \oplus a_5x_1x_3 \oplus a_6x_2x_3 \oplus a_7x_1x_2x_3. \quad (2.25)$$

Depending on the function, different coefficients a_i will be either 0 or 1. Notice that all variables retain positive polarity throughout.

Example 2.9. $x_1 \oplus x_2 \oplus x_1x_3 \oplus x_1x_2x_3$ is one example of a function represented in this form. In this example the coefficients a_1, a_2, a_5, and a_7 are 1 and the rest are 0. □

As there are 2^{2^3} or 2^8 possible functions of three variables and there are also 2^8 possible combinations of a_i coefficients being 1, the uniqueness of the representation in this case can be seen. For reference, the positive monoterms for three variables are shown in Table 2.6.

No.	Monoterm	μ_n
0	1	μ_0
1	x_1	μ_1
2	x_2	μ_2
3	x_1x_2	μ_3
4	x_3	μ_4
5	x_1x_3	μ_5
6	x_1x_2	μ_6
7	$x_1x_2x_3$	μ_7

Table 2.6 Positive Monoterms of three variables

A negative polarity AND/XOR Canonical form can be similarly defined:

Definition 2.34. *Negative Reed-Muller Canonical form (NRMC)* is sum of negative monoterms, the sum being over *GF* (2).

This canonical form is the result of $n-1$ successive applications of negative Reed-Muller operator, $\bar{\rho}$, on T_2 and is given as:

$$f(x_1, x_2, ..., x_n) = \bigoplus_{i=0}^{2^n-1} a_i \bar{\mu}_i \quad (2.26)$$

where $a_i \in \{0, 1\}$ and $\bar{\mu}_i = \bar{x}_n^{e_n} \bar{x}_{n-1}^{e_n-1} \cdots \bar{x}_2^{e_2} \bar{x}_1^{e_1} = \prod_{j=1}^{n} \bar{x}_j^{e_j}$ where $e_j \in \{0, 1\}$

such that $e_n e_{n-1} \cdots e_2 e_1$ is a binary number which is equal to i; $\bar{x}_i^0 = 1$ and $\bar{x}_i^1 = \bar{x}_i$.

Example 2.10. The function in example 2.9 in *NRMC* form is represented as:

$$\bar{x}_1 \oplus \bar{x}_1\bar{x}_2 \oplus \bar{x}_2\bar{x}_3 \oplus \bar{x}_1\bar{x}_2\bar{x}_3. \ \square$$

In both *RMC* and *NRMC*, the variables retain the same polarity throughout. If the variables occur as either positive or negative, more general forms are possible. One family of canonical forms, terms of which are not necessarily positive or negative monoterms, is called the *Generalized Reed-Muller Canonical forms (GRM).* A subset of *GRM* forms are the consistent *GRM* forms.

Definition 2.35. A *Consistent Generalized Reed-Muller Canonical form (CGRM),* also known as a *fixed polarity AND/XOR canonical form,* is a sum of monoterms in which each variable keeps the same polarity in all the monoterms. Sum is again over *GF* (2).

The consistent forms are a subclass of CGαρ family of forms where only ρ operators, positive or negative, are used. It is obvious that *RMC* and *NRMC* are special cases of *CGRM* where the polarities for all variables are the same. As each variable can take either negative or positive polarity, the total number of *CGRM* forms is 2^n, where n is the number of the variable in the function. These forms can be mathematically represented as:

$$f(x_1, x_2, ..., x_n) = \bigoplus_{i=0}^{2^n-1} a_i \dot{\mu}_i \tag{2.27}$$

where $a_i \in \{0, 1\}$ and $\dot{\mu}_i = \dot{x}_n^{e_n} \dot{x}_{n-1}^{e_n - 1} \cdots \dot{x}_2^{e_2} \dot{x}_1^{e_1} = \prod_{j=1}^{n} \dot{x}_j^{e_j}$ where $e_j \in \{0, 1\}$ such that $e_n e_{n-1} \cdots e_2 e_1$ is a binary number which is equal to i; $\dot{x}_i^0 = 1$ and $\dot{x}_i^1 = \dot{x}_i$; $\dot{x}_i$ standing for x_i or $\bar{x}_i$ but not both. $\oplus$ again denotes summation over *GF* (2).

Expanding Equation (2.27) one gets:

$$f(x_1, x_2, ..., x_n) = a_0 \oplus a_1\dot{x}_1 \oplus a_2\dot{x}_2 \oplus a_3\dot{x}_1\dot{x}_2 \oplus a_4\dot{x}_3 \oplus \cdots \oplus a_{2^n-1}\dot{x}_1\dot{x}_2 \cdots \dot{x}_n.$$

Example 2.11. Choosing polarity 010, or decimal number 2, among the 8 possibilities for a three variable function, the variables x_1, and x_3 will have positive polarities and x_2 will have a negative polarity. The function in Example 2.9 will then have the following *CGRM* representation for polarity 010:

$$1 \oplus x_1 \oplus \bar{x}_2 \oplus x_1\bar{x}_2x_3. \square$$

Retaining the same polarity for a variable throughout is a condition that is not necessary for unique canonical representation of Boolean functions. By removing this restriction, more general canonical forms can be introduced. One such forms is that of the *GRM* forms.

Definition 2.36. A *Generalized Reed-Muller Canonical Form (GRM)* or *Generalized AND/XOR Canonical Form* is a sum of monoterms in which each variable can occur in different polarities in each monoterm; however, a monoterm with the same set of variables can occur once and only once. The sum again is over *GF* (2).

GRM forms are given by:

$$f(x_1, x_2, ..., x_n) = \bigoplus_{i=0}^{2^n-1} a_i \dot{\mu}_i \tag{2.28}$$

where $a_i \in \{0, 1\}$ and $\dot{\mu}_i = \dot{x}_n^{e_n}\dot{x}_{n-1}^{e_{n-1}} \cdots \dot{x}_2^{e_2}\dot{x}_1^{e_1} = \prod_{j=1}^{n} \dot{x}_j^{e_j}$ where $e_j \in \{0, 1\}$ such that $e_n e_{n-1} \cdots e_2 e_1$ is a binary number which is equal to i; $\dot{x}_i^0 = 1$ and $\dot{x}_i^1 = \dot{x}_i$. $\dot{x}_i$ stands for x_i or $\bar{x}_i$ but not both. $\oplus$ again denotes summation over *GF* (2).

The *GRM* forms are a subset of the (G$\alpha\rho$) family of bases where only ρ is used. One subset of the *GRM* forms are the *Inconsistent* forms which are differentiated from *CGRM* forms in that if the variables keep the same polarity throughout, they are not included in inconsistent forms any more. The total sum of consistent and inconsistent forms makes up the Generalized Reed-Muller Canonical forms. The number of *GRM* forms is $2^n 2^{n-1}$ where as mentioned previously, 2^n of these forms are consistent and the rest are inconsistent *GRM* forms. *GRM* forms have also been termed as *Restricted Mixed*

Polarity forms (CRMP) [34].

Example 2.12. The following monoterms provide a canonical representation of any three variable function:

$$[1 \;\; \bar{x}_1 \;\; x_2 \;\; \bar{x}_1 x_2 \;\; \bar{x}_3 \;\; x_1 x_3 \;\; \bar{x}_2 \bar{x}_3 \;\; x_1 x_2 x_3] \tag{2.29}$$

These monoterms certainly do not meet the conditions for a fixed polarity form. This form is an *Inconsistent* form and the function used in Example 2.9 will have the representation $1 \oplus \bar{x}_1 \oplus x_2 \oplus x_1 x_3 \oplus x_1 x_2 x_3$ in this form. □

A larger family of AND/XOR canonical forms which include the fixed-polarity forms as a subset are constructed by inclusion of the Shannon operator. It can be noted that the minterm canonical forms presented in section 2.4 can be seen as a special AND/XOR canonical form where only Shannon operator is applied on the starting matrix T_3 (Eq. 2.17). The transition matrix produced by successive applications of σ operator starting from T_3 will be nothing but the identity matrix, resulting in the minterms themselves. A more general class of AND/XOR forms is that of *Kronecker Reed-Muller Canonical forms (KRM)* [38, 57, 58]. The term Kroncker comes the fact that these forms can be produced by the Kronecker product of the the three bases given in Equations (2.12), (2.14), and (2.18).

Definition 2.37. A *Kronecker Reed-Muller (KRM)*, canonical form or *Kronecker AND/XOR Canonical form* is a sum of monoterms where some variables appear only in positive polarity; some variables appear only in negative polarity; and some variables appear in either positive or negative polarity but they are present in every single monoterm. The sum is over *GF* (2).

The *KRM* forms are constructed by application of ρ, $\bar{\rho}$, and σ operators starting from the starting matrices T_1, T_2, or T_3 (Eq. 2.11, 2.13, and 2.17). As each variable can be expanded according to any of the above three operations, the total number of *KRM* forms is 3^n, where n is the number of variables.

Clearly, not all *inconsistent* forms are part of the set of *KRM* forms since in *KRM*s, a monoterm with the same variables can occur more than once though with different literal polarities. In addition, there are some *inconsistent* forms which can be not be constructed by the application of the three mentioned operators.

One set of canonical forms which includes *KRM* and some *inconsistent* forms is the *Pseudo-Kronecker Reed-Muller (PKRM)* Canonical forms [37].

PKRM forms are based on the operation of π on nonsingular matrices produced by σ, ρ, and $\bar{\rho}$ operations on T_1, T_2, or T_3. In addition to the *KRM* forms, this results in other inconsistent generations of the monoterms as compared to *KRM*s. It can be observed that by σ, ρ, and $\bar{\rho}$ operations, the Boolean function $f(x_1, x_2, ..., x_n)$ can be represented with respect to x_{n-1} as:

$$f = a_{00}\bar{x}_{n-1} \oplus a_{10}x_{n-1} \tag{2.30}$$

$$f = a_{01} \oplus a_{11}x_{n-1} \tag{2.31}$$

$$f = a_{02} \oplus a_{12}\bar{x}_{n-1}. \tag{2.32}$$

If the coefficients a_{0i} and a_{1i}, $i = 0, 1, 2$ are inconsistently developed with respect to the variable x_{n-2}; i.e. each part of the function is again expanded by the three operations σ, ρ, and $\bar{\rho}$ independently, there would be 3×3^2 different kind of forms possible. If a Boolean function in turn is expanded in this way with respect to each of its variables, there would be $3^{(\sum_{k=0}^{n} 2^k)} = 3^{2^{n+1}-1}$ possible forms which comprise the *PKRM* canonical forms. Among these, 3^n of them are *KRM* forms.

Example 2.13. The following monoterms give a *PKRM* form:

$$[x_2\ \bar{x}_2] \circ \{[x_1\ 1], [x_1\ \bar{x}_1]\} = [x_2[x_1\ \bar{x}_1]\ ]\ [\bar{x}_2[x_1\ 1]\ ] = [x_1x_2\ \bar{x}_1x_2\ x_1\bar{x}_2\ \bar{x}_2] \tag{2.33}$$

where $\circ$ is the pseudo-Kronecker operator defined in Equation 2.23. □

PKRM's were introduced by Davio [37] and expanded later in [38]. Green [58] and Sasao [123] have provided a more detailed examination of these forms.

Even higher supersets for *PKRM* canonical forms were introduced by Green [58]. One immediate superset of these forms was termed the *Quasi-Kronecker Reed-Muller (QKRM)* Canonical forms. These forms are the result of the interchanging of variables in a *PKRM* form. While *PKRM* forms can be viewed as inconsistent expansion of branches by the three operations, ρ, $\bar{\rho}$, and σ maintaining an order of the variables, the *QKRM* forms can be viewed as one that the order of variables is also inconsistent.

Example 2.14. If the variables x_1 and x_2 are interchanged in Equation 2.33, one gets a different basis:

$$[x_1\ \bar{x}_1] \circ \{[x_2\ 1],\ [x_2\ \bar{x}_2]\} = [\,x_1[x_2\ \bar{x}_2]\,]\ \ [\,\bar{x}_1[x_2\ 1]\,] = [x_2x_1\ \ \bar{x}_2x_1\ \ x_2\bar{x}_1\ \ \bar{x}_1]\ \square$$

As there are $n!$ possible permutations for n variables, the total number of *QKRM* forms could be $n!\,3^{2^{n+1}-1}$. However, not all of these permutations will result in new forms and therefore, the total number of *QKRM* forms is actually less than this number.

Still another superset can be identified for *QKRM* forms, termed *skew* forms [58]. In these forms, the variables can be inconsistently interchanged to result in even different canonical forms.

Example 2.15. Interchanging variables $\bar{x}_1$ and x_2 in (2.33) will result in the following *skew* basis:

$$[x_1x_2\ \ x_2\ \ x_1\bar{x}_2\ \ \bar{x}_2]\ \square$$

There are still certain supersets of *skew* forms which do not show any general structure of one-variable modules. Green has termed these forms *residual* forms and he points out that due to their "lack of general structure", it is difficult to enumerate them. Many of the above forms are the result of collapsing certain multi-level DAG structures into a Two-level form. These DAG structures will be described in chapter 6.

Since many of the above mentioned forms are defined by operations on single variables, one can conjecture that still more forms can be defined by modules of more than one variable. This is an open problem and has not been tackled yet in the literature.

The most general representation, in which there are no restrictions on the ring summation of monoterms has been termed the *Exclusive Sum of Products (ESOP)* [120]. *ESOP* is also used specifically to denote AND/XOR canonical forms which are not in any of the forms defined above. This is the most often use of the term in the literature. Still other AND/XOR canonical forms have been introduced which are not described here [100, 123].

Definition 2.38. *Exclusive Sum of Products (ESOP)* is a ring sum of monoterms.

Figure 2.1, which is a combination of Figure 4.7 in [38] and Figure 9 in [58], shows different AND/XOR canonical forms and the relations amongst them.

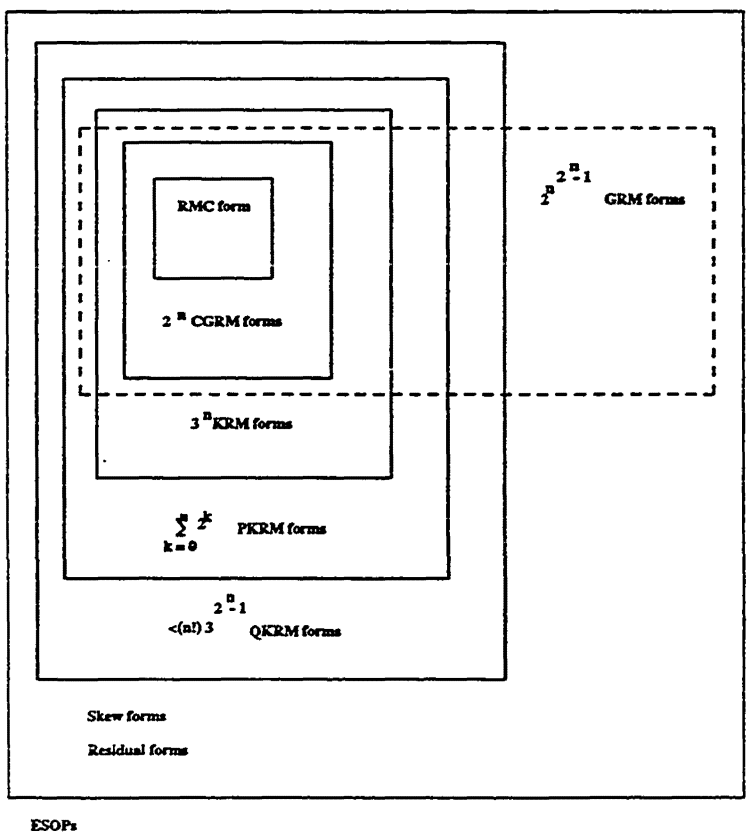

Figure 2.1 The AND/XOR Canonical Forms

A note on the total number of possible AND/XOR canonical forms can be made

here. For two variable binary Boolean functions, there are 81 different AND/XOR canonical forms possible. This number for three variable binary Boolean functions has been reported as 28 431 [58]. As the number of variables increases, there are much more forms possible to define.

Introduction of the first canonical form in Boolean rings with unit goes back to Zhegalkin [158]. Reed [110] and Muller [89] reached at the same results and different canonical forms have been known after them. Their work was concentrated on *RMC* or the positive polarity AND/XOR canonical form. Introduction of consistent *GRM* is due to Akers [1]. Inconsistent forms were introduced by Cohn [30]. The *KRM* and *PKRM* are the contributions of Davio [37, 38]. *QKRM*, Skew and residual forms were introduced by Green [58].

2.7. Summary

In this chapter it was reiterated that the Boolean algebra is not the only possible representation for Boolean functions. Moreover, as logics other than Boolean algebra can be realized in FPGAs at no extra cost, it is even more imperative to not be confined only to Boolean algebra for synthesis purposes. Furthermore, Universal XOR Forms were introduced here for the first time as a large family of canonical representations of the functions based on XOR logic. It was shown that the UXF comprise a much larger family of forms than the known AND/XOR canonical forms.

Specifically this author introduced various AND/OR/XOR canonical forms which can provide more compact representation of the functions than either SOP or ESOP representations of the functions. Still other generalizations to the AND/OR/XOR canonical forms are possible by applying the known generalizations in AND/XOR forms directly to these UXF.

It is still an open problem as how to identify a minimal multi-level XOR canonical representation of a function without the need for extensive search in the large space of

the XOR canonical forms. One useful approach proposed by this author can be through the investigation of the general linear group and the identification of possible partitions of these forms.

A major advantage of the various AND/OR/XOR UXF for CA-Type FPGA synthesis is that they can be directly mapped to the FPGA. As these forms are generated by one operation for each variable at a time, the variables do not need any prior reordering.

Chapter 3

Minimal Realization of Boolean Functions in Fixed Polarity AND/XOR Forms

3.1. Introduction

It was shown in chapter 2 that a given Boolean function can have numerous realizations. Among these realizations, the one with a minimal number of operations will be the most economical. In terms of the cellular synthesis, fewer number of operations translates into fewer number of cells. For the case of minimal realization of Boolean functions, the task of identification and realization of the minimal realization is directly influenced by the data structure used to represent the function. In this chapter a fast method for minimal realization of Boolean functions in fixed Polarity AND/XOR forms based on Cube Comparison Method (CCM) is presented. It is shown that the identification of the minimal polarity as well as the fast realization in this form can be efficiently performed utilizing this technique.

Two-level fixed polarity AND/XOR forms of the Boolean functions are among the most fundamental approaches to CA-type FPGA synthesis. These forms not only are of importance as an approach by themselves, but many other synthesis problems can be transformed to those utilizing this realization. Examples for these transformations can be found in such areas as classification of functions, Ashenhurst and other decomposition methods [104, 149], and multi-level design. Minimization of other realizations can also be based on the fixed polarity forms. It has been shown [12] that minimization of ESOPs

---

This chapter is based on the technique published in abridged form in A. Sarabi, M. A. Perkowski, proceedings of the 29th ACM/IEEE Design Automation Conference, Anaheim, CA, June 1992.

can be based on these forms. *GRM* minimization can also utilize the fixed polarity forms as shown in the next chapter. Of paramount importance is the easily testability properties of these forms which makes them the most easily testable realizations for Boolean functions. This aspect which can play a key role in design is discussed further in chapter 7.

Following Fisher [46], the problem of minimization of a Boolean function in AND/XOR fixed polarity form can be divided into two steps. The first step is to identify the optimal polarity for the function and the second is to realize the function in that polarity form.

The first step can further be approached in two ways, direct and indirect. In the direct approach, certain characteristics of the optimized polarity are used to identify this polarity for the given function. In the indirect approach, a search is involved, where different polarities are examined to see whether they meet the criteria of minimization or not. This search can be either exhaustive, searching all possible polarities, or heuristic. In heuristic search, certain characteristics are used to guide the search towards the minimum solution. Depending on the heuristics, the result can be the optimum or in majority of cases, quasi-optimum. For Boolean functions with small number of variables exhaustive search methods can be possible, but as the number of variables increases, this task will become impossible.

The second step of the minimization problem is very much dependent on the data structure used to represent the function. Among the factors that need to be considered are the ease of manipulation and representation. For large number of variables, any minimization scheme has to be amenable to computer use and manipulation. Hence the representation mode has to satisfy certain criteria as well.

The method utilized for the representation of the functions here is that of the *Cube Comparison Method (CCM)* [45, 128, 130]. The cube representation has the advantage of less memory requirements than truth vector representation and is easy for computer

representation and manipulation. It is also very much compatible with existing logic synthesis tools such as ESPRESSO where many cube operations are defined and employed.

As the problem of identifying the minimal polarity is NP-complete, direct methods do not apply in general but only in special cases. Due to the nature of the problem, search methods are the only viable solution when one is dealing with functions of a very large number of variables.

The basic approach introduced for the identification of the minimal polarity is based on the identification of certain characteristics represented in terms of disjoint cubes. In this way, for some functions it is shown that the polarity of certain literals in the minimized polarity vector can be determined without necessarily performing any search. Hence, it is possible to cut the number of necessary searches for identification of the minimized polarity vector. For other functions, exhaustive search is performed for the literals with unknown minimal polarity to come up with the minimal form(s). In the case of functions which possess many unknowns in their minimized polarity vector, a heuristic search is performed to come up with a quasi-minimal solution.

This chapter is comprised of six sections. In the next section, the problem of minimization as well as certain approaches to the problem are presented. Next, the realization of the fixed polarity form of any given polarity with improved CCM techniques is introduced. In section four, the identification of the minimal polarity vector is discussed. In particular, the monoterms in each cube and their commonality for each polarity and the minimal polarity as a function of these two is discussed. In the fifth section, the basic algorithms for identification of the minimal and quasi-minimal polarity vectors are presented and evaluations on MCNC (Microelectronic Center of North Carolina) benchmarks are discussed.

3.2. Approaches to Minimization

The data structures used to represent the Boolean functions directly influence the minimization approaches. The early approaches in 60's and 70's relied on formulations based on coding theory, graph theory, and later on Taylor series expansion. The complexities of these methods grow quickly with increasing number of variables and most of them loose their merits for more than 5 or at best 6 variables in the function. During the 80's the vector space formulations found more attention and several fast methods were devised. Spectral transforms and the new Cube Comparison Method were mostly applied in 1990 for the generation of fixed polarity forms. One method by Fisher [46], using a different terminology but very similar to the Cube Comparison Method, developed a very efficient method of generating these forms for Boolean functions with a large number of variables. Several 2-dimensional map oriented approaches were also devised in the 80's which are obviously mostly useful for functions with small number of variables.

As it was shown in the previous chapter, depending on the polarity of the literals, the same Boolean function can have representations with different number of monoterms. The goal of the minimization algorithm is to seek the minimal fixed polarity form, or the polarity of the literals which results in the smallest number of monoterms.

Some definitions make the formal presentation of the approaches possible.

Definition 3.1. Let f be a Boolean function. The vector $\dot{\mathbf{X}} = (\dot{x}_1, \dot{x}_2, ..., \dot{x}_n)$ is called the *polarity vector* of f, where $\dot{x}_i$ is either x_i or $\overline{x}_i$.

This vector can be represented as an ordered set of 1's and 0's corresponding to the polarity of the literal. If a literal $\dot{x}_i$ takes positive polarity, i.e. $\dot{x}_i = x_i$, a 1 will be placed in the i th position in the vector. If it takes a negative polarity, or $\dot{x}_i = \overline{x}_i$, then a 0 will be placed in that location. As an example, (1, 0, 0) refers to $x_1, \dot{x}_2, \dot{x}_3$.

Definition 3.2. For all possible fixed polarity forms of a Boolean function, the *minimal polarity vector* is the polarity vector for which the fixed polarity form will have

the least number of monoterms.

Example 3.1. The function $\bar{x}_1\bar{x}_2\bar{x}_3$ in positive polarity will be represented as:

$$\bar{x}_1\bar{x}_2\bar{x}_3 = 1 \oplus x_1 \oplus x_2 \oplus x_1x_2 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3 \oplus x_1x_2x_3.$$

The same function will just be represented as $\bar{x}_1\bar{x}_2\bar{x}_3$ if negative polarities are chosen for all the three variables. This *CGRM* obviously has less number of terms. □

As mentioned previously, there can be one or more minimal polarities for a given Boolean function. The one(s) which best fits all minimization criteria can be distinguished from all minimal polarities. The other criteria could be minimal number of literals, etc.

Definition 3.3. For all possible *CGRM* forms of a Boolean function, the *optimal polarity vector* is the polarity vector which meets all the specified minimization criteria.

As the *RMC*, the positive polarity AND/XOR form, was introduced in the context of coding theory and its applications in Boolean systems, some of the earlier attempts at minimizing *CGRM* forms approached the problem with coding-theoretic schemes. As reported in [87], Kautz formulated the problem as an error correction of a certain code. The problem with this approach was that with the increasing number of variables, the formulation became combinatorially more complex to solve.

It is reported in [80] that Pospelov has reduced the problem to linear integer 0 - 1 programming. Some other approaches reported in the same reference are by Hausenblas, Wallach, Ceitlin, Tosic, and Pospelov. All these methods, however, can handle Boolean functions with small number of variables.

Mukhopadhyay and Schmitz [87] gave a graph-theoretic approach. In this approach, the problem is defined in terms of determination of the maximum clique of a linear non-directed graph. The approach requires the exhaustive computation of all the 2^n possible forms in terms of "polarity functions" and obviously can not again handle functions with large number of variables.

Following the first introduction of *CGRM* forms, being in Taylor series expansion [1], several schemes of minimization in Taylor series form were introduced in 70's [17], [38]. Marinkovic and Tosic [80] have proposed a non-exhaustive search for identifying the minimal *CGRM* form which allows different minimization criteria to be used. Kodandapani and Setlur [75] modified this approach for the case of minimization in terms of the number of the monoterms. Page [94] used a different modification to the method for minimizing the number of literals appearing in an even number of monoterms which proves to have improved testability properties. The essential method here is that of a heuristic tree search. Its variants can result both in a minimal or in a quasi-minimal solution.

Vector space approaches to the fixed polarity minimization problem essentially rely on linear algebraic techniques to generate all 2^n possible *CGRM* forms. From a first erroneous attempt by Swamy [147] up to late 80's, several fast algorithms have been devised to speed up this exhaustive method and no serious attempt has been made at identifying the characteristics of the optimized polarity in this representation. In these approaches basically the structure of the transition matrices is exploited to generate one *CGRM* form from another in a *fast* scheme in different orderings. As reported by Green [57] the complexity varies as given below:

Absolute fast	Lexicographical fast	Gray code	Ternary map
$n\ 4^{n-1}$	$4^n - 2^{n-1}(n+2)$	$2^{n-1}(2^n - 1)$	$3^n - 2^n$

More recent approaches have utilized more efficient representations of Decision Diagrams for the minimization problem. The special relation of *CGRM*s and Binary Decision Diagrams (BDD) were shown in [107]. Minimizations similar to the ones presented in the present approach were used in [148] using BDDs rather than cubes and Functional Decision Diagrams (FDD) were utilized in [39].

3.3. Improved Techniques for a Given CGRM Realization of a Boolean Function

As mentioned in the introduction, realization of *CGRM* form of a given polarity is one of the two steps involved in any minimization scheme. The more efficient this realization is executed, the faster and more efficient will be the overall minimization scheme; especially when a search method involves several *CGRM* realizations. In this section, the Cube Comparison Method and certain improvements to the existing CCM method for *CGRM* realization will be described.

3.3.1. Cube Comparison Method and Spectral Methods

Spectral methods and the *Cube Comparison Method* provide fast methods for generation of *CGRM* coefficients and overcome the deficiencies of the transform methods discussed in the previous section. This method provides a generalized approach to all transforms of Boolean functions and the fixed polarity forms are one case in point.

The *spectrum* of a Boolean function is essentially the representation of the truth vector of the function in terms of some basis functions. Similar to Fourier transform of functions, the set of coefficients of the transform are called the *spectrum* of the function and each coefficient will be referred to as a *spectral coefficient.* The methods which use the transform of the truth vector to represent and study the properties of Boolean functions are referred to as *spectral techniques.* In these methods, as pointed out in the previous chapter, there can be various orthogonal transforms of the truth vector presented. As Reed-Muller bases are orthogonal, they were shown in previous chapter to define various orthogonal sets of functions. Each *CGRM* form, being comprised of a *CGRM* basis, defines a set of orthogonal functions and the coefficients of the expansion are the spectral coefficients of this *CGRM transform.* In this terminology then the coefficients of *CGRM* expansion and the spectral coefficients of the *CGRM* transform refer to the same objects.

There are certain advantages in representing the truth vector of the Boolean function in spectral domain. As described by Hurst *et al.* [66], *"Each of the* 2^n *... spectral*

coefficients contains some information about the behavior of the function at all 2^n points, but does not contain complete information about any of them. The combination of all the values in the spectrum leads to complete information about the whole function. In this sense the spectral coefficients are giving us global information about the function, while the Boolean domain consists of local information. For some applications this global information is more directly useful than the Boolean representation of the functions." These applications are mainly those of classification and decomposition of functions as well as in logic synthesis.

The correspondence between vector space representation and orthogonal transforms, which in the case of Reed-Muller canonical forms are the same thing, makes the fast transform procedures described in previous section applicable here as well. The fast methods of finding the coefficients of the *CGRM* expansions result in fast generation of spectral coefficients. The goal of minimization in this context is to identify the spectrum which has the least number of 1's.

The Cube Comparison Method is devised for transformation of Boolean functions and its merits stem from the fact that it operates on cube representation of the functions rather than the truth vector. All the fast methods discussed previously start from the truth vector and perform the operations given in the transformation matrix exploiting certain properties of the matrix. Operating on the cube representation of the function reduces the number of data points that need to be operated on and is much more efficient. The first to use this alternative scheme were Muzio and Hurst. In their work they start from the cube representation of the function and present a procedure for generating the coefficients of transform in case of Walsh functions. Their method, however, starts from nondisjoint cube representation of the Boolean function. The Cube Comparison Method starts from disjoint cube representation. In this scheme, additional calculations needed for generation of the coefficients are avoided since there are no overlaps in the cubes and

coefficients are not re-calculated more than it is necessary. This makes the generation of spectral coefficients even faster and more efficient.

As the CCM is a general method, it is defined in the general case and then the special case of *CGRM* transforms is given in this method. Like any representation, some of the concepts previously are represented and defined differently here. In order to familiarize the reader with basic traits of this method, the terminology of the method is presented in the following. First the spectral coefficients are defined:

Definition3.4. Each *spectral coefficient* of a transform is a value representing a correlation between the Boolean function and a set of *basis functions* corresponding to this coefficient.

Basis functions of a given transform in the original derivation were defined as a minimal set of orthogonal functions specific to that transform. In this definition, the basis functions of Walsh transform are the Walsh functions and in the case of Reed-Muller transforms the basis functions are the monoterms of each relevant AND/XOR basis. However, as these functions can be defined for non-orthogonal transforms as well, they are more loosely defined. In general, basis functions in a cube comparison method are set of functions to which the coefficients of the transform are correlated to.

The *correlation* between the coefficients of the transform and the basis functions in general is found in two steps. The operations performed for each transform are different and this is what results in derivation of different transforms. These general steps are:

1. Perform a *matching operation* between the disjoint cubes of the Boolean function and the basis functions of the transform;
2. Perform *transform operation* on the results of step 1.

This is the most general way that any transform can be defined. The procedure to derive any transform in the Cube Comparison Method is to:

i) Represent the Boolean function in terms of disjoint cubes.

ii) Define the basis functions of the transform.

iii) Perform the matching operation between each cube and the basis functions to get partial set of coefficients of the transform.

iv) Perform the corresponding operation of the transform on the partial set above to get all the coefficients of the transform.

Now, each of these steps will be discussed in more detail. As the concept of disjoint cubes is a major advantage of this method, this concept will be described in the following. The basis functions, the matching operation and transform operation will next be described in terms of the *RMC* transform. Following that, the Equivalence operation and the method of generating any *CGRM* form from the disjoint cubes of the Boolean function will be described.

Step *i)* in CCM deals with the representation of Boolean functions in terms of disjoint cubes. In order to define disjoint cubes, the intersection of two cubes needs to be defined:

Definition 3.5. Let C_1 and C_2 be two cubes. The *cube intersection* is

$$C_1 \cap C_2 = \begin{cases} \varnothing & \text{if } any\ C_{1i} \cap C_{2i} = \varnothing \\ C_3 & otherwise, where\ C_{3i} = C_{1i} \cap C_{2i} \end{cases} \tag{3.1}$$

where C_{ki} represents the ith literal of the cube C_k and literal intersection is defined in Table 3.1.

$\cap$	0	1	-
0	0	$\varnothing$	0
1	$\varnothing$	1	1
-	0	1	-

Table 3.1. Cube Literal Intersection

Definition 3.6. Let C_1 and C_2 be two cubes. C_1 and C_2 are *disjoint* if $C_1 \cap C_2 = \varnothing$.

Example 3.2. The following examples show the intersection of cubes:

$$000 \cap 00\text{–} = 000$$

$$000 \cap 001 = 00\,\varnothing = \varnothing$$

$$0\text{–}1 \cap \text{–}11 = 011 \;\square$$

From the definition of intersection, it can be observed that for any two cubes to be disjoint, they should at least have one corresponding literal of opposite polarities. The disjoint concept can be extended to more than two cubes.

Definition 3.7. Let $\mathbf{C} = \{C_1, C_2, ..., C_n\}$ be a set of cubes. Then **C** is a set of disjoint cubes if $C_i \cap C_j = \varnothing \; \forall \; 1 \le i,j \le n, i \ne j$.

The disjoint cubes are generated by the technique in [44].

Step *ii)* in the Cube Comparison Method is definition of basis functions of the given transform. For any *CGRM* transform, the basis functions are the monoterms of that particular *CGRM* basis. For operational purposes, it is needed in this method to represent the basis functions in cube form as well.

Definition 3.8. A *basis function cube* is the cube representation of a basis function.

Example 3.3. The basis function cubes of *RMC* transform as well as the corresponding spectral coefficients are shown in Table 3.2.

No.	Basis Functions	Basis Function Cubes	Corresponding Spectral Coefficient
0	1	---	S_0
1	x_1	1--	S_1
2	x_2	-1-	S_2
3	x_1x_2	11-	S_{12}
4	x_3	--1	S_3
5	x_1x_3	1-1	S_{13}
6	x_2x_3	-11	S_{23}
7	$x_1x_2x_3$	111	S_{123}

Table 3.2 Basis Functions and Basis Function Cubes of *RMC* Transform

Step *iii)* of the CCM is performing the matching operation between each cube and the basis functions to get partial set of coefficients of the transform. In the case of *CGRM* transforms, the operation which designates which basis functions match, that is their corresponding coefficient is 1, was given by Fisher [46]. Although the terminology used in [46] is different from the terminology of the CCM, he exactly showed the method of identifying the partial set of spectral coefficients from each cube. Fisher's method is given in Theorem 3.1 without proof. The reader is referred to the paper for the proof of the theorem.

Theorem 3.1. (Fisher) Let $\mathbf{C} = (C_1, C_2, ..., C_n)$ be a cube ($C_i \in \{0, 1, -\}$) and let $a_j{}^k$ be the j'th coefficient at polarity k. Then $a_j{}^k = 1$ if and only if $(j_1, j_2, ..., j_n)$ is covered by the cube $\vec{\mathbf{C}}(\mathbf{C}, k) = (\vec{C}_1, \vec{C}_2, ..., \vec{C}_n)$ where

$$\vec{C}_i = \begin{cases} 0 & \text{if } \vec{C}_i = - \\ 1 & \text{if } \vec{C}_i = 0 \text{ or } 1 \text{ and } \vec{C}_i \neq k_i \\ - & \text{if } \vec{C}_i = 0 \text{ or } 1 \text{ and } \vec{C}_i = k_i \end{cases} \tag{3.2}$$

A cube $\mathbf{C}$ covers vertex v if $v_i = C_i$ whenever $C_i \neq -$.

By Fisher's theorem, the monoterms originating from a cube for the *RMC* expansion are all the cubes that have their 1's in th same literal positions as the 1's of the original cube and either "1" or "–" in the 0-literal positions of the original cube. For a *CGRM* of a different polarity, Example 3.4 shows the results of the above theorem.

Example 3.4. For *CGRM* of polarity vector 1; i.e. *RMC*, the cube 00--11 will have the following spectral coefficients as 1 and the rest will be zero:

S_{56}: ----11
S_{156}: 1---11
S_{256}: -1--11
S_{1256}: 11--11 □

The matching operation described by the theorem, and shown in Example 3.4, is essentially a pattern matching operation. This operation is performed between each cube

of the function and the basis function cubes in order to give the partial set of spectral coefficients for each function cube. It can be observed from Example 3.4 that the basis function cubes *have the exact 1's and –'s as in the function cube and it is only the 0's which differ.* As a matter of fact, all the basis function cubes are generated by exhausting all combinations of replacing every instance of a 0 in the function cube with either a 1 or a –. This is also essentially what the theorem is referring to.

Example 3.5. All the possible subsets of $\{a, b\}$ are $\{\ \}$, $\{a\}$, $\{b\}$, and $\{ab\}$. The 0's in the function cube act similar to the variables a and b above. In Example 3.4 for 00--11, ----11 corresponds to $\{\ \}$, 1---11 to a, -1--11 to b, and 11--11 corresponds to ab. □

As the number of all possible combinations of elements is 2^n, where n is the number of the elements, the number of spectral coefficients is also determined by the number of zeros in the cube. To be more exact, this number is equivalent to 2^{N_0}, where N_0 is the number of 0's in the cube.

In an existing program [129], the matching operation is used to check whether a spectral coefficient matches with the cubes of the function. The function cubes are first changed to their positive polarity equivalents through the concept of the *Equivalence operation.* Next, the spectral coefficients are matched with the equivalent function cubes and later transformed back to the original *CGRM* polarity. The bit-wise Equivalence operation is shown in Table 3.3.

		C_2		
	≡	0	1	–
	0	1	0	–
C_1	1	0	1	–
	–	–	–	–

Table 3.3. Bit-Wise Equivalence Operation Between Two Cubes

The program, however, checks all spectral coefficients and performs the matching

operation with all cubes of the function. This is unnecessary for many cases and with 2^n spectral coefficients for any function of n variables, the program becomes very slow. Based on the argument that only 2^{N_0} of the coefficients are 1, only these coefficients need to be generated. An improvement to the program in [129] has been made in this regard which will be discussed in section 4.2.

Step *iv)* deals with performing the corresponding operation of the transform to the partial set in step *iii)* in order to get all the coefficients of the transform. This operation is defined differently for each transform, hence, it is termed the *operation of the transform.* This operation in *CGRM* transforms is the *ring sum.* By this operation, the partial spectral coefficients which occur in an even number of cubes are removed and only those appearing in an odd number of them will be considered as the spectral coefficients of the transform. The operation of Walsh transform is the integer addition.

In order to differentiate the partial spectral coefficients generated in step *iii)* and the final spectrum generated in step *iv)*, the following terminology will be used. For *CGRM* transform, the partial coefficients will be called the monoterms *representing* the function (or cubes). The monoterms that occur in an odd number of the disjoint cubes will be referred to as the *expansion* monoterms.

Example 3.6 The monoterms representing the cube 001-10 at polarity 111000 are given in the following:

1) Perform the Equivalence Operation ≡ on the cube:

$$\begin{array}{rl} \equiv & 111000 \\ & 001\text{-}10 \\ \hline & 001\text{-}01 \end{array}$$

2) Generate the monoterms representing the cube:

No.	Spectral coeff.	cube representation
0.	S_{36}	--1--1
1.	S_{136}	1-1--1
2.	S_{236}	-11--1
3.	S_{1236}	111--1
4.	S_{356}	--1-11
5.	S_{1356}	1-1-11
6.	S_{2356}	-11-11
7.	S_{12356}	111-11

3) Perform the Equivalence Operation ≡ on each monoterm:

No.	Spectral coeff.	cube representation
0.	S_{36}	--1--0
1.	S_{136}	1-1--0
2.	S_{236}	-11--0
3.	S_{1236}	111--0
4.	S_{356}	--1-00
5.	S_{1356}	1-1-00
6.	S_{2356}	-11-00
7.	S_{12356}	111-00

□

The Equivalence operation can also be seen as a bitwise *mod*-2 sum operation, or XOR, if the literals in the cube are first inverted. This is shown in the following:

$$001\text{-}10 \rightarrow 110\text{-}01$$

$$\begin{array}{rl} \oplus & 111000 \\ & 110\text{-}01 \\ \hline & 001\text{-}01 \end{array}$$

which is the same cube as in the first step of Example 3.6.

As indicated previously, the monoterms representing any cube are the DC- and 1-literals of the cube with all the combinations of the 0-literals replaced with a 1 or –. The number of these monoterms were shown to be 2^{N_0} where N_0 represents the number of 0-literals in the cube. As for any polarity vector the 0-literals are exactly those literals which do not match with their corresponding literal in the polarity vector, the number of resulting monoterms for any cube in a given polarity vector is equal to 2^{N_v} where N_v is

the number of non-matching literals in the cube. The following theorem has then been proven:

Theorem 3.2. The number of monoterms representing a given cube for a given polarity vector is 2^{N_v}, where N_v is the number of non-matching literals of the cube and the polarity vector.

Example 3.7. In Example 3.6, there were 3 non-matching literals in the cube 001-10 for the polarity vector 111000, namely 1, 2, and 5, two matching literals; i.e. 3, and 6; and one DC-literal; i.e. 4. There are 8 resulting monoterms, which is equivalent to 2^3, and 3 is the number of non-matching literals. □

Using the set of disjoint cubes was mentioned to increase the efficiency of this method. If the cubes are not disjoint, the coefficients which represent the intersection of the cubes will be unnecessarily generated. The method proposed by Fisher [46] also uses disjoint cubes and as such it can be called a Cube Comparison Method in the case of *CGRM* transform long time before CCM was introduced.

In summary, the Cube Comparison Method is based on spectral techniques and fast generation of the spectrum of Boolean functions directly from disjoint cube representation of the function. The disjoint cube representation provides improved overall efficiency over the classical transform of the truth vector of the function. The cubes are "matched" against the basis functions to generate partial spectral coefficients. These partial coefficients are then operated on by the operation of the transform to result in the final spectrum of the transform.

3.3.2. Generation of the Monoterms Representing Each of the Disjoint Cubes

In this section, two methods for generation of the monoterms representing the disjoint cubes of the function will be introduced. One method uses a barrel shifter analogy to generate the monoterms from combinations of 0's, and the other a Gray code order. It

is shown that the second approach is more efficient.

In the methods introduced, only the monoterms which represent each cube are generated rather than the whole spectrum. Thus, although some monoterms are generated several times, as they might be common in several cubes, for a very wide range of cases the number of total monoterms generated would be much smaller than the whole spectrum of the function. The approach, however, will be less efficient in cases where there are many non-matching literals in the cubes for a given polarity vector. In those cases, and especially when all the literals of a given cube are all non-matching, this method not only requires the generation of the whole spectrum, but also requires even more operations. This is, however, an extreme case and as mentioned before, for a very wide range of cases, there will be far less number of monoterms required to be generated with this method as compared with the method in [129]. In any case, the number of operations required here is a function of the number of 0's in the cubes. The actual number is given by

$$\sum_{i=1}^{m} 2^{N_{v_i}}$$

where m is the number of cubes and N_{v_i} is the number of the non-matching literals in cube i. The other extreme, will be the case where there are only a handful of monoterms that need to be generated for a function with very large number of variables. In this case, this method will be very fast while the previously mentioned method will take a very long time to generate the whole spectrum. The overall advantage of this improvement is shown later as the programs are compared for different benchmark functions.

For the purpose of generation of the monoterms representing each disjoint cube, two approaches were taken where one proved to be faster. These methods are based on bitwise operations on the literals of the cubes to generate the corresponding monoterms. The first method uses a "barrel shifter" scheme to generate all the possible 2^{N_0} combinations that give the monoterms representing that cube. Here no special order is required as

the order does not alter the results. The second method is based on Gray code order of generation of the 2^{N_0} combinations which proves to be simpler and faster. This is due to the fact that for each new monoterm only one bit needs to be changed at a time. When dealing with large number of cubes the speed improvement contributes a lot to the speed of the overall method.

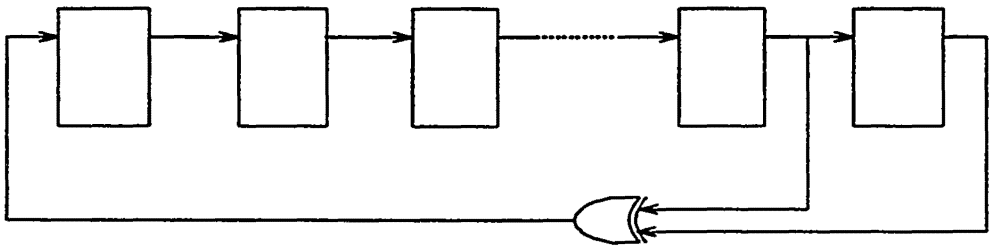

Figure 3.1. Example of a "BILBO" barrel shifter

The barrel shifter is essentially a sequence generator that generates 2^n different combinations of n literals taking values of either 1 or 0. In hardware, this sequence generator is comprised of n flip-flops and some combinational logic to determine what kind of sequence is generated. The outputs of the flip-flops designate the number generated. This shifter starts with a value of 1 at the first flip-flop and continues from that while 00...0 is never produced. Figure 3.1 shows one such generator where the order is not necessarily of importance.

Note that the XOR gate shown should have its inputs as the outputs of the last and the next to the last flip-flops to the right for the even number of flip-flops and the last and two before the last for the odd number of flip-flops. Otherwise, all the 2^n different codes would not be generated.

In software, the 1-literals are set and the position of each of the 0-literals is stored for each cube. The location of each of the 0-literals is taken as acting as the flip-flops and the bitwise XOR operation on the values of the appropriate bits works as the combi-

national logic part of the barrel shifter. Similarly, the same sequence generator is implemented in software. The values of the 0-literals are changed to 0 or 1 as required and each time one of the possible spectral coefficients (or the monoterms) is generated. The procedure here requires n - 1 shift operations and one XOR operation each time and depending on the sequence some conditions such as generation of 00...0 has to be taken into consideration.

The second approach uses a Gray code order for generating the spectral coefficients. In this approach, each time only one bit will change its value and as such is very fast. Similar to the first approach, the 1-literals are set and the location of each of the 0-literals is stored for each cube. However, rather than using the BILBO analogy, the appropriate bit is negated when required in a Gray code order.

3.3.3. Implementation of the Ring Sum Operation

The forth step of CCM, in the case of *CGRM* s, is the operation of the ring sum on the partial monoterms generated from the methods in section 4.2. In this section, four different schemes are introduced for this purpose. Among these, three of them have been implemented and they will be evaluated over several functions.

In the program REED [129], the ring sum operation is performs each time a spectral coefficient is examined. That is every single cube is matched against the spectral coefficient and it is discarded unless the coefficient occurs in an odd number of the cubes. Since in the methods introduced in section 4.2, the monoterms are generated in a different way, this operation can not be applied the way it is performed in [129]. The methods examined are described in the following:

In the first method, each newly generated monoterm is checked against all the other disjoint cubes which can have this monoterm as one of their representing monoterms. If this monoterm occurs in an even number of cubes, it is discarded, otherwise it is kept as an expansion monoterm of the Boolean function.

In the second method, first all the monoterms representing each of the cubes are generated and are stored in an array. This array is then sorted and the monoterms occurring an even number of times are discarded and the ones occurring an odd number of times are retained as expansion monoterms.

The advantage of the first method is that it requires less memory and each monoterm of the Boolean function is stored only once. The second method, however, stores all the monoterms of the cubes and there will be several copies of the same monoterms leading to a large array to be sorted and processed. The latter, however, is faster and does not have to check all the disjoint cubes for each monoterm to see if they have that monoterm as common.

A third method uses the second method but periodically performs the sorting and processing of the monoterms to reduce the memory requirements.

Another alternative, which is not yet implemented, is to use a combination of the first and the second techniques. One can partition the monoterms with some constant characteristic such as their number of DC-literals. Then once a new monoterm is generated, it can be compared with all the previous monoterms of the same characteristics to see if it needs to be discarded or kept. This has the advantage that is faster than the first method and requires less memory than the second method. Because this alternative is not implemented yet, it is not possible to compare its speed with the other two.

The methods implemented are compared in the Table 3.4 for several functions. REED is the program that existed before. reed0 is a program that uses barrel shifter approach in generation of the monoterms and the first method for the implementation of the ring sum operation. reed1 also uses barrel shifter approach in generation of the monoterms but uses the second method above in the implementation of the ring sum operation. reed2 uses Gray code approach in the generation of the monoterms and the second method above in the implementation of the ring sum operation. The functions are

all evaluated for polarity 111..1, or the *RMC* expansion.

Function	in	terms	REED	reed0	reed1	reed2
duke21	22	10	329.8u 2.9s	6.5u 0.1s	5.3u 0.4s	4.9u 0.2s
rd842	8	128	0.5u 0.1s	2.1u 0.2s	1.5u 0.2s	1.0u 0.1s
9sym	9	165	1.2u 0.2s	2.8u 0.1s	2.0u 0.2s	1.3u 0.1s
rd844	8	75	0.4u 0.2s	0.6u 0.1s	0.4u 0.1s	0.3u 0.1s
rd841	8	84	0.4u 0.2s	1.1u 0.1s	0.9u 0.2s	0.5u 0.1s
vg25	25	92	22061.1u 18.6s	107.1u 1.7s	77.1u 2.2s	61.7u 1.8s

Table 3.4. Comparison of *CGRM* Realization Programs

The improvement of the new techniques for the case of functions with large number of variables is self-evident. For functions with lower number of variables, the advantage of the program REED is when the function is comprised of a large number of cubes. The comparison of the programs shows that the program reed2 reaches to the time performance of REED in these cases as well. The programs were compared on a SUN 3 machine and the user and system CPU times in seconds are denoted by "u" and "s", respectively.

3.4. Minimal Polarity Vector

The main part of the minimization scheme is that of the identification of the minimal polarity vector. Once the minimal polarity vector is found, with the aid of the method introduced in the previous section, the minimal *CGRM* realization of the Boolean function can be generated. As the Boolean function is represented by a set of disjoint cubes, the properties of monoterms representing each cube and the commonality of the monoterms among different disjoint cubes are the parameters that determine the polarity with the least number of monoterms. Before describing the actual exhaustive and non-exhaustive methods developed for this purpose, the above mentioned properties of the monoterms representing disjoint cubes are described. In particular, the commonality of the monoterms is described in more detail. The search methods developed based on the properties of the monoterms for different polarities are then presented.

The most elementary case for Boolean functions given as disjoint cubes is that of a single cube. The monoterms representing a single cube for a given polarity vector were discussed before. Based on those results, one can determine the minimal polarity vector(s) which result in the smallest number of monoterms for this cube. The minimal polarity vector(s) for this special case are given by Theorem 3.3.

Theorem 3.3. The minimal polarity vector(s) for a single cube are polarity vector(s) which match all the literals in the cube. The number of such vectors is equal to $2^{N_{DC}}$ where N_{DC} is the number of DC-literals in the cube.

Proof. The proof follows from Theorem 3.2. It was argued that the number of monoterms representing a cube is given by 2^{N_v}, where N_v is the number of non-matching literals of the cube and the polarity vector. For minimal number of monoterms, one just needs to minimize the number of the non-matching literals. This will occur when each literal in the polarity vector is chosen exactly as the corresponding literal in the cube of the function. The DC literals in the cube can be matched by either 0 or 1 in the polarity vector, and hence these literals can take any of the two possible values. As the possible number of combinations of values for these DC literal positions is 2 to the power of number of DC literals, then one has the argument for the theorem. *QED*

Example 3.8 In Example 3.6, the minimal polarity vectors are given by 001-10, i.e. the vectors 001010 and 001110. To check this, one has:

$$\begin{array}{rr} \equiv & 001010 \\ & 001\text{-}10 \\ \hline & 111\text{-}11 \end{array}$$

The resulting monoterm is 111-11, which after performing the equivalence operation, $\equiv$, with the polarity vector will become 001-10, the cube itself. The same can be verified for the polarity vector 001110. □

When the Boolean function is represented by more than one cube, one first has to observe that the changes in the literals in the polarity vector are reflected as **column-wise** ≡ operations in the array of disjoint cubes. As the *i* th literal changes in the polarity vector, the *i* th position literals in each cube change their values. If the *i* th literal of one cube had value 1, it will change to 0; if it had value 0, it changes to the value 1; and if it is -, it stays the same. This is repeated for the next disjoint cube and so on.

Example 3.9 Let a Boolean function be represented by the following disjoint cubes:

0111-
1011-
00001
1101-
111--

By CCM analogy, the above cubes will be represented as the following for the polarity vector <0,1,1,0,1>:

1110-
0010-
10011
0100-
011--

It can be observed that there are column-wise changes on the first and the forth columns from the left. □

When the number of the disjoint cubes is greater than one, there will be certain interactions among the monoterms representing each cube that need to be investigated. It was stated in Theorem 3.2 that the number of monoterms representing one cube is equal to the 2 to the power of number of non-matching literals of the cube and the polarity vector. When there are more than one disjoint cube involved, the number of expansion monoterms is not the simple sum of the number of monoterms representing each cube. The actual number is this sum minus the total number of times the monoterms occurring

in an even number of cubes are represented and the total number of times the monoterms occurring in an odd number of cubes are represented more than once in this sum. Hence, the commonality of the monoterms among the cubes comes into the picture as well. Theorem 3.4 gives the number of expansion monoterms of a Boolean function.

Theorem 3.4. Let α_i be the number of monoterms that are common in i number of disjoint cubes. The number of expansion monoterms of a Boolean function for a given polarity vector, N_P, is given by:

$$N_P = \sum_{i=1}^{\eta} 2^{N_{V_{C_i}}} - \sum_{i=2}^{\eta} \lambda_i \tag{3.3}$$

$$\lambda_i = \begin{cases} \alpha_i \cdot (i-1) & \text{if } i \text{ is odd} \\ \alpha_i \cdot i & \text{if } i \text{ is even.} \end{cases}$$

where η is the number of disjoint cubes and $N_{V_{C_i}}$ represents the number of non-matching literals of the cube C_i and the polarity vector P.

Proof. The proof follows the argument given by Theorem 3.2. As each disjoint cube contributes 2^{N_V} different monoterms to the monoterms representing the Boolean function, the total number of the monoterms will be the sum of each of these monoterms if none of them are common. However, there will be some monoterms that occur in more than one disjoint cube. These monoterms, which are described in more detail in section 5.1, will be part of the monoterms representing the function. However, those monoterms that are common in an even number of disjoint cubes will be all subtracted from the total number. Those that occur in an odd number of cubes will be counted only once and all their other occurrences have to be subtracted as well.

For the monoterms that occur in an even number of cubes, the number of times that they are counted in $\sum_{i=1}^{\eta} 2^{N_{V_{C_i}}}$ is exactly i. By multiplying the number of these monoterms, α_i, by i, one obtains the total number of monoterms to be subtracted for

given i. This is the value of λ_i for the case when i is even. For the monoterms occurring in an odd number of cubes, the number of times they are counted in the total number of monoterms is again i. Multiplying the number of these monoterms, α_i, by $i-1$, gives the number of times that these monoterms are counted more than once in the total number. As these monoterms are among the expansion monoterms, only the number of times that they are counted more than once in the total number of monoterms needs to be subtracted. Therefore λ_i for the case of i being odd is $\alpha_i \cdot (i-1)$. The total number to be subtracted is then the summation of all these subtractions for 2, 3, ..., η cubes, which is represented by the second summation in Equation (3.3). *QED*

From Theorem 3.4, it can be inferred that the minimal polarity vector is the one which has the best balance between the least number of the all monoterms of the cubes and the number of monoterms that can be subtracted because they are common in several cubes. The overall number of monoterms for a given polarity vector was shown to be determined by the number of 0's in each cube. Hence, if one just attempts to minimize the overall number of monoterms, the reduction of $\sum_{i=1}^{\eta} 2^{N_{v_{C_i}}}$ would be of interest.

As the non-matching literals contribute to the number of overall monoterms and the change in polarity vector is a column-wise operation, the number of matches in a column plays an important role. One can notice that for a column i in the disjoint cube array, all the ith positions of the cubes with the matching value of the polarity literal will take the value of 1 and all with non-matching values will take the value 0. The –'s remain the same as shown in Example 3.9. So for each column, taking the polarity literal as the one which occurs the most in that column will result in the most number of 1's in the array and the least number of 0's. This, however, by itself does not guarantee that one gets the least number of overall monoterms.

Example 3.10. For the function given by the disjoint cubes in Example 3.9,

Choosing the polarity vector <1, 1, 1, 1, 1> will result in more number of 1's in the array as any other polarity vector. This array which is the same as the original array of disjoint cubes is shown below again:

0111-
1011-
00001
1101-
111–

It can be seen that the array is comprised of 13 1's, 7 0's and 5 -'s. This is found by using the value for a literal in the polarity vector which occurs the most in that column position.

It can be checked, however, that this polarity vector will not result in the least number of overall monoterms as one might suspect. The actual number of overall monoterms, using Equation (3.3) is 2 + 2 + 16 + 2 + 1 = 23. One can check to see that the polarity vectors which will result in the least number of overall monoterms are actually the polarity vectors <1, 0, 1, 1, 1> and <1, 1, 0, 1, 1>, each having 19 overall monoterms. The resulting array for the polarity vector <1, 0, 1, 1, 1> is shown below:

0011-
1111-
01001
1001-
101–

The number of monoterms is 4 + 1 + 8 + 4 + 2 = 19 as indicated. The array, however, is not comprised of more 0's than the original polarity. This array is comprised of 12 1's, 8 0's and the same number of –'s. □

As can be seen by the above example, it is not the overall number of 0's which needs to be minimized but their distribution is important also, when one is striving for the least number of overall monoterms. The least number of 0's in the array, however, is a good starting point and this is actually used for the heuristic search method presented later on.

As indicated by Theorem 3.4, the number of overall monoterms is still not the number of exact monoterms that represent the Boolean function. The other factor in this regard is the number of monoterms that are common and are subtracted from this number of overall monoterms. Example above can be used again to show this point.

Example 3.11. Although polarity vectors <1, 0, 1, 1, 1> and < 1, 1, 0, 1, 1> are the vectors which result in the least number of overall monoterms, they are not the minimal polarity vectors. The minimal polarity vector can be checked for this problem to be <0, 0, 0, 0, 0> which results in 8 monoterms. <1, 1, 1, 1, 1> results in 21 monoterms; i.e. only 2 are subtracted from the overall. <1, 0, 1, 1, 1> results in 15 monoterms; i.e. 4 monoterms are subtracted; and <1, 1, 0, 1, 1> also results in 15. □

One can then see that for certain functions, the number to be subtracted plays a dominant role. There are many functions for which the polarity vector(s) resulting in the least number of 0's in the array are also the minimal polarity vector(s). For other functions, the number of overall monoterms is of determining factor in identification of the minimal polarity vector. As the number of the monoterms that are common among cubes plays an important role, the characteristics of the commonality is described in more detail in the next section.

3.4.1. Subtracting Monoterms of Disjoint Cubes

It was mentioned before that the monoterms that are common in disjoint cubes are subtracted from the overall number of monoterms. Depending on the number of occurrences of these monoterms, the number of these monoterms can be different. The monoterms that are common in two cubes are all subtracted, while the ones that are common in three cubes will have two of each of them subtracted. This can be extended for all the monoterms that are represented in either an even or an odd number of disjoint cubes. In order to identify each of these categories of monoterms, one has to start from the commonality of monoterms in two disjoint cubes. Once this commonality is

established, one can extend this notion to larger number of cubes.

Commonality is not, however, the only issue that needs to be considered when one is dealing with the notion of subtracting monoterms. If one tries to apply Equation (3.3), the exact number of monoterms occurring in two, three, ..., η number of disjoint cubes needs to be precisely indicated.

In this section the commonality of monoterms of disjoint cubes will be defined and its properties discussed. This is done first by defining the operation Γ between two cubes which gives their common monoterms. Properties of this operation are described afterwards. Next, Commonality of monoterms of more than two cubes is presented. This is followed by the discussion of subtracting monoterms with a study of maximum number of subtracting monoterms which follows the discussion.

3.4.1.1. Commonality of Monoterms of Disjoint Cubes

Two cubes are the most elementary case where common monoterms could exist and that is where the issue of commonality is started. A monoterm is called common among two cubes if it is a member of both sets of monoterms that represent each cube. The cube commonality is a cube representing the monoterms that are common in the two cubes. The commonality operation, Γ, creates this new cube and is given below:

Definition 3.9. Let C_1 and C_2 be two cubes. The *cube commonality* operator on C_1 and C_2 is:

$$C_1 \Gamma C_2 = \begin{cases} \varnothing & \text{if } any\ C_{1i} \Gamma C_{2i} = \varnothing \\ C_3 & otherwise, where\ C_{3i} = C_{1i} \Gamma C_{2i} \end{cases} \tag{3.4}$$

where C_{ki} represents the ith literal of the cube C_k and the commonality operator for a single bit is defined in Table 3.5.

It can be seen from Table 3.5 that no commonality exists between two cubes when a DC-literal in one cube corresponds to a 1-literal in the other cube. The table can be

arrived at by recalling that the monoterms representing a cube match all of its 1-literals, have DC's in the same positions as DC-literals of the cube and have either 1 or DC in 0-literal positions of the cube.

Γ	0	1	–
0	0	1	–
1	1	1	∅
–	–	∅	–

Table 3.5. Cube Commonality Operator for a Single Bit

Example 3.12. The monoterm commonality of the two cubes 10-00-1 and 01010-1 is given as:

$$\begin{array}{ll} \Gamma & \begin{array}{l} 10\text{-}00\text{-}1 \\ 01010\text{-}1 \end{array} \\ \hline & 11\text{-}10\text{-}1 \end{array}$$

This can be checked as the monoterms representing 10-00-1 are:

1----1	11---1
1---1-1	11--1-1
1--1--1	11-1--1
1--11-1	11-11-1

and the ones representing 01010-1 are:

-1-1--1	11-1--1
-1-11-1	11-11-1
-111--1	1111--1
-1111-1	11111-

As it can be seen, the common monoterms of the two cubes are 11-1--1 and 11-11-1. Expanding the monoterm commonality of the two cubes, i.e. 11-10-1, the result will be 11-1--1 and 11-11-1 which matches the expected result. □

Alternatively, one can negate the value of the literals in the cubes and use bit-wise AND operation to get the cube that represents the common monoterms. This procedure is shown in Example 3.13 below:

Example 3.13. The literals in the two cubes 10-00-1 and 01010-1 can now be inverted to give 01-11-0 and 10101-0. The bit-wise AND operation on the two cubes

results in the cube 00-01-0. Again, bit-wise inverted, this cube results in the cube 11-10-1, found earlier in Example 3.12. □

The monoterm commonality of two cubes has the following properties:

i) *Commutativity*: $C_1 \Gamma C_2 = C_2 \Gamma C_1$, where C_1 and C_2 are two cubes.

ii) *Associativity*: $(C_1 \Gamma C_2) \Gamma C_3 = C_1 \Gamma (C_2 \Gamma C_3)$

From the associativity of Γ, one can infer the commonality of monoterms of more than two cubes. The common monoterms of three disjoint cubes are shown in Example 3.14 below.

Example 3.14. The common monoterms of the disjoint cubes 01110, 00110, and -1011 are:

$$\begin{array}{ll} \Gamma & 01110 \\ & 00110 \\ & -1011 \\ \hline & -1111 \end{array}$$

Again this result can be evaluated by expanding and observing the common monoterms of the cubes. The monoterms representing 01110 are:

-111-
-1111
1111-
11111

The ones representing 00110 are:

--11-	1-11-
--111	1-111
-111-	1111-
-1111	11111

And those representing -1011 are:

-1-11
-1111

As it can be seen, the only common monoterm of the three cubes is -1111 which was given by operation Γ on them. □

3.4.1.2. Subtracting Monoterms of More Than Two Cubes

Subtracting monoterms in the case of 3 or more disjoint cubes are not just the common monoterms and as such require further investigation. When the array of disjoint cubes is comprised of only 2 cubes, the subtracting monoterms are exactly the common monoterms. As these occur twice, the number to be subtracted is 2 times the number of such monoterms. In the case of more than two cubes, however, the subtracting monoterms are a function of the common terms and their number varies with their commonality as well. This section is devoted to the specification of these subtracting monoterms.

For getting a feeling about these subtracting monoterms, some set-theoretic analogies will be used. Here each cube can be considered as a set with the monoterms representing it being the elements of this set. Then the common monoterms can be represented simply as the intersections of these sets. This analogy will be shown with the use of Venn diagrams. Figure 3.2 shows all possible intersections of three sets. In this figure, the sets are represented by A, B, and C respectively. The intersection of the sets is represented through their products and the union of any subsets with the sum notation, +.

The expansion and subtracting monoterms can now be viewed in these figures. In Figure 3.2a) the monoterms in $A-(B+C)$, $B-(A+C)$, and $C-(A+B)$ all occur only once. The monoterms in $AB-ABC$, $AC-ABC$, and $BC-ABC$ occur twice. Finally, the monoterms in ABC occur three times, as they are common in all three cubes. Identifying the subtracting monoterms, from Equation (3.3), one sees that α_2 is comprised of all the monoterms in $AB-ABC$, $AC-ABC$, and $BC-ABC$. α_3 is comprised of the monoterms in ABC. Now the number of subtracting monoterms is $2\cdot(\alpha_2+\alpha_3)$. Using the notion of **symmetric difference**, $\oplus$, one can confirm that the monoterms to be subtracted are those in $AB \oplus AC \oplus BC$. Recall that the symmetric difference of two sets is defined as:

$$A \oplus B = \{x \mid (x \in A \wedge x \notin B) \vee (x \in B \wedge x \notin A)\} \quad (3.5)$$
$$= (A - B) \cup (B - A)$$

One can verify that ABC is also included in the symmetric difference of AB, AC, and BC. It is actually the α_3 part of the subtracting monoterms.

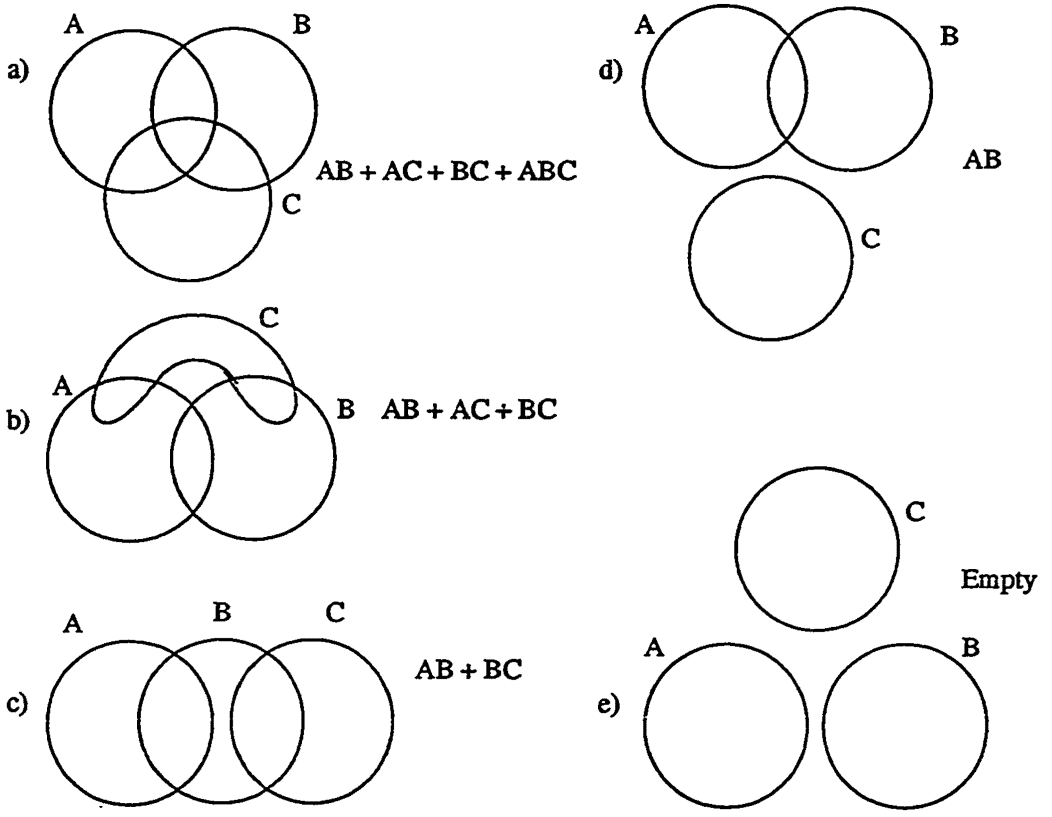

Figure 3.2 All Possible Intersections of Three Sets

In Figures 3.2b), c), d), and e), the symmetric difference can also be used for the subtracting monoterms. The differences are that in b) ABC is not part of the subtracting monoterms and it is not in the equation either. In c), d), and e) the same argument holds with the distinction that AC, BC, and AB are absent correspondingly. One then has the following theorem:

Theorem 3.5. Let C_1, C_2, and C_3 be three disjoint cubes. Let $\Psi^2_{C_1, C_2, C_3}$ be the subtracting monoterms of the three cubes that occur twice. Then:

$$\Psi^2{}_{C_1, C_2, C_3} = C_1 \Gamma C_2 \oplus C_1 \Gamma C_3 \oplus C_2 \Gamma C_3. \tag{3.6}$$

For more than 3 cubes, the above theorem can be expanded according to the **Inclusion-Exclusion Principal** of the combinatorial mathematics. This is shown in Theorem 3.6.

Theorem 3.6. Let $C_1, C_2, \ldots C_n$ be n different disjoint cubes. Let $\Psi_{C_1, C_2, \ldots, C_n}$ be the total subtracting monoterms of the n cubes and $\Psi^k{}_{C_1, C_2, \cdots, C_n}$ be the subtracting monoterms of the n cubes that occur k number of times. Then:

$$\Psi_{C_1, C_2, \ldots, C_n} = \sum_{i=1}^{\lfloor n/2 \rfloor} \Psi^{2i}{}_{C_1, C_2, \cdots, C_n}. \tag{3.7}$$

where $\Psi^k{}_{C_1, C_2, \ldots, C_n}$ is the symmetric difference of the $C_{i_1} \Gamma C_{i_2} \Gamma \cdots \Gamma C_{i_k}$'s for all sets of k C_i's and $\lfloor n/2 \rfloor$ denotes $n/2$ for n being even and $\frac{n-1}{2}$ for n being odd.

Proof. The theorem is a variation of the inclusion-exclusion formula where only the exclusions ar presented and the proof can be found in e.g. ([154]).

Example 3.15. For the sets A , B , C , D , and E , the following can be observed:

$$\begin{aligned}
\Psi^2{}_{A, B, C, D, E} &= AB \oplus AC \oplus AD \oplus AE \oplus BC \oplus BD \oplus BE \\
&\quad \oplus CD \oplus CE \oplus DE \\
\Psi^4{}_{A, B, C, D, E} &= \bar{A}\bar{B}CD \oplus ABCE \oplus ACDE \oplus BCDE \\
\Psi_{A, B, C, D, E} &= \Psi^2{}_{A, B, C, D, E} \oplus \Psi^4{}_{A, B, C, D, E}. \square
\end{aligned}$$

As monoterms common in an even number of disjoint cubes contribute the most to the subtracting monoterms, and the odd ones stay, the polarity vector that results in most number of subtracting monoterms can be identified. This is summarized in proposition 3.1 below:

Proposition 3.1. *For a given number of monoterms representing a set of disjoint cubes, the number of subtracting monoterms is largest when there are most number of monoterms common in an even number of disjoint cubes and least common in an odd number of cubes.*

3.4.2. Expansion Monoterms

Similar to the subtracting monoterms, one can identify the expansion monoterms and the operations that identify these monoterms. In the following, these monoterms are described and a simple method for their generation are presented.

The expansion monoterms can again be determined by application of the inclusion-exclusion principal. As exclusion part helped previously in the determination of the subtracting monoterms, the inclusion part can be used to identify those monoterms that occur an odd number of times and are retained. For the case of three sets, A, B, and C, one can verify that the monoterms in $A-(B+C)$, $B-(A+C)$, $C-(A+B)$ which occur only once and ABC are the ones that are retained as the monoterms representing the three sets. One can further verify that the monoterms that are common in an odd number of sets are nothing but the symmetric difference of these three sets. This can be further generalized to include the cases with more than three sets as in Proposition 3.2 below:

Proposition 3.2. *The expansion monoterms of a set of disjoint cubes are the result of symmetric difference of all the cubes.*

Proof. Induction will be used for the proof of the Proposition. For two cubes, the *symmetric difference* will give the monoterms that are not common in the two, and hence, the monoterms occurring in an odd number of cubes. Now, let us assume that it is true that the expansion monoterms of n cubes result from the *symmetric difference* of all of them. It needs to be shown that by adding one more cube, the expansion monoterms of this new function are the result of the *symmetric difference* of all the $n+1$ cubes. For the $n+1$ cubes, due to the associativity of the *symmetric difference*, the *symmetric difference* of all $n+1$ cubes is the same as the *symmetric difference* of the $n+1$th cube with the result of the *symmetric difference* of the previous n cubes. The *symmetric difference* of the n cubes, by assumption, gives the monoterms that have occurred an odd

number of times in the n cubes. Now, if there exists a monoterm that occurs in the n + 1th cube which is also present in the *symmetric difference* of the previous n cubes, the *symmetric difference* will remove it. This monoterm had occurred an odd number of times in the n cubes, and with the n + 1th cube it is now occurring an even number of times. So the monoterms occurring an even number of times in the n +1 cubes are not represented in the *symmetric difference* of the n +1 cubes. Similarly, if there exists a monoterm representing the n + 1th cube which does not occur in the *symmetric difference* of the previous n cubes, the *symmetric difference* of the n + 1th cube with the first n cubes retains this monoterm. This monoterm either did not occur at all or occurred an even number of times in the n cubes and now with the n + 1th cube, it is occurring an odd number of times. Therefore, the *symmetric difference* gives only those monoterms that occur an odd number of times in the n + 1 cubes. This proves the induction step. *QED*

Example 3.16. For the sets $A, B, C, D,$ and $E,$ in Example 3.15, the elements in the following subsets will be retained. Let Ω represent the set of retained elements. Then:

$$\Omega = A - (B + C + D + E) + B - (A + C + D + E) + C - (A + B + D + E) + D - (A + B + C + E) + E - (A + B + C + D +) + ABC \oplus ABD \oplus ABE \oplus ACD \oplus ACE \oplus ADE \oplus BCD \oplus BDE \oplus CDE \oplus ABCDE$$

One can verify that

$$\Omega = A \oplus B \oplus C \oplus D \oplus E. \square$$

The symmetric difference proves to be a very useful operation in dealing with monoterms representing a set of cubes, this operation will be further discussed. As it is evident from the definition of the symmetric difference, the main operation to be defined is that of the difference operation, $-$. This operation is the counterpart of the Sharp operation for monoterms and is defined in the following:

Definition 3.10. Let C_1 and C_2 be two cubes. Their *difference* is:

$$C_1 - C_2 = \begin{cases} C_1 & \text{if } C_{1i} - C_{2i} = \varnothing \text{ for } \textit{some } i \\ \varnothing & \text{if } C_{1i} - C_{2i} = \varepsilon \text{ for } \textit{all } i \\ \bigcup_i & (C_{11}, C_{12}, \ldots, \alpha_i, \ldots, C_{1n}) \textit{ otherwise} \\ & \textit{where the } \bigcup \textit{ is for all } i \text{ for which} \\ & C_{1i} - C_{2i} = \alpha_i \in \{-, 1\} \end{cases} \tag{3.8}$$

where C_{ki} represents the ith literal of the cube C_k and the literal difference operator is defined in Table 3.6.

–	0	1	–
0	ε	–	1
1	ε	ε	∅
–	ε	∅	ε

Table 3.6. Cube Literal Difference

Example 3.17. The two cubes 10-00-1 and 01010-1 of Example 3.12 will be examined for their differences. First the difference of 10-00-1 and 01010-1 will be shown. The generation of the monoterms and their commonality were shown in that example before. One would expect that the difference will give all the monoterms representing the cube which are not in their commonality. First Table 3.6 is used bitwise:

$$\begin{array}{rl} - & \text{10-00-1} \\ & \text{01010-1} \\ \hline & \varepsilon\text{-}\varepsilon\text{-}\varepsilon\varepsilon\varepsilon \end{array}$$

In the resultant cube $\alpha_i \neq \varepsilon$ ($\alpha_i = -$) for $i = 2, 4$. Then, according to Equation (3.8), the results of the Difference operator are created from $C_1 = 10\text{–}00\text{–}1$ by inserting DC (symbol "–") in positions 2 and 4 consecutively which leads to {1–-00-1, 10–-0-1}.

Now, it can be verified that the *difference* of the two cubes, C_1 and C_2 is exactly given by the cube set {1–-00-1, 10–-0-1}. The monoterms representing the cube set are:

1----1	1--11-1
1---1-1	11---1
1--1--1	11--1-1

The monoterms that are not listed above and represent C_1 are 11-1--1 and 11-11-1, the commonality of the two cubes. Note that the monoterms in 1---0-1, the commonality

of 1--00-1 and 10--0-1, occur twice, but due to the property of union in the definition of the difference operation, are shown only once. The same can be repeated for the difference of 01010-1 and 10-00-1. This is shown below:

$$\begin{array}{rl} - & 01010\text{-}1 \\ & 10\text{-}00\text{-}1 \\ \hline & \text{-}\varepsilon 1\varepsilon\varepsilon\varepsilon\varepsilon \end{array}$$

By inserting DC [-] in the first literal of 01010-1, one gets -1010-1. Inserting 1 in the third literal results in 01110-1. This leads to the *difference* $C_2 - C_1$ = {-1010-1, 01110-1} which results in the following monoterms:

-1-1--1	-1111-1
-1-11-1	1111--1
-111--1	11111-1

Here again, comparing with Example 3.12, the monoterms representing 11-10-1 are absent from 01010-1 which is the commonality of the two cubes 01010-1 and 10-00-1 which is expected. The *symmetric difference* will then be $C_1 \oplus C_2$ = {1--00-1, 10--0-1, -1010-1, 01110-1}. □

Now the monoterms representing any number of cubes can be given explicitly with operation of symmetric difference and Proposition 3.2. This can also be regarded as another alternative for implementation of the ring sum operation in CCM. Depending on the number of disjoint cubes and their symmetric differences, this method of symmetric difference of the disjoint cubes can be comparable with other methods. This also requires further study. As an example of this method, three cubes are evaluated below:

Example 3.18. The monoterms representing the three cubes in Example 3.14 are now directly presented. The three cubes were 01110, 00110, and -1011. Let C_1, C_2, and C_3 represent these cubes accordingly. First the monoterms present in single cubes will be shown. This is simply the union of $C_1 - (C_2 + C_3)$, $C_2 - (C_1 + C_3)$, and $C_3 - (C_1 + C_2)$. The only commonality of the three cubes is that of $C_1 \Gamma C_2 \Gamma C_3$. Now each of them is evaluated separately.

$C_1 - (C_2 + C_3)$ can be represented also as $(C_1 - C_2)\,\Gamma\,(C_1 - C_3)$. This is shown first.

$$\begin{array}{rr} - & 01110 \\ & 00110 \\ \hline & \varepsilon\varepsilon\varepsilon\varepsilon\varepsilon \end{array}$$

The difference is then empty by definition.

$$\begin{array}{rr} - & 01110 \\ & -1011 \\ \hline & 1\varepsilon\varepsilon\varepsilon- \end{array}$$

The result is the two cubes 11110 and 0111-. The commonality of these cubes is then $\varnothing$ as one of the cubes is empty. Now $(C_2 - C_1)\,\Gamma\,(C_2 - C_3)$ will be evaluated.

$$\begin{array}{rr} - & 00110 \\ & 01110 \\ \hline & \varepsilon\text{-}\varepsilon\varepsilon\varepsilon \end{array}$$

resulting in the cube 0-110.

$$\begin{array}{rr} - & 00110 \\ & -1011 \\ \hline & 1\text{-}\varepsilon\varepsilon- \end{array}$$

The result are the three cubes 10110, 0-110, and 0011-. The commonality of 0-110 with these three cubes will be the union of commonality of 0-110 with each of them.

$$\begin{array}{rr} \Gamma & 0\text{-}110 \\ & 10110 \\ \hline & 1\text{-}110 \end{array}$$

$$\begin{array}{rr} \Gamma & 0\text{-}110 \\ & 0\text{-}110 \\ \hline & 0\text{-}110 \end{array}$$

$$\begin{array}{rr} \Gamma & 0\text{-}110 \\ & 0011- \\ \hline & 0\text{-}11- \end{array}$$

Now $(C_3 - C_1)\,\Gamma\,(C_3 - C_2)$ will be derived.

$$\begin{array}{rr} - & \text{-1011} \\ & 01110 \\ \hline & \varepsilon\varepsilon\text{-}\varepsilon\varepsilon \end{array}$$

resulting in the cube -1-11.

$$\begin{array}{rr} - & \text{-1011} \\ & 00110 \\ \hline & \varepsilon\varepsilon\text{-}\varepsilon\varepsilon \end{array}$$

which is the same as above. The Γ of the two differences is again -1-11. The commonality of the three cubes was seen in Example 3.14 to be -1111. The monoterms representing these three cubes are then union of 0-110, 1-110, 0-110, 0-11-, -1-11, and -1111 which can be simplified as 0-110 and -1011. One way to check this result is to generate all the monoterms representing each cube and retaining only the ones which occur an odd number of times. Another is to notice that the two cubes 01110 and 00110 are adjacent and can be combined to form the larger cube 0-110. This new cube and -1011 do not have commonality. This matches with the previous result. Now direct application of the symmetric difference will be examined.

$C_1 - C_2$ was seen to be empty and $C_2 - C_1$ was 0-110. The symmetric difference of the first two cubes is then the union of 01110 and 0-110 which is again 0-110. One can then observe that the difference of this result with the third cube, -1011 and the difference of -1011 and 0-110 are both empty. Hence the monoterms representing the three cubes are obtained by the union of -1011 and 0-110 which matches with the result above. □

The total number of monoterms representing a set of disjoint cubes can be found through the same inclusion-exclusion principle. Noticing that the inclusion-exclusion formula gives the number of elements in a set of sets, one can modify the argument to

subtract the number of elements that occur in an even number of sets. With this modification one has the following proposition giving the number of expansion monoterms of a set of disjoint cubes:

Proposition 3.3. *Let* $\{C_{i_1}, C_{i_2}, \cdots, C_{i_n}\}$ *be a set of n disjoint cubes. Let* S_k *denote the sum of the number of all monoterms common in all possible k cubes. The number of expansion monoterms of the set of cubes equals:*

$$\sum_{k=1}^{n} (-2)^{k-1} S_k = S_1 - 2S_2 + 4S_3 - 8S_4 + \cdots + (-2)^{k-1} S_k + \cdots + (-2)^{n-1} S_n \quad (3.9)$$

In the above equation, S_1 denotes the sum of the number of monoterms of individual cubes, S_2 denotes the number of all monoterms that are common in any two cubes, etc.

Proof. As the number of expansion monoterms of the set of cubes is equal to the number of monoterms occurring in an odd number of cubes, one seeks to identify this number. S_1 gives the sum of the number of the monoterms representing all individual cubes. This number includes both the number of the monoterms that repeat more than once and need to be subtracted and those that occur only once and need to be included in the final number. Using induction, the number of monoterms that occur only in one cube are counted once, so Equation (3.9) accurately gives the number of monoterms for the case of the monoterms which occur only in one cube. It needs to be proven that for a monoterm occurring in more than one cube, if it occurs in an odd number of cubes, it is counted once in Equation (3.9) and if it occurs in an even number of cubes, it is not counted at all.

Therefore, in general, let us assume that the monoterms common in $m-1$ cubes are accurately represented by Equation (3.9), n being equal to $m-1$. Let μ_i be a monoterm that occurs in exactly m cubes. Using the identity $\binom{m}{k} = \binom{m-1}{k} + \binom{m-1}{k-1}$, the net count of μ_i in (3.9) is:

$$\sum_{k=1}^{m}(-2)^{k-1}\binom{m}{k} = \sum_{k=1}^{m}(-2)^{k-1}\left[\binom{m-1}{k} + \binom{m-1}{k-1}\right] \tag{3.10}$$

As now k can go from 1 to $m-1$ and substituting l for $k-1$, the above summation on the right can be written as:

$$= \sum_{k=1}^{m-1}(-2)^{k-1}\binom{m-1}{k} + \sum_{l=0}^{m-1}(-2)^{l}\binom{m-1}{l} \tag{3.11}$$

The first sum in the left is the case of monoterms common in $m-1$ cubes. The sum in the right can be further simplified using the following binomial expansion:

$$(1+x)^m = 1 + \binom{m}{1}x + \binom{m}{2}x^2 + \cdots + \binom{m}{m}x^m.$$

It can be verified that the summation on the right can be written as:

$$\sum_{l=0}^{m-1}(-2)^{l}\binom{m-1}{l} = (1-2)^{m-1} = (-1)^{m-1} \tag{3.12}$$

Now, as the first sum in Equation (3.11) gives the number of monoterms common in $m-1$ cubes and it was assumed that this number is correct, then there are two possibilities to consider. One that $m-1$ is odd and the other that $m-1$ is even.

If $m-1$ is odd, then it means that a monoterm such as μ_i, which is common in exactly m cubes, is counted only once in the left summation in Equation (3.11). In addition, $m-1$ being odd means that in the right summation in Equation (3.11) $(-1)^{m-1} = -1$. So the total number of occurrences of μ_i in (3.11) and therefore (3.9) is 1 + (-1) = 0. As $m-1$ is odd, m is even and μ_i occurs 0 times, so Equation (3.9) holds for the case of m being even.

Now, if $m-1$ is even, then it means that μ_i does not appear in the left summation in Equation (3.11). In the right summation in Equation (3.11), however, $(-1)^{m-1} = 1$ so the total number of occurrences of μ_i in (3.11) and therefore (3.9) is 0 + 1 = 1. Hence, Equation (3.9) for an odd number of cubes gives only 1 occurrence of the monoterm μ_i and it also holds for an odd number of cubes, proving the induction step. Any monoterm that is common in an even number of cubes is not counted and any monoterm that is common in an odd number of cubes is counted only once. Equation (3.9) then correctly

gives the number of expansion monoterms. *QED*

3.4.3. The Minimal Polarity

Identification of the minimal polarity vector, although being an NP-hard problem, is far from being just a blind search. From the material presented up to now, and some results to be presented shortly, one can identify cases which the minimal polarity vector can be found without the need for an exhaustive search or the search can be reduced. In this section, based on the previous results, certain characteristics of the minimal polarity vector will be described. These results will be used in devising minimization algorithms that are presented in section 6.

The minimal polarity vector can be basically viewed as one that results in a balance between the small number of overall monoterms and the large number of subtracting ones. Theorem 3.4 gave the argument for the number of monoterms representing a set of disjoint cubes for a given polarity vector. The least number of overall monoterms and the most number of subtracting monoterms were discussed succeedingly. As the least number of overall monoterms was shown to be close to the result of the polarity vector which results in the least number of zeros in the array of disjoint cubes, the main thrust of identification of the minimal polarity vector lies with the subtracting monoterms. In other words, the changes in the polarity vector which result in the most number of monoterms in an even number of disjoint cubes and the least number in an odd number of them are the criteria that require special attention.

The main characteristic of the array of disjoint cubes is that any two cubes will have at least one corresponding literal of opposite polarities. This would mean that if any two disjoint cubes have any common monoterms, they would have at least one 1 in their commonality. If the commonality of certain disjoint cubes is comprised of all 1's, the commonality of these with any other cube, if the commonality exists, will also be comprised of just 1's. This is due to the fact that the commonality of 1 with either 1 or 0

is always 1. Another observation is that when the commonality of several cubes is empty, the commonality of these cubes with any other cube will also be empty. A third observation is that the commonality of any number of cubes is always larger than or equal to their commonality with additional number of cubes. This can be used as certain heuristics to consider the commonality of two and three cubes as the main contributers to the subtracting monoterms. For those arrays with large number of cubes, the requirement of opposite polarities for corresponding literals makes it more probable to get more 1's in the commonality as the number of cubes is increased.

As the number of monoterms and their commonality is of interest, the columns in the array of disjoint cubes can be divided into those with DC-literals and those without. The columns with no DC-literals can be further divided into those that are all comprised of one value and those that have both 0's and 1's. These columns will not influence the commonality of the cubes but only their sizes. The commonalities or their lack of are determined by the columns which include DC-literals as they can cause the commonality of certain cubes to be empty (if there are corresponding 1's and DC's) or non-empty (if there are corresponding 0's and DC's). Proposition 3.4 provides a criterion for identifying the minimal polarity literal based on the columns of the array of disjoint cubes, using Proposition 3.3.

Before stating that Proposition, a notation of grouping for the cubes in the array will be presented. This grouping for any column is used later in the proposition.

Notation:

$S^1{}_i$:	S_i for the cubes which have a value of 1 in a given column.
$S^0{}_i$:	S_i for the cubes which have a value of 0 in a given column.
$S^{DC}{}_i$:	S_i for the cubes which have a DC value in a given column.
$S^2{}_i$:	S_i for the cubes which have either a value of 1 or a value of 0 in a given column.
$S^{1-DC^*}{}_i$:	S_i for the cubes which have either a value of 1 or a DC value in a given column. †

† The * denotes the fact that in the calculation of these S_i's, the 1's in that column should be re-

$S^{0\text{–}DC}{}_i$: S_i for the cubes which have either a value of 0 or a DC value in a given column.

These notations will be shown in the following example:

Example 3.19. Let a function be represented as the array of disjoint cubes shown below:

```
1   001-110-10
2   0-10001-11
3   011101--1-
4   00-1100-1-
5   000-001010
6   1100--1-11
```

For the first column on the right, S^1 refers to the summations regarding the 2nd and the sixth cube; S^0 to the first and the fifth; S^{DC} to the third and the forth; S^2 to the first, second, fifth, and the sixth; $S^{1\text{–}DC^*}$ to the second, third, forth, and sixth; and finally $S^{0\text{–}DC}{}_i$ refers to the first, third, forth, and the fifth cube. □

Proposition 3.4. *Let a function be given as an array of disjoint cubes. The polarity literal corresponding to a column in the array would be minimal if it is only changed when*

$$\begin{aligned} &S^1{}_1 - 2S^1{}_2 + 4S^1{}_3 - 8S^1{}_4 + \ldots + (-2)^{k-1}S^1{}_k + \cdots + (-2)^{n-1}S^1{}_n \\ &- 2S^{1\text{–}DC^*}{}_2 + 4S^{1\text{–}DC^*}{}_3 - 8S^{1\text{–}DC^*}{}_4 + \ldots + (-2)^{k-1}S^{1\text{–}DC^*}{}_k + \cdots + (-2)^{n-1}S^{1\text{–}DC^*}{}_n \\ &< 1/2\,[S^0{}_1 - 2S^0{}_2 + 4S^0{}_3 - 8S^0{}_4 + \ldots + (-2)^{k-1}S^0{}_k + \cdots + (-2)^{n-1}S^0{}_n] \\ &- 2S^{0\text{–}DC}{}_2 + 4S^{0\text{–}DC}{}_3 - 8S^{0\text{–}DC}{}_4 + \ldots + (-2)^{k-1}S^{0\text{–}DC}{}_k + \cdots + (-2)^{n-1}S^{0\text{–}DC}{}_n \end{aligned} \tag{3.13}$$

Proof. The proof follows from the fact that when a 1-literal in the cube is changed to 0, the number of monoterms representing that cube is doubled, and when a 0-literal is changed to 1, the number of monoterms is divided by two. Now, for a column, changing the polarity has the effect of changing every 1 in the corresponding literal position of every cube to 0 and changing every 0 into 1 with DC's remaining the same. In order to

placed with 0's. This is due to the changing of polarities. Now with this change, the cubes with literal values of 1 and those with DC literal values can have commonalities (referring to Table 3.5) while before the change they could not.

examine the effect of the change of the polarity literal on the number of monoterms, the cubes will be divided into different groups described before the statement of Example 3.19. Now using Equation (3.9), the number of expansion monoterms of the cubes before the change of polarity for the column and after it can be written in terms of these groups. Before change, that number is:

$$\sum_{k=1}^{n}(-2)^{k-1}S^{1}{}_{k}+\sum_{k=1}^{n}(-2)^{k-1}S^{0}{}_{k}+\sum_{k=1}^{n}(-2)^{k-1}S^{DC}{}_{k}+ \tag{3.14}$$

$$\sum_{k=2}^{n}(-2)^{k-1}S^{2}{}_{k}+\sum_{k=2}^{n}(-2)^{k-1}S^{0\text{-}DC}{}_{k}$$

The last two summations start from $K=2$ because the smallest number of cubes which can have either 1 and 0 or 0 and DC is 2. As the commonality of any two cubes which have a 1 and a DC in the same literal position is $\varnothing$, given by Table 3.5, the above separate sums are the only possible combinations of 1, 0, and DC groups which contribute to the number of monoterms.

Once the polarity of the corresponding literal is changed, there would be certain changes in the number of S_i's. For one, the 0's in that column will become 1's, 1's become 0's and the DC's stay the same. This has the effect of doubling the number of monoterms in cubes with previous values of 1 in the column and cutting the number of expansion monoterms of the cubes with previous values of 0 in the column into half. However, the number of the expansion monoterms of the cubes with values of DC in the column stays the same. This is due to the fact that DC's stay the same and DC literals do not contribute to the number of monoterms representing a cube. The same is true for the S^2's, as the commonality of a literal 1 and 0 is 1. With the change of polarity in the column, 1's will change to 0's and 0's change to 1 but the commonality stays 1 as before. In terms of commonalities, the cubes which had a value of 0 will no longer have commonality with those cubes that have DC values This is due to the fact that 0's are now changed to 1's. DC's and 1's, according to Table 3.5, can not have commonalities. Those cubes that had a value of 1 could now have commonality with those cubes with

DC values unless there are other corresponding literal positions of 1 and DC which would prohibit commonalities. These changes will be reflected in the number of monoterms to become:

$$2\sum_{k=1}^{n}(-2)^{k-1}S^{1}{}_{k}+1/2\sum_{k=1}^{n}(-2)^{k-1}S^{0}{}_{k}+\sum_{k=1}^{n}(-2)^{k-1}S^{DC}{}_{k} \qquad (3.15)$$
$$+\sum_{k=2}^{n}(-2)^{k-1}S^{2}{}_{k}+\sum_{k=2}^{n}(-2)^{k-1}S^{1-DC^{*}}{}_{k}$$

Now, if this number of monoterms is less than the number before changing the polarity literal, then there should be a change; otherwise, if it results in a larger number of monoterms, then there will be no need of changing the polarity. Hence, the criteria will be:

$$2\sum_{k=1}^{n}(-2)^{k-1}S^{1}{}_{k}+1/2\sum_{k=1}^{n}(-2)^{k-1}S^{0}{}_{k}+\sum_{k=1}^{n}(-2)^{k-1}S^{DC}{}_{k} \qquad (3.16)$$
$$+\sum_{k=2}^{n}(-2)^{k-1}S^{2}{}_{k}+\sum_{k=2}^{n}(-2)^{k-1}S^{1-DC^{*}}{}_{k}$$
$$<\sum_{k=1}^{n}(-2)^{k-1}S^{1}{}_{k}+\sum_{k=1}^{n}(-2)^{k-1}S^{0}{}_{k}+\sum_{k=1}^{n}(-2)^{k-1}S^{DC}{}_{k}$$
$$+\sum_{k=2}^{n}(-2)^{k-1}S^{2}{}_{k}+\sum_{k=2}^{n}(-2)^{k-1}S^{0-DC}{}_{k}$$

Canceling the equal terms and simplifying, will give the result in (3.13). *QED*

For the special case of a column which is comprised of only one value, the following corollary results:

Proposition 3.5. *For a column comprised of all 0's or all 1's, the corresponding minimal polarity literal is the same as the value in the column. (If the opposite is chosen, the number of expansion monoterms of the cubes would be doubled.) For a column comprised of all DC values, either 0 or 1 will be the minimal literal value.*

Proof. This is a special case of Proposition 3.4, where there is only one value involved. Assume that the column is all comprised of 0's. Then using Equation (3.13), one should change the polarity of the column if

$$0 < 1/2 \sum_{k=1}^{n} (-2)^{k-1} S^0{}_k$$

as the number of 1's is 0 and there are no DC values involved. Since this is always true, the polarity of the column should be changed to 0. If the column is all comprised of 1's, then the polarity of the column should be changed if

$$\sum_{k=1}^{n} (-2)^{k-1} S^1{}_k < 0$$

as the number of 0's is 0 and there are no DC values involved. Since this is never the case, the polarity should stay as 1. When the column is comprised of all DC values, then changing the polarity does not change the number of monoterms since the number of monoterms representing each cube and the commonalities stay the same. Now, if the opposite polarity is chosen for the previous two cases, the number of the monoterms representing each cube as well as the commonalities will double. This would be reflected in the overall number of monoterms to become equal to $2 \cdot \sum_{k=1}^{n} (-2)^{k-1} S_k$, which is twice the number of monoterms in the original case. *QED*

When DC values are not involved in a column, Equation (3.13) can be further simplified. In this case, the polarity literal corresponding to a column in the array would be minimal if it is only changed when

$$\begin{aligned} & S^1{}_1 - 2S^1{}_2 + 4S^1{}_3 - 8S^1{}_4 + \ldots + (-2)^{k-1} S^1{}_k + \cdots + (-2)^{n-1} S^1{}_n \\ & < 1/2\,[S^0{}_1 - 2S^0{}_2 + 4S^0{}_3 - 8S^0{}_4 + \ldots + (-2)^{k-1} S^0{}_k + \cdots + (-2)^{n-1} S^0{}_n] \end{aligned} \qquad (3.17)$$

In general, the existence or lack of commonalities can affect the number of monoterms in the cubes which in turn affects the decision of changing the polarity of a column according to Equation 3.13. This then shows the interrelation between the columns for identification of the minimal polarity vector. Changing the polarity of one column will result in less monoterms depending on the polarity of the other columns. The changing of the same column can result in more number of monoterms for a different polarity of other columns. This is the property which results in the NP-hardness of the problem.

3.5. Minimization Schemes

With the characteristics of the minimal polarity vector at hand, it is possible to devise schemes for the identification of this vector. An array of disjoint cubes can be evaluated for certain characteristics and based on these characteristics, the minimal polarity vector can be found. Based on these characteristics, two different algorithms are devised for the identification of the minimal polarity vector. Both of these algorithms involve searching, one being "exhaustive" to give the exact solution, the other being "heuristic" to give a quasi-minimal solution. These algorithms are then used to come up with programs which are presented and evaluated against some benchmark functions. At the end, the algorithms are evaluated in terms of their complexity.

Proposition 3.5 provided the criteria for minimal polarity literal for a column composed of only one value. If such a column exists, the search space will be reduced by two to the power of number of such columns.

In addition, Equation (3.17) provided the criteria for the changing of polarity in columns with no DC values. Notice that if the number of occurrences of one value is much higher than the opposite value, it will be more likely that the polarity with the same value as the most occurring value in the column will result in less number of monoterms. That is, for example, if more than 80% of the cubes have a 1 in a column and 20% have a 0, then adding the number of the monoterms in the first group will be more than half the number of monoterms in the second group. The more the number of cubes, the higher is the likelihood of this characteristic. One can then use this as a heuristic for dealing with columns with such a property. The less the number of cubes and the less the ratio of the number of cubes with one value to the opposite value, the least uncertain one will be about the minimal polarity literal of that column.

Proposition 3.1 provided the characteristics of the polarity which results in more number of subtracting monoterms. It was conjectured that there would be more subtract-

ing monoterms when there are more commonalities in an even number of cubes and less in an odd number of them. In addition, it was shown that a lower number of overall monoterms results in the polarity literal which matches the value of the most occurring value in each column. The minimization scheme can then start with the lower number of overall monoterms as a starting point. The polarity can then be altered, depending on the patterns of values in the columns, to increase the number of subtracting monoterms. Although the number of overall monoterms could increase for many cases as shown in Example 3.11, the increasing number of subtracting monoterms still would yield a minimal solution. Identification of the pattern which results in most commonalities in an even number of cubes and least commonalities in an odd number of the cubes is essentially the major part of the minimization scheme.

As searching for patterns of commonalities involves mutual comparison of all the cubes, for large number of cubes and literals, this becomes essentially a time-consuming task. For functions comprised of relatively small number of literals, it is possible to perform an exhaustive search to yield the minimal solution. Certain columns can be possibly preset to their minimal polarities, i.e. columns which are comprised of only one value; the other columns can be searched exhaustively for their minimal polarity values. Moreover, for the functions which have large number of unknown minimal polarity literals, a heuristic search method has been developed. This method starts from the polarity with low overall number of monoterms and then uses certain heuristics to alter the polarity of certain columns to increase the number of subtracting monoterms.

3.5.1. The "Exhaustive" Search Approach

Although this section is titled exhaustive, in essence the method is not always exhaustive. As described in the previous section, there are arrays of disjoint cubes which include columns comprised of only one value. The "exhaustive" search method identifies these columns and presets these columns to their minimal polarity literals. In the cases

where the ratio of the number of occurrences of one value in a column to that of its opposite value are high, and no DC values are involved, again these columns are preset to the polarity of the most occurring value as their minimal polarity literals. The method is only exhaustive for the cases where neither of these cases exist in the array.

For all the columns which can not be preset to their corresponding minimal polarity literals, the exhaustive method will generate all possible combinations of polarity literals for these columns and determines the one(s) with the least number of monoterms. The monoterms for the given polarity are realized by the methods presented in section 4. The procedure for generation of the polarities for the undetermined columns is through a Gray code scheme. This method, which is similar to the method presented in section 4.2, requires one change of polarity literal at a time and hence is very fast. The exact method is presented in Figure 3.3:

```
/* The exact algorithm */
CGRMIN_EXACT
{
   If ( the function is comprised of one or two cubes )
   {
      Preset the columns to the minimal polarity;
      Generate the CGRM of the array;
   }
   else
   {
      Preset the columns
         to minimal polarity literals if possible, or
         to polarity 0 otherwise;
      for ( all combinations of the columns with undetermined polarities ) do {
         Set one column to its opposite polarity using Gray-code order;
         Generate the CGRM of the array;
         if ( The number of expansion monoterms of the array decreases )
            min_polarity = polarity_of_current_CGRM;
      }
      Generate the CGRM of min_polarity;
   }
}
```

Figure 3.3. The Exact Algorithm

The method is of order 2^{un_preset}, where *un_preset* is the number of columns which their

corresponding minimal polarity literals can not be identified without search.

3.5.2. The Heuristic Search Approach

In this section, a fast heuristic approach to the minimization problem is introduced. The corresponding heuristics combine the characteristics of the two parts of Equation (3.3), i.e. the overall number of monoterms and the subtracting ones, to identify the minimal *CGRM* polarity for a given array of disjoint cubes. Based on these heuristics, a method based on a priority of search for different polarities is devised and a minimization algorithm is introduced.

Here, similar to the Exact method, the columns minimal polarities of which can be identified without any search are first pre-set. Next, for all the remaining columns in the array of disjoint cubes, the polarity literals are chosen such that there will be the least number of 0's in each column. This is the polarity which is equivalent to the value that occurs the most in that column. Since the overall number of monoterms is determined by the number of zeros in the cubes, this polarity is used as a starting point for the minimization. For many functions it was shown experimentally that this first choice gives the minimal solution. For other cases, however, a search scheme is required. The general heuristics below use the results of the previous section and can be used in any minimization scheme:

Heuristics based on the number of monoterms:

- **Heuristic 1.** *As the number of monoterms for a cube is* 2^{N_0}, where N_0 *is the number of 0's in the cube, if a cube has a relatively large number of 0's, some polarity literals should be changed to reduce the contribution of this cube to the number of expansion monoterms.*
- **Heuristic 2.** *By Equation (3.13), if the number of occurrences of one value in a column with no DC values is much higher than the opposite value, it*

is more likely that this value will be the minimal polarity literal for that column. A ratio of 4 to 1 for the most occurring and least occurring values almost guarantees this likelihood.

- **Heuristic 3.** *When there are DC values involved in a column, the greater the number of the DC values, the less will be the overall number of monoterms affected if the polarity is changed.*

Heuristics based on the subtracting number of monoterms:

- **Heuristic 4.** *The monoterms common in two cubes and three cubes are the major contributing subtracting monoterms for large number of cubes with large number of literals.*

Now, based on the above heuristics, a priority is calculated for every column which can not be preset to its minimal polarity using the previous theorems. The column with the highest priority is the first one for which the polarity is changed. This priority is used to guide the search towards the minimal polarity. As indicated, this is a heuristic search and is based on certain weights assigned to the above heuristics. As the heuristics based on the subtracting number of monoterms requires a row-wise as well as a column-wise search, this heuristic will not be used in the minimization method described below. The main heuristics to consider are those based on the number of monoterms. The basic approach is to combine the second and third heuristics to come up with an order of priorities for column change based on column-wise considerations. This priority is checked against the rows with most number of 0's each time to include the first heuristic in the method.

One scheme for prioritizing column changes based on column considerations is to include the number of DC values in a column and the ratio of 1 and 0 values into one number. When the columns are adjusted to the polarity which results in the least overall monoterms, the majority of the values in every column will be 1. Of course that is when

not all literals in a columns are DC's or the number of 0's and 1's is equal. Then, whenever the polarity of a column is changed, the number of 0's will become the majority. As the number of 1's determines the number of 0's after change, the number of 1's in a column at polarity of the least overall monoterms is a number useful in heuristic priority. The ratio of the number of 1's to the total number of 1's and 0's in a column combines both the ratio criteria and the number of DC's in the column. The larger the denominator, the less will be the number of DC's and hence the column should be considered later. The larger the number of 1's, the change will make more 0's in the array, making it more likely to increase the number of monoterms. This ratio is multiplied by the number of 1's in the column and the smaller this number, the higher the priority for the column to be changed.

Let the *priority* of the column i be represented by γ_i. Let the number of the 1's in that column be represented by $N^1{}_i$, and the number of 0's by $N^0{}_i$. Then

$$\gamma_i = \frac{(N^1{}_i)^2}{N^1{}_i + N^0{}_i} \tag{3.18}$$

The Quasi-minimal method can now be presented formally as shown in Figure 3.4.

In this algorithm, the number of the searches required for the worst case is of order $2n$ for a function of n variables, which significantly contributes to its speed. In general, the order depends on the patterns of 0's and 1's in the disjoint cube array.

3.5.3. Analysis of the Minimization Methods

The exhaustive and heuristic methods developed are evaluated against MCNC benchmarks in this section. The evaluations include the quality of the heuristics as well as the timing performance of the programs. It is shown that the heuristic program, while providing a very fast method, is also not far from the exact solution for many of the functions tested.

```
/* The quasi-minimal algorithm */
CGRMIN
{
   Preset the columns
      to minimal polarity literals if possible, else
      to most occurring values in the columns;
         Find the values of column priorities, γs;
   Sort γ's in a descending order;
   for (k = 1; k < number_of_columns_with_undetermined_polarity; k++)
   {
      Set the column with priority k to its opposite polarity;
      Generate the CGRM of the array;
      if ( The number of expansion monoterms or
         the overall number of monoterms decreases )
      {
         min_polarity = polarity_of_current_CGRM;
         continue;
      }
      else
         change the column back to its previous polarity;
   }
   Generate the CGRM of min_polarity;
}
```

Figure 3.4. The Quasi-Minimal Algorithm.

Table 3.7 shows the comparison of the exact and heuristic fixed polarity form with Two-level AND/OR. In this table "in" stands for the number of the input variables in the functions. The minimal Two-level AND/OR is achieved by ESPRESSO [111], the disjoint representation by DISJOINT [44], and the fixed polarity forms by CGRMIN.

No.	Function	in	ESPRESSO	DISJOINT	Minimal *CGRM*	Quasi-Minimal *CGRM*
1.	5xp1	7	65	71	61	71
2.	9sym	9	85	145	173	173
3.	bw	5	22	26	22	22
4.	con1	7	9	10	17	21
5.	duke2	22	87	103		255
6.	f51m	8	76	78	56	77
7.	rd53	5	31	31	20	20
8.	rd73	5	127	127	63	63
9.	rd84	8	255	255	107	107
10.	sao2	10	58	93	100	100
11.	misex1	8	12	14	20	20
12.	misex2	25	28	28		87

Table 3.7. Comparison of the Minimization Programs Against Benchmark Functions

The comparison of the timing performance of the heuristic and exact CGRMIN are shown in Table 3.8.

No.	Function	Minimal CGRM	Quasi-Minimal CGRM
1.	5xpl	2.8u 0.3s	0.1u 0.0s
2.	9sym	252.8u 13.6s	3.2u 0.1s
3.	bw	0.2u 0.0s	0.0u 0.0s
4.	con1	0.1u 0.0s	0.0u 0.0s
5.	duke2		0.5u 0.1s
6.	f5lm	9.3u 0.8s	0.2u 0.0s
7.	rd53	0.2u 0.0s	0.0u 0.0s
8.	rd73	28.4u 0.8s	
9.	rd84	187.0u 8.0s	6.2u 0.2s
10.	sao2	29.1u 1.4s	0.5u 0.0s
11.	misex1	0.2u 0.0s	0.0u 0.0s
12.	misex2		0.1u 0.0s

Table 3.8. Comparison of the Timings of Minimization Programs Against Benchmark Functions

The times given in the table are in seconds and the user and system CPU times are designated with "u" and "s" respectively. These results were obtained by running the programs on a Sparc 10. It can be observed that while the heuristic program provides exact solutions in may instances, the time it requires is substantially less than the exhaustive method.

The heuristic and exhaustive CGRMIM were compared for 112 single output functions generated from the MCNC benchmarks to evaluate the quality of the heuristics. These results are presented in Table 3.9.

Number Functions	Exact	Difference of one Term	Difference of Two Terms	Difference of Ten or More Terms
112	66	8	15	6

Table 3.9 Difference of Terms for Exact and Heuristic CGRM

As shown in Table 3.9, for majority of the functions, 66 out of 112 to be precise, the heuristic program found the exact minimal solutions. There were only 6 functions that differed by more than 10 terms. Further possible improvements to the program would include addition of backtracking and additional searching. Classification of

functions can also be used to determine useful heuristics for special types of functions.

As indicated in the timing performance, the required time for exact programs for functions that have limited number of undetermined polarity literals is very low. This can be used in combining the heuristic and minimal features in one program. From the results in Table 3.8, it can be inferred that for functions that have up to 12 undetermined polarity literals, i.e. do not have any of the features described in section 5.3, the timing is not very significant. This can be used to find minimal solution for the functions with up to 12 undetermined polarity literals and then use the heuristics for functions with more undetermined polarity literals for quasi-minimal solution with faster speed.

3.6. Summary

In this chapter, this author introduced several improved techniques for the realization as well as identification of the minimal representation of Boolean functions in fixed polarity AND/XOR forms. After early introduction of the CCM methodology, all the theorems, definitions, and algorithms have been developed solely by this author. The techniques developed rely on the characteristics of the functions as represented as an array of disjoint cubes. The programs developed based on these results, show great versatility and in great majority of cases, major improvement in timing performance.

Realization part of the *CGRM* forms shows major improvement over the previous methods. Further improvements in this aspect can still result in faster minimization schemes. Incorporation of the versatility of the programs over brute force has shown to be of positive consequence. The Quasi-minimal program shows in many cases to be the actual minimal solution. The future improvements to the heuristics can still result in solutions closer to the minimal for more functions.

There is room for further research in this area. These can be divided into improvement of realization of *CGRM*, improvement of the implementation of the programs, and improvement of the heuristics in the Quasi-Minimal program. Other research includes

extension of the programs into incompletely specified functions. Multi-valued functions can also be investigated. Minimization of incompletely-specified functions in *CGRM* form has been studied in the literature [56]. This topic is not presented here as it will be basically an extension of the methods for future study.

A further note should be made on the techniques developed following the one presented here which incorporate decision diagrams. It has been conjectured that for many examples, the decision diagrams can provide more compact representations than the cubes. It is the opinion of this author that the techniques presented in this chapter have provided major advantage over all the previously generated methods and can still be incorporated into the methods using decision diagrams.

Chapter 4

Application of CGRMIN in Minimal Realization of Boolean Functions in Generalized AND/XOR Forms

4.1. Introduction

The Generalized AND/XOR canonical forms, referred here to as *CRMP* forms, were shown in Chapter 2 to be a large class of AND/XOR canonical forms. Due to the large number of canonical forms embedded in this class, finding a minimal realization of Boolean functions in *CRMP* forms allows for a much reduced realization than the fixed polarity forms. These forms, similar to fixed polarity forms have high testability properties which are described in chapter 7. They are also known to be on average smaller than the SOP realizations of functions. In this chapter some of the properties of these forms are reviewed without giving any proofs. Furthermore an application of the program *CGRMIN* from the previous chapter for identification of a minimal *CRMP* form is presented.

The basic properties of the *CRMP* forms are due to Csanky [33, 34]. In these studies, the concepts of prime terms and nonexisting terms have been introduced. Prime terms are independent of the realization of the function and exist in all forms. Nonexisting terms, on the other hand, are the terms which will not exist in any *CRMP* form. This allows to start from any *CRMP* form, and in our case the fixed polarity forms, and based on the above concepts, devise a minimization scheme. Furthermore, in the above studies the lower and upper bounds for the number of terms in *CRMPs* were presented. It was

---

This chapter is based on the technique published in M. A. Perkowski, L. Csanky, A. Sarabi, and I. Schäfer, proceedings of the IEEE International Conference on Computer Design, Cambridge, MA, October 1992.

shown that the upper bound for *CRMP* forms are 3/4 of those for SOP forms.

Based on the above properties, a program to identify the minimal *CRMP* forms has been devised by Perkowski and Schäfer [101, 34]. In this program, CANNES, the above concepts have been used to decompose a representation in a *CRMP* form into component *CGRM* forms and minimize the components in these *CGRM* forms repeatedly in order to identify a minimal *CRMP* realization of the function. The program CGRMIN, described in the previous chapter, has been used in the newest version of CANNES to speed up this approach.

In this chapter, the basic properties of *CRMP* forms are reviewed without going into details and the basic approach of CANNES is presented. The application of CANNES on MCNC benchmark functions is also given.

4.2. Canonical Restricted Mixed Polarity Forms

Definition 4.1. The *Boolean difference* of function f with respect to variable x_i is denoted by f_{x_i} and defined as

$$f_{x_i} = f(x_1, ..., x_i, ..., x_n) \oplus f(x_1, ..., \overline{x_i}, ..., x_n).$$

Definition 4.2. The *Boolean difference* of function f with respect to term $t = \dot{x}_i \dot{x}_j \cdots \dot{x}_k$ is denoted by f_t and defined as:

$$f_t = (..(f_{\dot{x}_i})_{\dot{x}_j} \cdots)_{\dot{x}_k}.$$

For further details on the Boolean difference see [1].

Definition 4.3. Let t be a term. The *term set* $S(t)$ of t is $S(t) = \{x_i \mid \dot{x}_i$ appears in $t\}$.

Definition 4.4. Term t is a *prime term* with respect to function f iff $f_t \equiv 1$, where $\equiv$ stands for identical equality.

Theorem 4.1. Term t is a prime term with respect to function f iff in any *CRMP*

form of f there exists exactly one term t' such that $S(t) = S(t')$ and there exists no term t'' such that $S(t) \subset S(t'')$.

Some of the implications of this theorem are the following properties.

1. The minimal *CRMP* (and of course fixed polarity) forms of f will also have one such term t' that $S(t) = S(t')$ and there is no term t'' such that $S(t) \subset S(t'')$.
2. For all existing terms $\underline{t}$ of a *CRMP* form of f there is a prime term t of f such that $S(\underline{t}) \subseteq S(t)$.
3. Every function in a *CRMP* form has at least one prime term.

Definition 4.5. Term t is a *nonexisting term* with respect to function f iff $f_t \equiv 0$.

Theorem 4.2. Term t is a nonexisting term with respect to function f iff in any *CRMP* form of f there is no term t' such that $S(t) \subseteq S(t')$.

Definition 4.3. A function $f(x_1, ..., x_n)$ is *odd* iff $f_{x_1 \cdots x_n}$ is identically 1. A function $f(x_1, ..., x_n)$ is *even* iff $f_{x_1 \cdots x_n}$ is identically 0.

It can be proven that the following properties are true.

4. Every even function f in a *CRMP* form has at least one nonexisting term.
5. If function f is odd, it has no nonexisting terms in a *CRMP* form.

Example 4.1. $f_1 = x_1x_2x_3$ and $f_2 = x_1x_2x_3 \oplus x_1x_2$ are odd functions. Function $f_3 = x_1x_2 \oplus x_2x_3 \oplus x_2x_3$ is an even function and $x_1x_2x_3$ is an nonexisting term of this function.

It is possible to devise certain properties for the term-wise upper and lower bounds for the *CRMP* form. It is known that any Boolean function f of n variables can be described by at most 2^{n-1} terms in disjunctive or conjunctive form. Moreover, the value

2^{n-1} is the least upper bound, since there are functions whose description needs exactly 2^{n-1} terms.

The *CRMP* form can be proven to be more economical in the sense that it has an upper bound with lower number of terms.

Theorem 4.4. Any Boolean function of n variables ($n \geq 3$) can be described by at most $\frac{3}{4}(2^{n-1})$ terms in a *CRMP* form.

Furthermore, for any function f given in positive AND/XOR form, there is an algorithm to find this *CRMP* form which takes $\frac{3}{4}(2^{n-1})$ steps.

The same upper bound was obtained for ESOP forms in [120], which shows that for difficult functions, the *CRMP* forms are as good as *ESOP*.

Definition 4.5. A Term t_1 *is a proper subcombination* of term t_2 iff $S(t_1) \subset S(t_2)$.

Example 4.2. In function $f = x_1x_2x_3\overline{x_4} \oplus \overline{x_1}x_3$, the term $\overline{x_1}x_3$ is a proper subcombination of the term $x_1x_2x_3\overline{x_4}$. □

Theorem 4.6. All terms of a Boolean function f of n variables given in a *CRMP* form which are not subcombinations of other terms in the same *CRMP* form will exist in any *CRMP* form of f (possibly they will exist with another polarity of variables).

Corollary 4.1. The prime terms will exist in the minimal *CRMP* form too.

Corollary 4.2. All existing terms in any *CRMP* form of a Boolean function f of n variables are subcombinations of prime terms.

Corollary 4.3. For a given Boolean function f of n variables, the prime terms are entirely determined by f and they do not depend on the *CRMP* form from which they are determined.

Corollary 4.4. There exists a Boolean function of n variables for which the

minimal *CRMP* form contains as many as

$$\begin{bmatrix} n \\ \frac{n}{2} \end{bmatrix}$$

terms if n is even, and

$$\begin{bmatrix} n \\ \frac{n-1}{2} \end{bmatrix} = \begin{bmatrix} n \\ \frac{n+1}{2} \end{bmatrix}$$

terms if n is odd.

This proves that the conjectured upper bound on the number of terms in a term-wise minimal *CRMP* form cannot be further decreased.

Theorem 4.7. If term t of a Boolean function f of n variables, given in a *CRMP* form, does not exist and is not a subcombination of the prime terms of f, then in no *CRMP* form of f can there be a term t' such that $S(t) = S(t')$.

Example 4.3. There are some functions given below with the prime terms underlined and the nonexisting terms listed.

(*i*) $1 \oplus x_1 \oplus x_2 \oplus \underline{x_1x_2x_3}$

There are no nonexisting terms.

(*ii*) $1 \oplus x_1 \oplus \underline{x_1x_2} \oplus \underline{x_1x_3} \oplus \underline{x_2x_3}$

Nonexisting terms: $x_1x_2x_3$

(*iii*) $1 \oplus \underline{x_1} \oplus x_2 \oplus \underline{x_2x_3}$

Nonexisting terms: x_1x_2, x_1x_3, $x_1x_2x_3$

(*iv*) $x_1 \oplus x_2 \oplus \underline{x_3} \oplus x_4 \oplus \underline{x_1x_2x_4}$

Nonexisting terms: $x_1x_3, x_2x_3, x_3x_4, x_1x_2x_3, x_1x_3x_4$, $x_2x_3x_4$, $x_1x_2x_3x_4$

(*v*) $x_1 \oplus x_2x_3 \oplus \underline{x_1x_2x_3x_4}$

Nonexisting terms: none.

(*vi*) $\underline{x_1} \oplus \underline{x_2x_3} \oplus \underline{x_2x_4} \oplus \underline{x_3x_4}$

Nonexisting terms: $x_1x_2, x_1x_3, x_1x_4, x_1x_2x_3, x_1x_2x_4, x_1x_3x_4, x_2x_3x_4, x_1x_2x_3x_4$. □

It can be observed that if there exist only prime terms in the expression, then this expression is a both term-wise and literal-wise minimal *CRMP* form. If one can merge every other terms into the prime terms so that the resultant form has the same number of terms as the prime terms, the resultant form is also an exact minimum one.

Example 4.4. For case (*ii*) in previous example, $f = 1 \oplus x_1 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3$ after merging x_1 and x_1x_2 becomes $1 \oplus x_1\overline{x_2} \oplus x_1x_3 \oplus x_2x_3$. Now one can see that variable x_2 occurs in both forms. Also adding x_2 to the form would permit to merge it with 1 in order to create $\overline{x_2}$, which in turn could be merged with $x_1\overline{x_2}$. Therefore, the above form becomes: $1 \oplus x_1\overline{x_2} \oplus x_1x_3 \oplus x_2x_3 \oplus x_2 \oplus x_2 = x_1\overline{x_2} \oplus x_1x_3 \oplus x_2\overline{x_3} \oplus \overline{x_2} = \overline{x_1}\,\overline{x_2} \oplus x_1x_3 \oplus x_2\overline{x_3}$. All terms are prime, so the solution is an exact minimum *CRMP* form. □

4.3. Algorithm for Quasi-minimal *CRMP* Synthesis

Based on the above theorems and the properties indicated, a depth-first search algorithm has been devised. This algorithm, called CANNES (CANonic Nor Exor Synthesizer) [34], is based on the fact that all prime terms are entirely determined from the Boolean function, f, and they do not depend on any *CRMP* form and that all existing terms in a *CRMP* form of f are subcombinations of prime terms. In this sense, CANNES is an algorithm which generates the minimal *CGRM* form for the prime terms and their subcombinations.

The algorithm is shown in Figure 4.1.

Calculation of minimum CRMP form from a CGRM

Begin

$i := 1,\ f_1 := f;$

while $(f_{i+1}) \neq 0$ **do**

begin

decompose $f_i = f_{i+1} \oplus f_{ip}$, where f_{ip}
is the modulo 2 sum of prime terms with respect to f_i such that:

(*i*) Terms t and t' occurring in f_{ip} imply that $S(t') \not\subset S(t)$,

(*ii*) $(f_i)_t \equiv 1$ implies $(f_{ip})_t \equiv 1$,

(*iii*) Set $\{S(t) \mid (f_{i+1})_t \equiv 1\}$ has the minimal cardinality;

$i := i + 1.$

end;

Print *solution:* $f = f_{1p} \oplus f_{2p} \oplus \cdots \oplus f_{ip}$;

End.

Figure 4.1. The Calculation of a minimum *CRMP* from a *CGRM*

The above algorithm can be implemented in many different forms. An implementation by Ingo Schäfer, called CANNES (CANonic Nor Exor Synthesizer) is based on a depth-first tree searching algorithm that makes use of the above theory and particularly the properties stated in Corollaries 4.2 and 4.3. Those Corollaries stated that all prime terms are entirely determined by the Boolean function f and do not depend on the *CRMP* form and that all existing terms in a *CRMP* form of f are subcombinations of prime terms. Thus, CANNES is based on an algorithm which generates the minimal *CGRM* form for the prime terms and their subcombinations.

The simplified recursive minimization procedure of CANNES is shown in Fiugre 4.2.

CANNES-2 uses the heuristic *CGRM* minimizer, *CGRMIN* from the previous chapter to find minimal *CGRM* forms for subsets of variables. For the exact *CGRM* minimizer, while the method requires searching all polarities of a *CGRM*, and even several times during the *CRMP* minimization, it is usually done on a subfunction of the initial function. Only in the worst case of a single prime term, the polarities of all input variables are searched. Concluding, with an amount of search that is comparable to that of a *CGRM*, it is possible to find a form that is not worse than the *CGRM*.

Notation:

List - complete list of terms describing the function.
NewList - starting **List** of next recursion.
prime term - prime term of the **List**.
subset - subset of terms for a *prime term*.
minsubset - minimal CGRM form of the *subset*.

```
minimize( List )
{
        for each prime term of the List;
        {
            // calculate the subset for the prime term
            subset := subset_of (prime term);

            // calculate the minimal CGRM form of the subset
            minsubset := minimal_CGRM (subset);

            // compare number of terms
            if ( |minsubset| < |subset| )
            {
              NewList := List;
              replace subset in NewList by minsubset;
              minimize( NewList );
        }
    }
}
```

Figure 4.2 The CANNES Algorithm

4.4. Experimental Results

CANNES-2 was tested on 100 single output functions generated from the MCNC benchmarks. Table 4.1 shows the number of terms for ESPRESSO, CANNES-2, and EXORCISM [142], an *ESOP* minimizer, for some of these functions. In this table, n stands for the number of variables in the functions.

For the functions tested, the compactness of AND/XOR forms is confirmed. While for the 100 functions overall, ESPRESSO resulted in 1001 terms, CANNES-2 gave 845 and EXORCISM 652. For 40 percent of the functions, CANNES-2 gave better results than ESPRESSO while for 30 percent, ESPRESSO gave fewer terms. For the rest, they both gave the same number of terms. Some of the examples of these cases are shown in

Table 4.1. Moreover, for all small functions that can be verified (such as all single output functions of three and many functions of four variables), the algorithm produced the exact *CRMP* solutions. Whether the algorithm always gives the exact solution needs to be studied further.

Name	*n*	ESPRESSO	CANNES-2	EXORCISM
5xp1l	7	7	9	6
9sym	9	85	131	51
majority	5	5	6	5
bw7	5	6	5	5
con12	7	5	4	4
duke8	22	5	4	5
f51m4	8	10	6	5
rd532	5	16	5	5
rd732	5	64	7	7
rd842	8	128	8	8
mis70	5	6	7	5
vg28	25	5	9	7
z42	7	28	9	9

Table 4.1 Two Level AND/OR Compared to Two Level *CRMP* and *ESOP*

As seen in the table (and in many other benchmark results) there exist real-life functions for which *CRMP* is more compact and there are other where SOP is more compact. *CRMP* forms are however always much better testable.

Unfortunately, although it is possible to create whole classes of functions for which the proposed approach will lead to minimum solutions without much search, the MCNC benchmark examples show that on real-life examples the number of prime terms is much smaller than the number of terms in the minimal solution. This results in the decompositions given in the algorithm to occur rarely, and the cost evaluations to be too pessimistic. However, for some functions it is always possible to confirm an exact minima.

4.5. Summary

In this chapter, the application of the fixed polarity minimization technique of CGRMIN was introduced for the minimization of functions in Generalized AND/XOR canonical forms. The concepts in [33] [34] were used to decompose a *CRMP* into

component *CGRM* forms. Then the minimization technique from the previous chapter was utilized as a scheme to identify a minimal representation in *CRMP* form.

The advantage of *CRMP* forms is that they provide a more compact representation of the functions than *SOP*s in general. In addition, it is easy to devise testing schemes for these forms which do not exist for *SOP*s. A new testability scheme is presented in chapter 7.

Chapter 5

Complex Maitra Logic Array Approach to CA-Type FPGA Synthesis

5.1. Introduction

Complex Maitra Logic Array (CMLA) approach to CA-Type FPGA Synthesis is a rectangular array approach which combines the logic synthesis and physical design stages together. By combining the two, and maintaing the regularity of the architecture in the synthesis part, it is possible to devise efficient mappings for these type of FPGAs. This characteristic is especially important for the limited and local connections among the cells. The proposed rectangular array utilizes two-input AND, OR and XOR cells with local connectivity and limited horizontal and vertical buses. This approach not only takes the placement and routing directly into synthesis stage, but also uses a combination of AND, OR, and XOR cells resulting in more efficient synthesis than traditional Boolean AND/OR logic.

CMLA is based on the Maitra terms, originating from Maitra cascades [78]. A Maitra term is the generalization of the product term and is a sequence (row) of AND, OR and XOR operators with corresponding literals. In the CMLA, the input variables of the Boolean function are in vertical buses. The Maitra terms are realized horizontally and their outputs are given to horizontal buses. The plane where the Maitra terms are realized comprises the complex (input) plane of the CMLA. The terms are then XORed (or ORed) together in the collecting (output) plane. Complex Maitra Logic Array is a

---

This chapter is based on the techniques published in A. Sarabi, N. Song, M. Chrzanowska-Jeske, and M. A. Perkowski, proceedings of the 31st ACM/IEEE Design Automation Conference, San Diego, CA, June 1994.

powerful generalization of PLAs and XPLAs. The CMLA concept is shown in Figure 5.1.

The comprehensive approach to the logic and layout synthesis for CA-Type FPGAs includes two stages:

1. **Logic optimization** which takes the geometry and layout constraints into account to create a CMLA in which every output function is an OR or XOR of Maitra terms.
2. **Technology-folding** which maps CMLA representation of the function to the target architecture, such that the area of the layout is minimized.

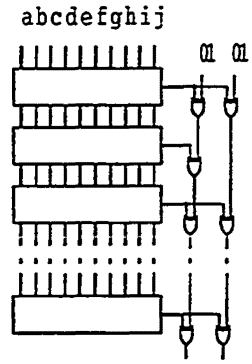

Figure 5.1. An Example of a Complex Maitra Logic Array

The fixed polarity and Generalized AND/XOR canonical forms and AND/XOR/XOR UXFs presented in chapter 2 can readily be used for the Logic optimization stage. In the AND/OR/XOR canonical forms presented, by each application of operations α, $\overline{\alpha}$, ρ, $\overline{\rho}$, σ, and π, a new literal is either ANDed or ORed with the existing terms. Therefore, the cascade realizability is guaranteed during the generation of the uxf-terms. As these AND/OR/XOR canonical forms are much more general than the simple AND/XOR forms, it is obvious that the same result holds for all AND/XOR realizations.

A faster multi-level algebraic method has been introduced by Song [143, 144]. In this method, called the *restricted factorization*, basic factorization techniques are applied on a Two-level SOP or ESOP term to produce a multi-level AND/OR/XOR term while maintaining the cascade realizability restrictions. While factorization techniques in general produce local minimal solutions, they have the advantage of being fast techniques. The basic concepts of restricted factorization theory are reviewed here and an algorithm to generate a minimal factorized form is introduced in section 5.3.

The folding stage is a "technology independent" approach that is always used after the optimization stage. In this stage, the basic architectural characteristics of each CA-Type FPGA are taken into account and the complex plane is further compacted. The main feature is to take the advantage of the local buses in the architectures to realize more than one Maitra term in a cascade inside the complex plane. The application of this folding technique is shown for the specific case of ATMEL6000 in section 5.4.

In section 5.2, the concepts of Maitra and Complex Maitra terms are defined. section 5.3 provides a review of the restricted factorization theory and introduces a logic optimization algorithm based on this theory. The solution method to the technology folding problem is discussed in section 5.4.

The concepts that are new in this chapter are the generalizations of the Maitra terms to include more than just the forward terms, the algorithm to generate the restricted factorized form of the functions as well as its implementation, and the concept of technology folding.

5.2. Maitra terms and Complex terms

Definition 5.1. A *forward Maitra* term is defined recursively as follows:

1. a literal is a forward Maitra term.
2. if M is a forward Maitra term, then $M \cdot a$, $M \cdot \bar{a}$, $M \oplus a$, $M \oplus \bar{a}$, $M + a$,

and $M + \bar{a}$ are also forward Maitra terms if no literal or its complement appears in the string more than once.

Example 5.1. Each of the following expressions represents a forward Maitra term: $(a\,\bar{b})+c, (a+b)c, (a \oplus b)+\bar{c}, ((c\,\bar{b})+a)\oplus d$. □

Definition 5.2. A *reverse Maitra* term is defined recursively as follows:

1. a literal is a reverse Maitra term.
2. if M is a reverse Maitra term, then $a \cdot M$, $\bar{a} \cdot M$, $a \oplus M$, $\bar{a} \oplus M$, $a + M$, and $\bar{a} + M$ are also reverse Maitra terms if no literal appears in the string more than once.

Example 5.2. Each of the following expressions represents a reverse Maitra term: $c+(a\,\bar{b}), c\,(a+b)$. □

Forward and reverse Maitra terms are called *simple Maitra* terms.

Definition 5.3. A *bidirectional Maitra* term has the form:

$$M_1 \alpha M_2$$

where α is a Boolean function of two arguments, M_1 is a forward Maitra, and M_2 is a reverse Maitra term, such that M_1 and M_2 have different sets of variables and do not exhaust together all input variables of the function.

Example 5.3. $M_1 \oplus M_2 = (ab) + c \oplus e(f + g)$ is a bidirectional term of function $f(a,b,c,d,e,f,g)$ since M_1 is a forward term on variables a, b, and c; M_2 is a reverse Maitra term on variables e, f, and g; and sets $\{a,b,c\}$ and $\{e,f,g\}$ are non-overlapping. Variable d is not used in any of these sets. □

Definition 5.4. A *complex Maitra* term *(complex term)* is a forward Maitra term, a reverse Maitra term, or a bidirectional Maitra term.

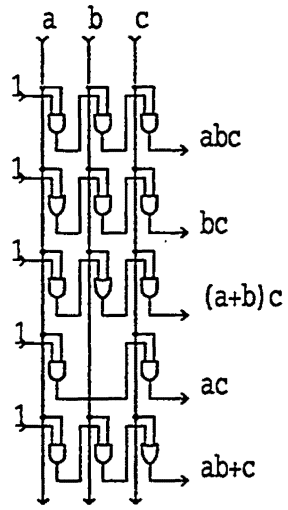

Figure 5.2. An Example of a Complex_Plane of a CMLA

Example 5.4. The expression $((a\,b)+\bar{b})c$ is not a Maitra term because the literal $\dot{b}$ appears twice. Similarly, $a+(b\,\bar{c})+d$ is not a forward Maitra term because it cannot be generated from the forward Maitra term definition (analyzing the expression from right to left, $a+(b\,\bar{c})$ is not a forward Maitra term). However, if the order of variables is changed to b, c, a, d, then $(b\,\bar{c})+a+d$ becomes a forward Maitra term. □

Example 5.5 shows that whether a given logic expression is a Maitra term or not, depends on the order of variables in this expression. Some expressions which are not Maitra terms can become Maitra terms by changing their order of variables. For every order of input variables, a Boolean function can be decomposed to an OR or XOR of Maitra terms. This is always possible, since the AND terms (used in SOPs and ESOPs) are particular cases of the Maitra terms. The example of CMLA is shown in Figure 5.2.

5.3. Restricted Factorization Theory

The Restricted Factorization Theory is based on the identification of complex term combinabilities and devising a method for their combination to result in a minimal number of terms. In this section, the conditions for the term combinability are reviewed

and an algorithm for the minimal realization of Boolean functions based on this factored forms is introduced.

5.3.1. Term Combinability †

The notion of identification of the complex terms is based on the distance and difference of the cubes representing these terms. The conditions for generation of the complex terms are given in the following:

Definition 5.5. Let T_i and T_j be two product terms. The *distance* of the two terms, $distance(T_i, T_j) = d'$, is the number of variables for which the corresponding literals of these terms have different polarities.

Definition 5.6. Let T_i and T_j be two product terms. The *difference* of two terms, $difference(T_i, T_j) = d$, is the number of variables for which the corresponding literals of these terms have different values.

Example 5.6. Let $T_1 = a\bar{e}$, $T_2 = \bar{b}ce$, and $T_3 = ab\bar{c}$. The difference of T_1 and T_2 is 4, because all literals are different. The distance of T_1 and T_2 is 1, because the literals of variable e have different polarities. The difference of T_2 and T_3 is also 4. Their distance however is 2, as b and c occur in opposite polarities in each. □

Definition 5.7. Two product terms T_1 and T_2 are *directly combinable*, if these two product terms are in one of the following forms:

$$\begin{aligned} T_1 &= \dot{x}_1\dot{x}_2 \cdots \dot{x}_{i-1}\ \ \dot{x}_{i+1} \cdots \dot{x}_n \\ T_2 &= \qquad\quad \dot{y}_i\dot{y}_{i+1} \cdots \dot{y}_n \\ &\qquad \dot{x}_j = \dot{y}_j \text{ for } j \geq i+1 \end{aligned} \tag{5.1}$$

$$\begin{aligned} T_1 &= \dot{x}_1\dot{x}_2 \cdots \dot{x}_{i-1}\dot{x}_i\ \dot{x}_{i+1} \cdots \dot{x}_n \\ T_2 &= \qquad\quad\ \ \dot{y}_{i+1} \cdots \dot{y}_n \\ &\qquad \dot{x}_j = \dot{y}_j \text{ for } j \geq i+1 \end{aligned} \tag{5.2}$$

† This section is based on the paper: N. Song and M. A. Perkowski, A Method for Logic Mapping for Fine Grain FPGAs, IWLS'93, Taho, CA, May 1993.

Example 5.7. $a\bar{b}de \oplus cde = (a\bar{b} \oplus c)de$. □

In Equation (5.1), the two product terms can be combined to

$$(\dot{x}_1\dot{x}_2 \cdots \dot{x}_{i-1} \oplus \dot{x}_i)\dot{x}_{i+1} \cdots \dot{x}_n$$

In Equation (5.2), the two product terms can be combined to

$$(\dot{x}_1\dot{x}_2 \cdots \dot{x}_{i-1}\dot{x}_i \oplus 1)\dot{x}_{i+1} \cdots \dot{x}_n =$$
$$(\vec{x}_1+\vec{x}_2+ \cdots +\vec{x}_{i-1}+\vec{x}_i)\dot{x}_{i+1} \cdots \dot{x}_n$$

here $\vec{x}_i$ indicates the negation of $\dot{x}_i$.

Example 5.8. $a\bar{b}cde \oplus de = (a\bar{b}c \oplus 1)de = (\bar{a}+b+\bar{c})de$, the two product terms are directly combinable. □

For convenience, two given product terms in the forms $T_1 = \hat{x}_1\hat{x}_2 \cdots \hat{x}_n$ and $T_2 = \hat{x}_1\hat{x}_2 \cdots \hat{x}_n$ are assumed. Without loss of generality, it is assumed that the pairs of literals which have different values appear at the left side in the terms.

Case when difference$(T_1, T_2) = 0$.

Difference = 0 means these two terms are identical. In case of an ESOP, since $A \oplus A = 0$, these two product terms can be removed. In case of a SOP, since $A + A = A$, one of the terms can be removed.

Case when difference$(T_1, T_2) = 1$.

(1) If distance$(T_1, T_2) = 0$, then $\hat{x}_1$ appears only in one term. Since $1 \oplus a = \bar{a}$, these two product terms are directly combinable.

(2) If distance$(T_1, T_2) = 1$, then $\hat{x}_1$ appears in both terms, but in different polarities. Since $a \oplus \bar{a} = 1$, these two product terms are also directly combinable.

Theorem 5.1 If the difference of two product terms is greater than 1, then these two product terms are directly combinable if and only if their distance is 0 and from all the literals that do not appear concurrently in both terms only one literal can appear in a term.

Definition 5.8. Two product terms are referred as *combinable* either when these two product terms are directly combinable or if they can become directly combinable by reshaping them.

Theorem 5.2. If difference$(T_1, T_2) \leq 2$, terms T_1 and T_2 are combinable.

Theorem 5.3. If difference$(T_1, T_2) = 3$, terms T_1 and T_2 are combinable if and only if distance$(T_1, T_2) < 2$, and if distance$(T_1, T_2) = 1$, both $\hat{x_1}$ and $\hat{x_2}$ occur in one term and are missing in the other. $\hat{x_1}$ and $\hat{x_2}$ are the two "other" literals which cause the difference to be 3.

It can further be shown that if difference$(T_1, T_2) > 3$, two product terms can be combined if and only if

(1) distance$((T_1, T_2) = 0$, and the two terms can be arranged to the form of Equation 5.2. Or

(2) distance$((T_1, T_2) = 1$, and the two terms can be arranged to the form of Equation 5.1.

5.3.2. Realization of functions in Minimized Restricted Factorized Form

In order to realize a function in a minimal factorized form, it is important to identify the combinable terms and the ordering of the variables which would result in most combinabilities. Here, a combinability graph is constructed and based on the maximum cliques in the graph, the ordering is chosen. Combinability graph is a graph, $G(V, E)$, where the vertices are the product terms and the edges indicate whether two terms are combinable or not. The algorithm to generate the complex terms is presented in Figure 5.3:

1. *For each pair of product terms T_i and T_j, if the terms are combinable, record all possible variable orderings for the pair;*
2. *Build the adjacency matrix for the combinability graph;*
3. *Create a priority list of complex terms in the decreasing order of the number of adjacents and adjacents of their adjacents in the combinability graph.*
4. *Choose the ordering of the input variables as the order of the variables in a maximum clique which does not violate all possible combinabilities in the clique.*
5. *Start with the product terms with the least priority and generate the possible complex terms with the chosen order of input variables. Generate the new terms which are not common among the corresponding output functions.*
6. *Repeat the procedure of generating the complex terms for the remaining terms until no complex terms can be generated.*

Figure 5.3. The Restricted Factorization Algorithm

As indicated, the ordering plays an important role in the number of terms that can be combined. For this purpose, the maximum cliques in the graph are used to identify the most number of terms that can be combined - having the same order of variables. A further restriction on the clique is put for the terms to hold in order to make them a candidate. That is any two of them would not have a conflicting literal position requirement. The order of all the variables is then chosen based on an ordering that fits such a maximum clique.

The terms are then sorted in the decreasing order according to the number of combinables. When two terms have an equal number of combinables, they are sorted according to the total number of combinables of their combinables. That is the number of combinables for each of their combinables are added up and the one with a larger number is given a higher priority in the list. The terms with low number of combinabilities are then compared with the higher ones in the list and if possible, are combined.

It has to be noted that for the multi-output functions, in the process of combining two terms, the new complex term will only belong to the common outputs and all outputs that are not common, still need to have their corresponding product terms present in the list. This then results at times in more terms being generated. Overall, however, for

many benchmark examples, it was observed that the method results in less number of terms. The procedure of combining terms is continued until there are no possible combinabilities.

The method has been tested and the results for several benchmark functions are presented in Table 5.1

benchmark	#in	#out	ESOP	CTERM
5xp1	7	10	32	33
cadr4	8	5	31	30
clip	9	5	63	57
clog8	8	8	87	84
cmlp4	8	8	61	54
cnrm4	8	5	69	52
cu	14	11	16	15
f51m	8	8	31	30
inc	7	9	26	26
mlp3	6	6	18	17
rd53	5	3	14	13
rd73	4	3	38	36
sao2	10	4	28	26
t481	16	1	23	18
vg2	25	18	184	179
Total			721	670

Table 5.1. Complex Terms for Benchmark Examples

The above method can be improved in many directions. The choosing of the variable ordering can be investigated for other possibilities. The combination of terms can be improved by introducing backtracking techniques. Reshaping of the terms can also be performed to restart the cubes in different distances and differences. Another improvement is to decompose the terms into clusters which can use different variable orderings, possibly resulting in further reductions in size.

5.4. Technology Folding

Once an optimized set of complex terms has been identified - either through factorization, UXF, or Two-level representation - folding techniques are used to even more economically utilize the FPGA cells. To minimize the area, a proper matching of

complex terms is found such that the number of rows occupied by complex terms is minimized. These folding techniques depend on the specific architecture of the FPGA which could allow different compatible terms to be placed on the same row. The row folding technique will be shown for the case of ATMEL6000 in order to illuminate the details of the technique.

The possible gates that can be utilized in ATMEL6000 are an inverter, an AND, OR, XOR, NAND gate, a wire and their combinations [7]. The number of available inputs is limited to three and outputs to two. Only one input can be taken from the local bus, and the number of local busses is limited to four. Each cell has connections to four neighboring cells. Based on the above architecture limitation, the number of complex terms which can be folded into one row is six. The complex terms can be accessed from the left-most cell, the right-most cell, and the two local busses. As each cell has two immediate outputs, A and B, two terms can be accessed from the right- and left-most cells. It is assumed that the vertical local busses in the complex_plane carry both input variables and their negations for each column. The cell personalized to the OR gate uses only output of type A.

AB Parallelism

The following rules provide all the possibilities for placing two terms in one row of ATMEL using the A and B inputs and outputs going horizontally through the row.

In the following, certain terminologies which will be used to define various foldings based on AB parallelism will be given. As the input variables are assumed to follow certain order in the row, an increasing order from the left is used as a convention. In this way, the leftmost input variable will be in lowest order and the ones in the right will have increasing orders. Moreover, the instance of a literal in a uxf-term which has the lowest order will be referred to as the **initial** literal and will be denoted by C^i where C is some term. The literal with the highest order will be referred to as the **last** literal and will be

denoted by C^l. The set of all literals appearing in a term will be referred to as the **literal set** and will be denoted by SL_j, where j refers to that particular term.

As the terms can be products, sums, or their combinations, these will be separately designated below. A monoterm, which is a product of literals, will be referred to as a **product-line** and will be denoted by P. A term which is only comprised of a summation of literals will be referred to as a **sum-line** and will be denoted by S. A term which is comprised of both sums and products of literals will be referred to as a **sum-product-line** and will be denoted by C.

Designating the relations between the literals of the terms will also be useful. The **distance** between two literals is the difference between their indices. As an example the distance between h and i will be 1. This distance is denoted by d. A term which has a continuous literal set, i.e. for any literal besides the initial and final, there exist two literals of distance 1 with that literal in the literal set (for initial and final literals there exist one literal with this property), will be referred to as **continuous** and will be denoted by Q.

Using the above notations, it is now possible to define different AB-Parallel foldings. The two terms which can be placed on the same row via AB-Parallel folding will be denoted by $<C_i, C_j>$ where C_i and C_j are the two terms.

TYPE_1. $<P_1, P_2>$ such that $SL_1 \cup SL_2 = SL \in Q$ and $SL_1 \cap SL_2 = P_1{}^i$. e.g. $<P1, P2> = <a, abcde>; <ae, abcd>; <ade, abc>; <acde, ab>; \cdots$ where $SL = \{a, b, c, d, e\}$.

TYPE_2. $<S, P>$ such that $SL_P \in SL_S, SL_P \in Q$, and $S^l \in SL_P$. e.g. $<S, P> = <a+b+c+d, abcd>; \quad <a+b+c+d, bcd>; \quad <a+c+d, cd>; \quad <a+b+c+d, d>$.

TYPE_3. $<P, S>$ such that $SL_S \in SL_P$, and $SL_P \in Q$. e.g. $<S, P>$

$= <abcd, a+b+c+d>$; $<abcd, b+c+d>$; $<abcd, b+d>$; $<abcd, a+c>$.

TYPE_4. $<C, S>$ such that $d(C_S{}^l, C_P{}^i) \geq 1, C_S \in Q, C_P \in Q$ and $SL_S \in SL_{C_P}$. e.g. $<C, S> = <(a+b+c)defg, d+f>$.

TYPE_5. $<C_1, C_2>$ such that $SL_{S1} = SL_{S2}, SL_{P1} \cup SL_{P2} \in Q$, $SL_{P1} \cap SL_{P2} = SL_P{}^i$ and $d(SL_{S1}{}^l, SL_{P1}{}^i) \geq 2$. Where SL_{P1} denotes the product part of the first term. e.g. $<C_1, C_2> = <(a+b+c)d, (a+b+c)dfg>$.

TYPE_6. $<C_1, C_2>$ such that $SL_{S1} = SL_{S2}, SL_{P1} \cup SL_{P2} = \in Q$ and $C_S{}^l < (C_{P1}{}^i - 1)$ and $C_S{}^l < C_{P2}{}^i$. e.g. $<C1, C2> = <(a+b+c)ef, (a+b+c)gh>$.

Some of the terms, or for that matter AB-Parallel terms, can be put on a local bus. Again this is due to the particular architecture of the ATMEL and they are given below:

L Placeability

L Placeability refers to the terms that can be placed on one of the local buses based on the architecture constraints. The following cases are the possibilities for ATMEL6000:

1. An AB-Parallel set of terms of TYPE_1 is L-placeable.
2. The OR part of AB-Parallel set of terms of TYPE_2 is L-placeable.
3. A single primary input is L-placeable.

Now, with AB_parallelisms and L-placeabilities listed above, it is possible to investigate the possibilities of placing two or more uxf-terms on the same row of ATMEL6000. The terms that can be placed in one row will be referred to as being row compatible or **R-compatible**. These conditions as well as the corresponding cells that makes them possible are listed as follows:

Row Compatibility

- Two AB-Parallel terms are R-compatible. If the terms belong to more than one output, they have to be L-placeable in order to be R-compatible.
- An L-placeable set of TYPE_1 and another AB-Parallel set are R-compatible if the distance between the last literal of the first set and the first literal of the second set is at least two. e.g. $<ad, abc>$ and $<f+g+h+i, fghi>$ are R-Compatible.
- An L-placeable term of TYPE_2 and AB-Parallel pair of terms of TYPE_5 are R-compatible if the sum-lines of all three terms are identical.
  e.g. $<a+b+c>$ and $<(a+b+c)e, (a+b+c)efg>$ are R-compatible.
- An L-placeable term of TYPE_2 and a sum-line which includes the L-placeable term as lower order literals and its higher order literals start with a distance of two from last literal of the L-placeable term and are continuous are R-compatible. $<S_1, S_2>$ such that $SL_1 \in SL_2$, $d(SL_1{}^l, SL\ (S_2-S_1)^i) \geq 2\ S_2-S_1 \in Q$. "–" denotes the difference of the literal sets; i.e. all literals which are in the first set and not in the other.
  e.g. $<a+c+d>$ and $<a+c+d+f+g+h>$ are R-compatible.
- An L-placeable term of TYPE_2 and an AB-Parallel set of terms of TYPE_1 are R-compatible if the distance between the last literal of the sum-line and the first literal of the AB-Parallel set of terms is at least two.
  e.g. $<a+b+c>$ and $<e, eg>$ are R-compatible.
- An L-placeable term of TYPE_2 and a sum-product-line which includes the L-placeable term as summation part and its product literals start with a distance of two from the last literal of the summation part and are continuous are R-compatible. $<S, C>$ such that $C_S = S, d(SL_S{}^l, SL_P{}^i) \geq 2\ C_P \in Q$.
  e.g. $<a+c+d>$ and $<(a+c+d)fgh>$ are R-compatible.

- All R-compatibilities involving L-placeable terms of TYPE_2 are applicable to L-placeable terms of TYPE_3. The difference is that the sum_lines in terms of TYPE_2 are replaced by a single variable.

- Two L-placeable terms are R-compatible only if they belong to disjoint outputs.

One possible row folding is shown in Figure 5.4.

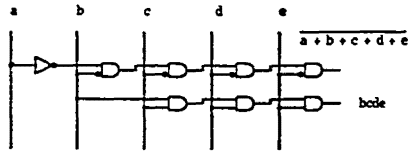

Figure 5.4. An Example of a Row Folding

Main Algorithm

Based on the compatibilities that are characteristic to the ATMEL architecture, the general approach to folding for CA-Type FPGAs is shown in Figure 5.5.

The compatibility graph refers to a graph where the nodes represent the terms and edges indicate whether two terms are R-compatible or not. Clique partition refers to identification of cliques of size 6 in the case of ATMEL6000.

As it can be observed, the method is general while the type of R-compatibilities and the size of the clique of interest varies with different architectures.

Implementation Considerations

In order to implement the above algorithm, certain input format for the sum-product terms is used. As the PLA format is devised for product terms alone, it is necessary to modify this format so that more complex terms can be represented. In this approach the following convention is used:

1. Sort the terms in a descending order according to the most occurrence of the literals and the number of outputs.
2. Identify the terms which are R-compatible and those which are do not have any compatibles. Assign the latter terms to a different row each and remove them from the list of the terms.
3. Construct the compatibility graph.
4. Find an optimum clique covering, with clique sizes smaller or equal to the maximum number of the terms which can be realized in the same row of a given architecture.
5. Sort the cliques according to decreasing size and increasing degree of the vertices.
6. Assign the cliques in the list to one row of the input plane at a time if none of the vertices in the clique has already been assigned. Delete the assigned vertices and edges from the graph.
7. Assign all remaining vertices which are connected with at least one edge in the graph to the different rows of the plane.
8. Assign all the remaining unconnected vertices to different rows.

Figure 5.5. The Technology Folding Algorithm

The input is a modified PLA format. In this format the number of inputs and outputs are given early on. Each term then has the following form:

input cube operation cube output cube

The input cube and output cube are the same as in the PLA format. The operation cube uses a 1 for product, a 0 for sum, and – for no operation. As an example, a term such as $(a+b)\bar{c}+e$ is represented as 110-1 101-0 1. The first cube designates the polarity of the literals. The second indicates that b and e are summed and c is ANDed. The 1 for the location of 1 is optional, however, it is always designated as a 1 for the initial literal in the term. The final cube 1 indicates that this is an uxf-term of a single-output function.

Each term has the following attributes:

The 1s of the term;
The DCs of the term;
The outputs;
The products;
The sums;
The row the term is assigned to;
The number of terms compatible with the term;
The number of L-placeable cubes with the term.

The output gives the rows with the compatible terms which are assigned to them. As an example, the input and output data for a given problem are shown in Figures 5.6 and 5.7. Here, the input terms are accordingly: ad, abc, $(f+g+h+i), fghi$, $(a+b+c+d)$, $(a+b+c+d+f+g)$, $(a+b+c)f, (a+b+c)efg$, $(a+b+c)$, $(a+b+c+d+e+f)$, $((a+b)d+e)f$, $ae, (a+c+d), cd$, $(ae+f)$, $(d+f), abcd, hi, h$.

```
# test input for Cellular-map
#
# .i 9  determines that there are 9 input variables
# .o 2  determines '1' ON-, '0' OFF- and '-' DC-cube for each of the two outputs
#       '0' in second row cubes designates product and '1' sum
# .e    determines the end of the file
.i 9
.o 3
.p 19
1--1-----    0--0-----    001
111------    000------    101
-----1111    -----1111    111
-----1111    -----0000    011
1111-----    1111-----    101
1111-11--    1111-11--    001
111--1---    111--0---    011
111-111--    111-000--    011
111------    111------    100
111111---    111111---    001
11-111---    11-010---    101
1---1----    0---0----    101
1-11-----    1-11-----    101
--11-----    --00-----    001
1---11---    1---01---    001
---1-1---    ----1-1--    011
1111-----    0000-----    101
-------11    -------00    010
------1-     ------0-     011
.e
```

Figure 5.6. The input to Cellular_map

As it can be observed from Figure 5.7, the 15 original uxf-terms are folded into 8 rows. The numbers refer to the numbers assigned to the terms in the input file. As an example, 1 2 3 4 refers to the first four input terms which can be placed in one row.

```
# output file of Cellular map
.i 9
.o 3
.c 10
1 2 3 4 - -
12 17 18 19 - -
5 14 - - - -
8 - 8 16 - -
- 10 - 9 - -
6
7
11
13
15
.e
```

Figure 5.7. The output of Cellular_map

There are many areas for improvement in the method presented. These possible improvements are listed below:

- The method can be further extended to column permutation and XOR-column folding. Hence, bidirectional Maitra terms can also be utilized. Presently, only row folding techniques have been addressed.

- The row assignment algorithm can be further improved based on graph-theoretic methods. This can be based on other methods of ordering and clique assignment.

- The notions of R-compatibility can be built into the synthesis stage for initial array. So, the two stages are not completely unrelated.

- Fast methods of identification of multiple R-compatibilities can be devised. In addition, other compatibilities including other serial compatibilities - placing several terms one after another in the same row - and checking of compatibles of degrees higher than 2 - three or more terms - could be investigated.

- The output format can be geared towards the appropriate manipulations. Currently, the results of the folding are shown as the terms that can be put in the same row. The output format can further be augmented with an indexed array of cells. Each cell having a 5 bit number designating the combinational function it is performing. In addition, each cell can have 4 two bit numbers designating A, B, and L_{in} and L_{out}.

- Once the synthesis method has been implemented, the number of cells can be compared with other mapping methods such as trees, compact irregular multi-level mapping, etc.

The same functions in Table 5.1 are again shown with folding included in Table 5.2.

benchmark	#in	#out	ESOP	CTERM	Folded
5xp1	7	10	32	33	26
cadr4	8	5	31	30	29
clip	9	5	63	57	52
clog8	8	8	87	84	69
cmlp4	8	8	61	54	48
cnrm4	8	5	69	52	46
cu	14	11	16	15	13
f51m	8	8	31	30	27
inc	7	9	26	26	23
mlp3	6	6	18	17	13
rd53	5	3	14	13	10
rd73	4	3	38	36	28
sao2	10	4	28	26	24
t481	16	1	23	18	15
vg2	25	18	184	179	179
Total			721	670	602

Table 5.2. The Effect of Folding

As indicated, the average improvement over Two-level ESOP realization on this set of benchmarks is 17%. For majority of the tested examples the improvement is significant and for some as high as 33%. It has to be emphasized that the reduced number of terms is multiplied by the number of input variables, as these would be the actual number of cells that will be reduced. Hence, any reduction of the number of terms

contributes a multiple of the input variables to the number of cells saved.

5.5. Summary

The comprehensive logic synthesis and physical design for CA-Type FPGAs was presented in this chapter. It was shown that through Complex Maitra Logic Arrays, the UXF, fixed polarity and Generalized AND/XOR forms of chapters 3 and 4, as well as the restricted factored forms can be directly mapped to these FPGAs. Hence, there is no need for placement and routing stage after synthesis. The folding technique was shown also to be advantageous in reducing the number of cells utilized. Both the concept of the folding of CMLA and the development of the particular folding technique presented are due to this author.

The restricted factorization technique and the technology folding were shown to provide a fast and in many times efficient techniques. While these techniques can be both much improved, the factorized forms were shown to be more compact than ESOPs. The folding also showed to be a definite advantage in these cases. These are the first results of this type ever published. There is room for much improvement for both techniques, however, they have been shown to be promising techniques. Again, the main advantage here is the combining of logic synthesis and physical design as well as the applicability of the technique to any general purpose function. Moreover, the logic utilized takes advantage of two input AND, OR, and XOR and results in more compaction than either AND/OR or AND/XOR.

Chapter 6

Minimal Multi-Level Realization of Boolean Functions Based on Kronecker Functional Decision Diagrams

6.1. Introduction

Directed Acyclic Graphs provide another important structure for CA-type FPGA synthesis. These structures similar to the rectangular have the advantage of the regularity of the structure which is of importance in CA-type FPGA synthesis. If reduction techniques are not applied, it is possible to directly map the DAG to the FPGA requiring little routing resources. This, however, results in wastage of cells both as unreduced realization and also the cells left unused due to the "triangular" shape of the mapping. The challenge, however, is to generate a reduced DAG structure and devise efficient routing techniques to allow for more global connections. In this chapter, techniques for reduced DAG structure realizations are presented. These DAG structures not only find applications in CA-type FPGA synthesis, but are also of major significance in other areas of synthesis.

There has been several studies on layout of graphs on planes which are also of value in CA-type FPGA mapping [10, 15, 64, 53, 54]. A typical approach is that of H-

---

The Kronecker Functional Decision Diagrams were first published in A. Sarabi, F. Ho, K. Iravani, W. R. Daasch, and M. A. Perkowski, Minimal Multi-level Representation of Switching Functions Based on Kronecker Functional Decision Diagrams, IWLS'93, Taho, CA, 1993. The earlier concepts for KFDDs were developed in papers by Perkowski [99, 100] under different terminologies. This chapter is mostly based on the joint work with the colleagues at J. W. Goethe University in Frankfurt Germany. It is based on the two papers R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski, Efficient Representation and Manipulation of Switching Functions Based on Ordered Kronecker Functional Decision Diagrams, DAC'94, San Diego, CA, 1994, and On the Computational Power of Oredered Kronecker Functional Decision Diagrams by the same authors.

tree embedding of a complete binary tree [10]. More compact schemes such as Hexagonal and Square-Connected Arrays have been also reported [54]. Here, the main focus will be that of efficient representation and manipulation of functions in form of DAG structures.

DAG structures used in logic synthesis are special kind of DAGs called *Decision Diagrams*. The most popular Decision Diagram is that of the *Binary Decision Diagrams (BDD)* popularized by Bryant [23]. While earlier works of Lee [77] and Akers [3] drew the foundations of BDDs, it was the reduction of and operations on BDDs introduced by Bryant which paved to way for the adoption and popularity of these decision diagrams. Bryant showed that through application of reduction techniques and assignment of orders to the variables, it is possible to devise a canonical representation of Boolean functions.

As pointed out by Bryant [25], solution to a large class of complex problems can result from efficient representation and manipulation of Boolean functions symbolically. Binary Decision Diagrams as a distinct method of representing functions symbolically has attracted special attention in many areas of synthesis, verification, testing, modeling, etc. Based on BDDs, problems for symbolic representation of matrices, integer programming, and spectral methods have been tackled. BDDs have found applications in such fields as combinatorial optimization, mathematical logic and Artificial Intelligence.

However, the representation of large functions has been especially problematic; since for certain classes of functions, notably multipliers, it has been known that BDDs will be of exponential size irrespective of the order of the variables. Thus research has been geared towards variations of BDDs as well as more efficient techniques for their construction. The criteria have been the ease of construction and manipulation compared with the compactness of the representation.

The more recent techniques have made it possible to handle large functions without any basic variation of the BDD itself. The dynamic variable ordering with sift-

ing introduced by Rudell [112] has made it possible to represent certain hard examples which could not be represented by any previous heuristic methods. Moreover, the variable ordering in [112] is handled by the package itself, alleviating the need for variable ordering before the actual processing. Other techniques such as the ones used in zero-suppressed BDDs [82], utilize new reduction rules in order to produce a more compact BDD representations.

Other researches have been concentrated on variations in BDDs to make realization of large functions possible. Among these variations there are those that utilize less restricted Decision Diagrams and there are other ones which augment BDDs with additional constructs.

The constructs such as General BDDs [26], or pBDDs [49], IBDDs [69], XBDDs [70], and free BDDs [151], [11] (also known as "1-time branching programs") remove the ordering constraint on BDDs at the expense of loosing the canonicity of the structure. Other Decision Diagrams such as Ternary Decision Diagrams [124] modify the BDDs by introducing a third edge for each node in the BDD.

There have also been recent attempts at varying the nodes in BDDs. These include the FDDs [72] and FBDs [138]. While the FBDs are free and thus not canonical, FDDs like BDDs provide a canonical representations of the functions. The new *Kronecker Functional Decision Diagrams (KFDD)* are the generalization of BDDs and FDDs and similarly provide a canonical representation for Boolean functions. Furthermore, they are more compact than both BDDs and FDDs and are shown in section 6.4 to be on average 35% more compact than BDDs for hard benchmark examples.

The advantage of using KFDDs over BDDs and FDDs is shown by the fact that there exists a class of functions for which BDDs are exponential while FDDs are polynomial and vice versa. Hence, just using BDDs or FDDs will prove inefficient in these cases. Furthermore, there exists a class of functions for which both BDDs and FDDs are

exponential while KFDDs are polynomial in size. These results will be described in section 6.2.

For the KFDDs to be utilized in many applications, it is of paramount importance for them to be easy to construct and manipulate. It is shown in section 6.3 that it is possible to define recursive structures such as *if then else* constructs in BDDs to produce an efficient package for easy construction and manipulation of KFDDs.

The compactness of KFDDs together with ease of construction and canonicity of KFDDs should provide a strong argument for utilization of KFDDs in many applications where BDDs have been traditionally used.

In section 6.2, the basic structure of KFDDs as well as their computational power compared to BDDs and FDDs will be presented. The efficient package for easy construction and manipulation of KFDDs is described in section 6.3. The compactness of KFDDs over BDDs and FDDs is shown over MCNC benchmark in section 6.4. A short description of the relation between KFDDs and various Two-level AND/XOR forms is given in section 6.5.

6.2. Decision Diagrams

In this section, essential definitions and properties of OKFDDs are presented. As OKFDDs are generalizations of OBDDs and OFDDs, these two structures are described further and compared with one another. Procedures for reduction of the OKFDDs are also presented.

The core of the data structures is a decision diagram (DD), which is a directed acyclic graph with some additional properties.

Definition 6.1. A *decision diagram (DD)* over $X_n := \{x_1, x_2, ..., x_n\}$ is a rooted directed acyclic graph $G=(V, E)$ with vertex set V containing two types of vertices, *non-terminal* and *terminal* vertices. A non-terminal vertex v is labeled with a variable

from X_n, called the *decision variable* for v, and has exactly two successors denoted by $low(v)$, $high(v) \in V$. A terminal vertex v is labeled with a 0 or 1 and has no successors.

The size of a DD, denoted by $|DD|$, is given by its number of nodes. If DDs are to be used as data structures in design automation, it turns out that further restrictions on their structure will be necessary. Two such restrictions are defined below:

Definition 6.2. A DD is *free* if each variable is encountered at most once on each path in the DD from the root to a terminal vertex. A DD is *complete* if each variable is encountered exactly once on each path in the DD from the root to a terminal vertex. A DD is *ordered* if it is free and the variables are encountered in the same order on each path in the DD from the root to a terminal vertex.

In the following, letter "F" will be used to describe free DDs, letter "C" to describe complete DDs, and letter "O" to denote ordered DDs.

It is possible to define certain reductions on the decision diagrams in order to reduce their size. In the following, three reduction types are given which can be partially combined:

1: Delete a node v' with sub-DDs isomorphic to sub-DDs of another node v and redirect the edges pointing to v' to point to v.

2: Delete a node v whose two outgoing edges point to the same node and connect the incoming edges of the deleted node to the corresponding successor.

3: Delete all nodes v whose successor $high(v)$ points to the terminal 0 and connect the incoming edges of the deleted node to the corresponding successor.

Definition 6.3. A DD is (t_i)–*reduced* if no reductions of type i can be applied to the DD. DD is (t_{ij})–*reduced* if no reductions of type i and type j are applicable to the DD.

Definition 6.4. Let $i \in \{1, ..., 3\}$. Two DDs, G_1 and G_2, are called (t_i)–*equivalent* iff G_2 results from G_1 by repeated applications of reductions and inverse reductions of type i. A DD, G_2, is called the (t_i)–*reduction* of a DD, G_1, if G_2 results from G_1 by repeated applications of reductions of type i and G_2 itself is reduced.

Analogously, the (t_{1j})–*reduction* $(j \in \{2, 3\})$ of a DD is defined.

A careful analysis of the proofs in [23] [60] shows that the following lemma is valid for DDs:

Lemma 6.1. The (t_k)-reduction (k = 1, 2, 3, 12, 13) of a free DD, G, is uniquely determined and can be computed in linear time in the size of G.

Until now it has not been defined how DDs can be related to Boolean functions. To do this, the following notions are helpful. Let $f: \mathbf{B}^n \rightarrow \mathbf{B}$ be a Boolean function over the variable set X_n. All nodes labeled with the same variable are denoted as a *level* in the following. Then f_i^0 denotes the *cofactor* of f with respect to $x_i = 0$, defined by $f_i^0(x) := f(x_1, .., x_{i-1}, 0, x_{i+1}, .., x_n)$ for $x = (x_1, x_2, ..., x_n) \in \mathbf{B}^n$. Similarly, f_i^1 denotes the cofactor for $x_i = 1$. Finally, f_i^2 is defined as $f_i^2 := f_i^0 \oplus f_i^1$. (Notice that the three functions f_i^0, f_i^1, f_i^2 can naturally be interpreted as Boolean functions from $\mathbf{B}^{n-1}$ to $\mathbf{B}$ defined over the variables $x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n$.) Using the above definitions, the following decompositions can be proven for an arbitrary Boolean function f:

$$f = \overline{x_i} f_i^0 + x_i f_i^1 \qquad \textit{Shannon decomposition} \qquad (6.1)$$

$$f = f_i^0 \oplus x_i f_i^2 \qquad \textit{positive Davio decomposition} \qquad (6.2)$$

$$f = f_i^1 \oplus \overline{x_i} f_i^2 \qquad \textit{negative Davio decomposition} \qquad (6.3)$$

It can be observed that Shannon, positive, and negative Davio decompositions are counterparts of σ, ρ, and $\overline{\rho}$ operators of chapter 2 in multi-level.

Furthermore, these are the only possible single-variable decompositions which can lead to the unique representation of the functions, up to negation. Single-variable

decompositions refer to all the decompositions to subfunctions f^i which totally remove a single variable from both subfunctions. It has to be mentioned that the uniqueness of the representation is only under the condition that all negations are transformed as described later on by complemented edges.

Now, the ordered Kronecker Functional Decision Diagrams can formally be defined as follows:

Definition 6.5. Each ordered DD over X_n with a uniquely determined decomposition type list (DTL), d_i, assigned to each variable x_i $(i = \{1, .., n\})$ is an *OKFDD* over X_n. Nothing else is an OKFDD when the function $f_G: \mathbf{B}^n \rightarrow \mathbf{B}$ represented by an OKFDD G over X_n is given as:

If G consists of a single node labeled with 0 (1), then

G is an OKFDD for $f = 0$ $(f = 1)$.

If G has a root v with label x_i, then G is an OKFDD

for

$$\begin{cases} \overline{x}_i f_{low(v)} + x_i f_{high(v)} & iff\ d_i\ is\ Shannon; \\ f_{low(v)} \oplus x_i f_{high(v)} & iff\ d_i\ is\ positive\ Davio; \\ f_{low(v)} \oplus \overline{x}_i f_{high(v)} & iff\ d_i\ is\ negative\ Davio \end{cases}$$

where $f_{low(v)}$ $(f_{high(v)})$ are the functions represented by the OKFDD rooted at $low(v)$ $(high(v))$.

If at every node in above definition only Shannon decomposition is applied, the OKFDD will be an OBDD. If only Davio decompositions are applied, the OKFDD will be an OFDD. As it is evident, the OKFDD is the more general decision diagram than both OBDD and OFDD.

Definition 6.6. A node in an OKFDD is called a *Shannon-node* if it is expanded by Shannon decomposition - Equation (6.1). It is called a *Davio-node* if it is expanded

by Davio decompositions - Equations (6.2) or (6.3); the latter being *negative Davio-node* and the former *positive Davio-node.*

Example 6.1. An OKFDD is shown in Figure 6.1, where the left outgoing edge at each node denotes $f_{low(v)}$. The OKFDD represents the function $x_1x_2x_4 \oplus x_1x_2\bar{x}_3 \oplus x_1\bar{x}_3 \oplus \bar{x}_1x_2x_4$. The Shannon-node decomposes the function into x_2x_4 and $x_2x_4 \oplus x_2\bar{x}_3 \oplus \bar{x}_3$, respectively. The latter is in turn decomposed into $\bar{x}_3$ and $\bar{x}_3 \oplus x_4$ through the positive Davio-node, x_2. The negative Davio-nod x_3 on the right results in x_4 and 1. □

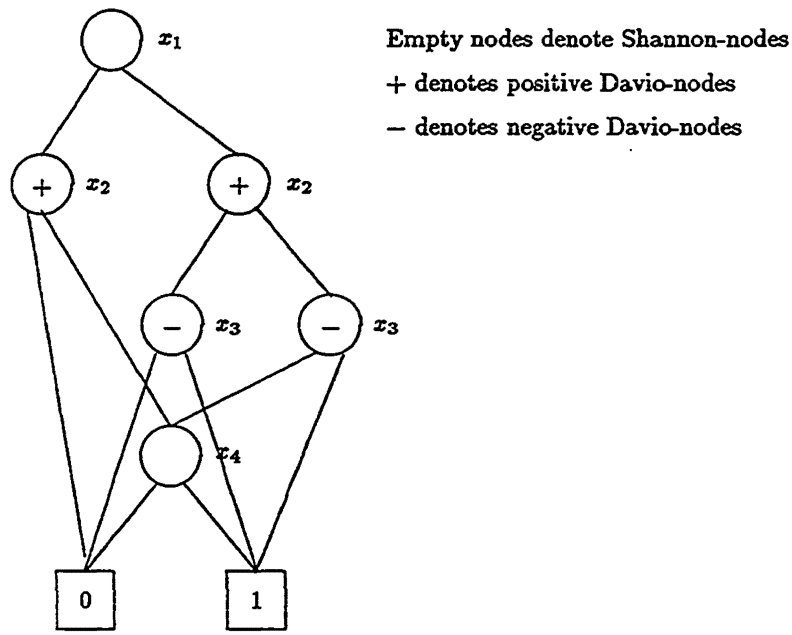

Figure 6.1 Example for OKFDD

Utilizing reductions, it is possible to define canonical representations of functions based on OKFDDs. The combination of reduction types 1 and 2 is well-known for OBDDs. This reduction has also been applied to OFDDs [72]. It can be shown that the reduction obtained by combination of types 1 and 3 is more natural for OFDDs since

only those nodes that are not further needed for the calculation are deleted [9]. This is analogous to the representation known for OBDDs. In contrast, the (t_{12})-reduced OFDDs require additional operations to reconstruct the function from the graph description.

The notion, f_G, is well defined for OFDD in the sense that $f_{G_1} = f_{G_2}$ iff G_1 is (t_{13})-equivalent to G_2. It has been proven in [72] that (t_{12})-reduced OFDDs define canonical representations for a fixed ordering π. Using Lemma 6.1, this can also be proven for (t_{13})-reduced OFDDs and following OKFDDs:

Lemma 6.2 (t_{13})-reduced OFDDs are canonical representations for Boolean functions. Furthermore, OKFDDs with (t_{12})-reduced Shannon-nodes and (t_{13})-reduced Davio-nodes are also canonical representations for Boolean functions if the decomposition types are fixed for every variable.

OBDDs and OFDDs are special cases of OKFDDs for which either Shannon decomposition or the Davio decompositions are used for all decision variables, respectively. In the case of OKFDDs in general, each variable can be split by any of the three decompositions given in Equations (6.1), (6.2), and (6.3). Hence, they can provide a more compact representation of the functions than either of the OBDDs or OFDDs. The advantage of using OKFDDs over just OBDDs or OFDDs is for example that there are classes of functions for which OBDDs are exponential in size while OFDDs with only positive Davio-nodes are polynomial and vice versa. Using the OKFDDs, it is possible to achieve a reduced size DD which is not restricted by the type of decomposition. These relations are presented next.

For the following consider a DD, G. As defined before, the function represented by the KFDD G with DTL d is denoted by $f_G{}^d$. In addition, positive Davio is represented by pD, negative Davio by nD, and Shannon with S. The following cases are of special interest: If d with $d_i \in \{pD, nD\}$ is fixed, the KFDD is an FDD and $f_G{}^d := f_G{}^d$ is called

the FDD-function of G (for DTL d).

Analogously, $f_G{}^{pFDD} := f_G{}^{d}$ is the pFDD-function of G (for DTL $d = (pD, pD, ..., pD)$) and $f_G{}^{BDD} := f_G{}^{d}$ is the BDD-function of G (for DTL $d = (S, S, \cdots, S)$).

It is possible to establish a close relation between the functions $f_G{}^{BDD}$ and $f_G{}^{FDD}$ for G being a complete DD. For an intuitive approach to this problem, first an example is given which provides a graph-theoretic interpretation for the relation between $f_G{}^{BDD}$ and $f_G{}^{pFDD}$ [9].

Example 6.2. Consider any complete DD, G. (For simplicity one may assume that G is the complete binary tree with 2^n leaves.) Then fix an assignment $a = (a_1, ..., a_n)$ to the variables $x_1, ..., x_n$. Let v be any non-terminal node of G labeled with a variable x_i. The edge $(v, low(v))$ $((v, high(v)))$ is called *BDD-active* iff $a_i = 0$ $(a_i = 1)$. The edge $(v, high(v))$ is *pFDD-active* iff $a_i = 1$ whereas the edge $(v, low(v))$ is always *pFDD-active*. A path in G is called *BDD-active* (*pFDD-active*) iff it leads from the root to a terminal node and only contains BDD-active (pFDD-active) edges.

Obviously, for a fixed assignment, a, there is exactly one BDD-active path in G and this path leads to a terminal node labeled by $f_G{}^{BDD}(a)$. Furthermore, since G is complete, it can be conclude that each pFDD-active path for $(a_1, ..., a_n)$ corresponds exactly to a BDD-active path for an assignment $b = (b_1, ..., b_n)$ with $b_i \leq a_i$ for all i. Thus, $f_G{}^{pFDD}(a) = \bigoplus_{b \leq d} f_G{}^{BDD}(b)$ and the relation between the functions represented by G if G is viewed as BDD and pFDD, respectively. □

Motivated by the above example, the *generalized τ–operator* τ_d (d is a DTL), has been introduced by Becker [41] to relate the BDD-function of G and the KFDD-function of G with DTL d For the exact definition consider the relation $\leq_d$ on $\mathbf{B}^n$, where

The τ-operator τ_d for the special case $d = (pD, ..., pD)$ was already used in [96] to analyze circuits over $\{\vee, \oplus\}$, in [13] (as the Reed-Muller Transform) to synthesize two level circuits and in [8] to show the relation between the BDD-function and the pFDD-function.

$(a_1, ..., a_n) \leq_d (b_1, ..., b_n)$ iff all components i satisfy $a_i \leq_{d_i} b_i$. Hereby, $a_i \leq_{d_i} b_i$ means $a_i \leq b_i$ $(a_i \leq \bar{b}_i, a_i = b_i)$ iff $d_i = pD$ $(d_i = nD, d_i = S)$.

Definition 6.7. Let $f \in \mathbf{B}_n$ and $d := (d_1, \cdots, d_n)$ with $d_i \in \{S, pD, nD\}$ for all i. The *generalized τ–operator* $\tau_d(f)$ is defined by

$$\tau_d(f)(x) := \bigoplus_{y \leq_d x} f(y).$$

It can be shown that τ_d is bijective and its inverse is given by $\tau_d^{-1}(f)(x) = \oplus_{\bar{x} \leq_d \bar{y}} f(y)$. Through induction on the number of nodes in G, it can be concluded that the BDD-function and the KFDD-function of a complete DD G can be computed from each other via the generalized τ-operator.

Theorem 6.1. [41] For each complete DD, G, and each DTL d, it holds that $f_G{}^d = \tau_d(f_G{}^{BDD})$ and $f_G{}^{BDD} = \tau_d^{-1}(f_G{}^d)$.

Theorem 6.1 is valid only for complete DDs. Fortunately, with a similar proof as in [8], it can been shown [41] that each free (ordered) KFDD can be transformed into an equivalent complete KFDD the size of which grows at most by a factor of $O(n)$. Combining Theorem 6.1 together with this property, it is possible to transfer results about (free, ordered) BDDs to KFDDs and vice versa. In particular, the existence of classes of functions can be proven which are good for OFDDs and bad for OBDDs and vice versa:

Theorem 6.2. [41] Consider OFDDs for a fixed DTL, d, with $d_i \in \{pD, nD\}$ for all i. There exist families of Boolean functions $(f_m)_{m \in \mathbf{N}}$, $\mathbf{N}$ denoting the set of natural numbers, such that each OBDD (OFDD) for f_m has size exponential in m $(2^{\Omega(m)})$, while for each ordering of the variables, there exists an OFDD (OBDD) of polynomial size for f_m.

It follows from the above theorem that it is advantageous to consider both OKFDDs with Shannon and OKFDDs with Davio nodes than using only type of these Decision Diagrams. This section is concluded by showing that mixed-type OKFDDs, i.e.

OKFDDs containing both Shannon and Davio nodes in one single DD, are even more powerful.

Theorem 6.3. [41] Let the DTL d be fixed with $d_i \in \{pD, nD\}$ for all i. There exists a family of Boolean functions $(f_m)_{m \in N}$, such that each OFDD (with DTL d) and each OBDD for f_m has size exponential in m, while there exist OKFDDs for f_m of polynomial size.

Proof. The outline of the proof given by Becker [41] is as follows: Let g_1 (g_2) be a function that can be represented efficiently by an OBDD (OFDD), but only has OFDDs (OBDDs) of exponential size (see Theorem 6.2). One may further assume that g_1 and g_2 depend on disjoint sets of variables. Then the function $f = x_1 \cdot g_1 \oplus x_2 \cdot g_2$ (x_1, x_2 being new variables) can obviously be represented efficiently by an OKFDD. However, neither a small OBDD nor a small OFDD exists for f: Assume that there exists a small OBDD (OFDD) for f. Since cofactoring is an efficient operation, an efficient OBDD (OFDD) for g_2 (g_1) should be obtained in contradiction to the assumption. *QED*

The OKFDDs can further be reduced in size by using complemented edges. The constraints of complemented edges to maintain a canonical form for BDDs were given in [19]. Similarly, complemented edges can be used for the representation of a function and its complement by the same node in the case of Davio-nodes [9].

The difference between the Shannon-nodes and Davio-nodes is that different restrictions have to be imposed on where complemented edges are set in order to obtain a canonical form. These restrictions are given in [9]. The OKFDDs with complemented edges given in [19] for Shannon-nodes and those shown in [9] for Davio-nodes are unique.

6.3. Implementation of an OKFDD Package

In this section, implementational details of the OKFDD package are described and

OKFDD manipulation algorithms are introduced. The package is built on top of the OFDD package in [9].

6.3.1. Technical Details

The programming techniques and methods of implementation used to speed-up the package are similar to other packages used for representation and manipulation of OBDDs and OFDDs [19, 81, 9]. Hence, these techniques are only briefly reviewed.

For the fast availability of the functions, a hash-based unique table is used to store the nodes. A computed table is implemented for the optimization of the synthesis algorithms. Furthermore, level lists are used for the management of the nodes of each stage.

In this way, fast access to the nodes is possible by the algorithms and efficient local transformations can be performed. The memory management is done by garbage collection. The nodes are only deleted if the storage place is needed for other nodes. Thus, it would not be needed to recompute the results each time if they were used earlier on. By the unique table, different OKFDDs can share the same sub-OKFDDs. Therefore, several functions can efficiently be represented at the same time.

6.3.2. The Construction and Operations on OKFDDs

First, the XOR-operation is presented as it provides the basis for construction of certain other operations. Notice that for two functions, f and g, decomposed by positive Davio expansion, one has:

$$f \oplus g = (f_0 \oplus x_i f_2) \oplus (g_0 \oplus x_i\, g_2) = (f_0 \oplus g_0) \oplus x_i (f_2 \oplus g_2) \qquad (6.4)$$

This equation makes it possible to recursively split up a positive Davio-node into its left and right subgraphs. The algorithm for negative Davio-nodes is performed analogously. This provides an efficient algorithm for Davio-nodes while the basic XOR-operation for Shannon-nodes is based on the following equation:

$$f \oplus g = \bar{x}_i (f_0 \oplus g_0) + x_i (f_1 \oplus g_1) \qquad (6.5)$$

```
kfdd_xor_kfdd (F, G) {
    if = (terminal case) {
        return result;
    } else if (computed-table has entry (F, G)) {
        return result;
    } else {
        let v be the top variable of (F, G);
        if Shannon(v) {
            low(v) = kfdd_xor_kfdd (F0, G0);
            high(v) = kfdd_xor_kfdd (F1, G1);
            if (high(v)==low(v)) return low(v);
        } else if pos_Davio (v) {
            low(v) = kfdd_xor_kfdd (F0, G0);
            high(v) = kfdd_xor_kfdd (F2, G2);
            if (high(v)==0) return low(v);
        } else {
            low(v) = kfdd_xor_kfdd (F1, G1);
            high(v) = kfdd_xor_kfdd (F2, G2);
            if (high(v)==0) return low(v);
        }
        R = find_or_add_unique_table (v, low(v), high(v));
        insert_computed_table (F, G, R);
        return R;
        }
    }}
```

Figure 6.2. Algorithm for XOR-operation

The resulting algorithm for XOR-operation on two OKFDDs is presented in Figure 6.2.

The efficient XOR-operation allows the construction of OKFDDs from OBDDs. Here, one starts with a recursive computation in the OBDD. At each Shannon-node, v (labeled x_i), which is to be transformed into a positive Davio-node, the Davio-node, v', corresponding to the function represented by v is constructed. The successor $low(v)$ can be directly used for $low(v')$ in positive Davio-node since it represents the cofactor with respect to $x_i = 0$. For the case of negative Davio-node, $high(v)$ needs to be used. For the successor $high(v')$, the XOR-operation has to be performed on the successors of v. But this operation can be performed efficiently for OBDDs and OKFDDs. The construction algorithm is given in Figure 6.0. In this scheme, a list with the decomposition type of the variable is passed to the function. The algorithm is based on static order and decomposi-

tion types of variables. Although the operations for each node can be performed efficiently, the algorithm has exponential worst case behavior.

```
bdd_to_kfdd (F) {
    if (terminal case) {
        return result;
    } else if (computed-table has entry (F)) {
        return result;
    } else {
        let v be the top variable of (F);
        if Shannon(v) {
            low(v) = bdd_to_kfdd (F_0);
            high(v) = bdd_to_kfdd (F_1 ));
            if (high(v) == low(v)) return low(v);
        } else if pos_Davio (v) {
        low(v) = bdd_to_kfdd (F_0);
            high(v) = bdd_to_kfdd (ite(F_0, F̄_1, F_1));
            if (high(v) == 0) return low(v);
        } else {
            low(v) = bdd_to_kfdd (F_1);
            high(v) = bdd_to_kfdd (ite(F_0, F̄_1, F_1));
            if (high(v) == 0) return low(v);
        }
        R = find_or_add_unique_table (v, low(v), high(v));
        insert_computed_table (F̄, R);
        return R;
    }}
```

Figure 6.3. Algorithm for OKFDD-construction

An algorithm to transform OKFDDs to OKFDDs with other choice of decomposition rules easily follows from the algorithm in Figure 6.3 with a slight modification. Here, only the recursive call of *ite* has to be substituted with the procedure *kfdd_xor_kfdd ()*. Additionally, different cases for the availability of the successors should be distinguished. For instance, if a negative Davio-node must be transferred to a Shannon-node, the function f^0 must first be computed.

The realization of the AND-operation turns out to be more complicated for Davio-nodes in comparison to the XOR-operation. The following recursive equation holds for positive Davio-nodes:

$$f \cdot g = (f_0 \oplus x_i f_2) \cdot (g_0 \oplus x_i g_2) \tag{6.6}$$
$$= (f_0 \cdot g_0) \oplus x_i((f_2 \cdot g_2) \oplus (f_0 \cdot g_2) \oplus (g_0 \cdot f_2))$$

This equation again defines a recursive algorithm similar to the one from Figure 6.2 which has exponential worst case running time [9]. The same results hold for negative Davio-nodes. However, for OKFDDs with a constant number of levels, where the Davio expansion is performed, the operation is polynomial.

The negation of a function, f, for a Davio-node can be computed by observing that $\bar{f} = 1 \oplus f$. Thus, the operation requires an XOR-operation with the constant 1 in the OKFDD. Since the package uses complemented edges, this operation can be performed in constant time.

Now, using the algorithms for the XOR-, AND-, and NOT-operations, any binary operation can be realized.

For an OKFDD, G, the restriction $G|_{x_i = c}$ for variable x_i and constant c can be computed by traversing the graph and performing the corresponding substitutions. The case for Shannon-nodes is given in [19]. For the case of positive Davio-nodes, if $x_i = 0$, edges from nodes v with label x_i to $high(v)$ have to be deleted. If nodes with indegree 0 result, they and their outgoing edges are also deleted. All this can be done in linear time. If $x_i = 1$, then at each node, v, with label x_i and subfunctions g_0 and g_1, the following has to be done. As before, the *high*-edge has to be deleted, at the *low*-edge an OKFDD for $g_0 \oplus g_1$ must be rooted, i.e., an XOR-operation has to be executed. For negative Davio-nodes, a similar procedure is required.

6.3.3. Optimization of OKFDD-Size

While the variable ordering plays a dominant role in the identification of the minimal OBDD representation of the functions, in OKFDDs both the ordering and the decomposition type are important. Depending on the order of the variables and the particular decomposition among the possible three, the size of the OKFDD can vary from

linear to exponential [9]. It is well-known that in the case of OFDDs and OBDDs, the size of the decision diagram can be minimized by an exchange of adjacent variables [51, 9]. It can be proven that this idea can be extended to OKFDDs. Therefore, it is also possible to use all techniques based on exchanging of adjacent variables for OKFDDs. Especially the sifting algorithm, window permutation, and exact minimization algorithms [112, 68] can be used.

By the sifting algorithm, the variables are sorted into decreasing order based on the number of nodes at each level and then each variable is traversed through the DAG in order to locate its local optimum position while all other variables remain fixed.

The dynamic variable ordering based on sifting algorithm can be utilized in the minimization of OKFDD sizes as well. In this scheme, the sifting algorithm is modified so that at each position all three types of decompositions are tested and the local optimum is chosen based on both the position and the type of the decomposition of the variable. Thus, each time a variable changes the decomposition rule, the new local optimal position is determined by exchange of adjacent variables.

It is also possible to use a restricted sifting operator for the dynamic variable ordering, i.e. to only exchange the variable with new decomposition type. This method guarantees that the new sifting operator differs from the original sifting only by a small constant factor, since all experiments performed have shown that the time for additional XOR operation needed for the change of the decomposition type is negligible compared with the time needed to exchange variables.

Since sifting can be very time consuming, in a variation of sifting operator, lower and upper bounds are allowed for the reordering [41], i.e. one parameter gives the maximal growth and a second parameter gives the lower bound on when to stop the sifting.

The minimization scheme can be summarized as follows: A heuristic or random order OBDD is constructed initially. If the size of the OBDD exceeds a chosen number

of nodes, the reordering is applied. Here, each variable traverses through the levels, exchanging its level with its adjacent variable of lower level. With each exchange, an OKFDD is constructed with the chosen variable expanded with each of the Davio decompositions respectively and the optimal position of the variable is found by sifting. This is repeated for all levels and the local optimum position and type of decomposition is designated. The procedure is repeated for a next variable, and so on. Thus, only one variable is changed in each step and from this point of view the heuristic is very simple.

6.4. Experimental Results

The experimental results confirm the advantage of OKFDDs over OBDDs and OFDDs and show the efficiency of the approach presented.

Herein the size of OKFDDs is compared with the size of OBDDs and OFDDs. First, the minimal size for arithmetical benchmark circuits [21] is considered. The benchmarks are minimized by an algorithm similar to the one presented in [68]. The results are given in Table 6.1. The optimal size can only be determined for small benchmarks due to the exponential running time of the minimization algorithm. For small functions there is only a minor improvement. But for larger functions there are larger gains, e.g. Z5xp1.

Name	in	out	OBDD	OFDD	OKFDD
b1	3	4	6	5	5
c17	5	2	6	8	6
cm82a	5	3	11	9	9
majority	5	1	7	7	7
rd53	5	3	16	13	13
rd73	7	3	30	21	21
wim	4	7	19	22	17
Z5xp1	7	10	41	45	28

Table 6.1. Comparison of optimal *OKFDD* with optimal *OBDD* and optimal *OFDD* with positive Davio-nodes

Name	in	out	OBDD	OFDD	OKFDD
add6	12	7	68	45	44
apex7	48	37	296	360	266
bc0	26	11	535	727	431
chkn	29	7	322	459	279
cps	24	109	1040	1293	766
f51m	8	8	38	35	25
intb	15	7	656	624	480
mlp4	8	8	134	107	106
radd	8	5	33	20	19
ts10	22	16	193	155	155
total			3315	3825	2571

Table 6.2. Comparison of *OKFDD* with *OBDD* for certain benchmark functions

In a next series of experiments, medium size benchmarks are considered for which the optimal ordering can not be determined. For this purpose, some benchmarks from [21] and MCNC are used. These results are shown in Table 6.2.

For all types of decision diagrams (OBDDs, OFDDs and OKFDDs), dynamic variable ordering [112], starting from the original variable ordering were used. In the last row the *total* sum of nodes for all the considered benchmarks is given. As it can be observed, the average gain is 25% when the simple ordering heuristic from previous Section is used.

In the next set of test, hard benchmark functions were examined. Here, a node limit of 140,000 nodes (and for larger benchmarks a limit of 280,000 nodes) were set. Sifting is performed when the OKFDD becomes larger than 70,000 nodes and a garbage collection would not delete more than 30% of the nodes. During sifting, the OKFDD is allowed to double in size. A lower bound for the sifting is not set. The results are presented in Table 6.3, where the numbers in the table denote thousands of nodes.

Name	in	out	OBDD	[112]	OKFDD
C432	36	7	1.2	1.2	1.1
C499	41	32	30.3	44.8	16.3
C880	60	26	4.5	9.1	4.0
C1355	41	32	29.5	36.2	16.3
C1908	33	25	7.1	12.4	4.9
C2670	233	140	6.6	6.6	3.8
C3540	233	140	24.0	27.2	23.1
C5315	178	123	2.7	3.1	1.8
C7552	207	108	26.0	8.2	20.9
s1423	91	79	4.9	5.7	1.5
des	256	245	3.0	3.3	2.9
pair	173	137	3.3	4.5	2.9
rot	137	107	9.4	5.0	3.8
total			152.5	167.3	103.3

Table 6.3. Comparison of the number of nodes for *OKFDD* vs *OBDD*.

In column OBDD, the results for OBDDs using the sifting operator are presented. All siftings started with the initial ordering as it occurs in the benchmarks. The OBDDs obtained by the presented method are also compared with the OBDDs obtained in [112], where additionally an initial topology-based heuristic was used. In all cases but one, C7552, the OBDDs show equal or better results. The results for the OKFDDs are given in the last column. The improvements observed reach up to 75%. In the last row the *total* sums of nodes for all the considered benchmarks are given. As it can be observed, even despite the huge effect of initial random ordering, especially in the case of C7552, the average gain of OKFDDs over OBDDs is about 35% when the simple ordering heuristic from Section 3 is used. As reported by Rudell in [112], the heuristic start can make substantial difference in certain cases. The random start for C7552 is reported to result in 23,700 nodes in [112].

It is further possible to incorporate more sophisticated schemes such as changing several decompositions in parallel. From the available results it can be inferred that the use of OKFDDs can potentially have drastic influence on realizations for which efficient OBDDs do not exist.

6.5. Relations Between KFDDs and Two-level AND/XOR Forms

Many Two-level AND/XOR canonical forms can be constructed by flattening certain KFDDs. These forms range from sum of minterms to *QKRM* forms. In this section, a brief description of these relations will be presented.

OKFDDs can be considered to be the counterpart of the multi-level *KRM* forms. It can be recalled that the *KRM* forms are constructed by the application of ρ, $\bar{\rho}$, and σ operators starting from the three matrices T_1, T_2, and T_3. These operators result in the same equations as positive Davio, negative Davio and Shannon, Equations (6.2), (6.3), and (6.1). In flat forms, OKFDDs translate into *KRM* forms where the order of variables is not of importance. If only Equation (6.2) is used for decomposition, the corresponding Two-level form would be that of *RMC*. If both Equations (fc.2) and (6.3) are used, the DD will be that of FDD and the corresponding Two-level form will be that of fixed polarity AND/XOR forms. If only Equation (6.1) is used, the corresponding Two-level form will be that of sum of minterms.

When FKFDD and variations in ordering are used, larger families of Two-level forms can be identified. When the KFDD is ordered in terms of the variables but each branch can have different decomposition, the DD could be termed *PKFDD* as it corresponds to the Two-level *PKRM* form in flattened form. When KFDD is also free, then the corresponding Two-level form will be that of *QKRM*. Of course, it is possible to identify other Two-level forms where the order of the decomposition is fixed but the order of the variables is free, etc. The reference [100] can be referred for more discussion on these relations.

In the following, some relations between an *OKFDD* and its corresponding *KRM* form will be introduced. These can be used for the investigation between a minimal *KRM* representation and a corresponding *KFDD*. The results are mainly for single output functions.

Theorem 6.4. Let Ω be a *OKFDD* corresponding to a given Two-level *KRM* form of a Boolean function. Let the number of times a literal, x_i, appears in the monoterms at any level L_j be given by $L_j(x_i)$ and let r be the level at which x_i is split. Then $L_k(x_i) = L_r(x_i), \forall\, k \leq r$.

Proof. Let v be a vertex in the *OKFDD*. Let x_j be the decision variable at that vertex and let us assume that this variable is Shannon_type. Then $f_{low(v)}$ will include all those terms which have x_j in negative polarity and $f_{high(v)}$ will include all the terms which have x_j in positive polarity. As these terms are mutually exclusive, the terms will be divided between the two successors exclusively. With these terms, all variables x_i, other than the decision variable, x_j, will be just separated exclusively. So there will be no change in the number of times x_i occurs from one vertex to the next level. This is true for all vertices in the same level, so the total number of occurrences of x_i is preserved from one level to the next.

The same result holds for Davio-nodes. This will be shown for positive Davio-nodes. Negative ones are similar. Again let v be a vertex in the DD. Let x_j be the decision variable for this vertex and assume the vertex is a positive Davio-node. As the Two-level *KRM* and the *OKFDD* correspond to each other, the decision variable in the *OKFDD* will have the same polarity as it appears in the Two-level form. Therefore, $f_{low(v)}$ will include all those terms which do not include the decision variable x_j and the exclusive sum of $f_{low(v)}$ and $f_{high(v)}$ will include all those terms which will include x_j. Again these terms are mutually exclusive and similarly as above, the total number of occurrences of any other literals x_i will be preserved from one level to the next.

The decomposition process continues until the variable x_i is split and in that case, it will not occur in any of the subsequent levels. *QED*

This means that the number of times a literal occurs at any level can be found directly from the corresponding Two-level expression.

Theorem 6.5 shows the independency of the nodes at certain level of the DD with the order of the prior decision variables.

Theorem 6.5 Let Ω be an *OKFDD* corresponding to a Two-level *KRM* form of a Boolean function. Let $x_1, x_2, \ldots, x_n$ be the variables expanded in the *OKFDD*. Let N_r be the number of the nodes at the rth level. Then N_n will be the same no matter in what order the variables are split. Furthermore, the nodes will be the same with only difference of their locations in that level.

The proof is rather involved and will not be given here. It, however, follows basic properties of Boolean difference, i.e. $\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$ and $\frac{\partial (f \oplus g)}{\partial x_i} = \frac{\partial f}{\partial x_i} \oplus \frac{\partial g}{\partial x_i}$. In addition, due to the correspondence between the *OKFDD* and the two-level *KRM* form, $f_{\bar{x}_i} f_{\bar{x}_j} = f_{\bar{x}_j} f_{\bar{x}_i}$, $f_{\bar{x}_i} f_{x_j} = f_{x_j} f_{\bar{x}_i}$, $f_{x_i} f_{\bar{x}_j} = f_{\bar{x}_j} f_{x_i}$, and $f_{x_i} f_{x_j} = f_{x_j} f_{x_i}$.

6.6. Summary

In this chapter, this author presented KFDDs, a compact multi-level representation of functions as a decision diagram. KFDDs introduced were shown to be a generalization of both BDDs and FDDs. Their compaction was presented both theoretically and experimentally.

The main personal contribution of this author was the application of the sifting and dynamic ordering to KFDDs. In this way, it was possible to show experimentally the compactness of these decision diagrams. While KFDDs were popularized before by the author, it was the compactness for the case of large functions as well as the ease of construction and manipulation that needed to be demonstrated. This task was accomplished in a joint effort with colleagues at J. W. Goethe University at Frankfurt. While, iterative generation of the Davio nodes was independently developed by the author and these colleagues, the package itself was developed based on their FDD package. In the final section, the relation between various DDs and Two-level AND/XOR forms were given.

Two theorems were introduced by this author for the relation between the OKFDD and its corresponding Two-level *KRM* form.

As the MUX, and AND/XOR nodes of KFDDs are available in many CA-Type FPGAs, using graph embedding techniques, it is possible to map these diagrams to the FPGA architectures. They can also be used for other FPGAs as for example in a bin packing approach to LUT-Type FPGAs. The application of KFDDs is, however, more general than just FPGAs as they can be investigated for many applications where BDDs are currently being used. Their main advantage is being canonical as well as more compact than BDDs with ease of manipulation and construction. It is the opinion of this author that this is a very important contribution to logic synthesis at large to which the author actively participated in.

Chapter 7

Design For Testability Properties of AND/XOR Networks

7.1. Introduction

The XOR forms are highly testable and provide a major advantage over AND/OR logic in this regard. Testability of XOR gates has long been known. However, AND/XOR functions have the special property that their required test set is independent of the actual function being realized. This property is of major importance in the design process. Among all the AND/XOR forms, it is the *CGRM* that has the least number of tests required.

The testability properties are of major importance in VLSI. However, for FPGAs still testability is considered to be of a high value. While it is claimed that the FPGAs are fault free themselves, and taking this claim as a fact, there are still arguments for testing. The problems with programming the FPGA, defects that might be caused during their usage, etc. are among these arguments. The testing problem is of major concern in other technologies. The problem of test generation is known to be NP-complete [52], Hence, for large functions this becomes quite a formidable task. Having a universal set of tests for any function obviously reduces this problem drastically. Realization of Boolean functions in AND/XOR forms is exactly the case which results in an independent test set. This property will be presented in this chapter. Before presenting the testability characteristics of these forms, certain terms and concepts related to testing will be provided.

This chapter is based on the original paper, A. Sarabi, and M. A. Perkowski, Design for Testability Properties of AND/XOR Networks, IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg, Germany, September 1993.

The basic aim of testing at the chip level is the identification of faults in a circuit. A *fault* of a circuit is a physical defect of one or more components. The basic faults in any VLSI integrated circuit can be classified into two major categories, that of *perimetric faults* and *logical faults*. While the perimetric faults are related to the physical defects, the logical faults are associated with the logical aspects of the circuits. Perimetric faults are those responsible for alterations in the magnitudes of a circuit parameter, causing a change in some factor such as the circuit speed, current, or voltage. Logical faults on the other hand, are the ones that cause changes in the logic function of a circuit element or an input signal to some other logic function. The perimetric faults are of concern in the structural level design of logic circuits and the logical faults are associated with the functional level and this is precisely where the properties of the AND/XOR forms are of paramount importance. Other faults include those of *delay faults*, e.g. slow gates which usually only affect the timing performance and could result in hazards or critical races, and *intermittent faults* which occur only in some intervals and are very difficult to detect.

The logical faults in their place can be also divided into two classical classes of *bridging faults* and *stuck-at-faults*. Bridging faults occur particularly in MOS LSI circuits. These are mainly due to a short connection in the circuit between two or more lines resulting in circuit malfunction. This short circuit can be modeled as either a wired-AND or wired-OR function. The stuck-at-faults are themselves divided into stuck-at-1 and stuck-at-0 which occur at the inputs of the logic gates and cause the inputs to remain either at 1 or 0 permanently.

AND/XOR forms, due to the testability characteristics of the XOR, have major properties of interest in design for testability. The small number of test sets and their independence of the Boolean function itself, as well as their controllability and observability are the major factors. This is due to the property of XOR that any change over one of its inputs is directly reflected on its output. Among all of these forms, Reed-Muller

canonical forms require the least number of test sets for detecting stuck-at and bridging faults. It will be shown in the next section that the *CGRM* forms have also the same number because the input variables retain the same polarity throughout the expansion. The other forms are known [105] to require larger test sets. In the following, the testability properties of the *RMC*, *CGRM*, and *CRMP* forms will be presented. This will be next contrasted with other forms. The presentation will be divided into discussion of stuck-at-faults and bridging faults. Each of these faults will also be divided into further subdivisions. The testability of the circuits realized from KFDDs has been studied by our colleagues in Frankfurt and this topic will not be mentioned here.

7.2. Detection of Stuck-at-Faults in CGRM Networks

The discussion of stuck-at-faults (SAF) can be presented in terms of single SAF and multiple SAF. Single SAF can further be investigated depending on whether the primary inputs are fault-free or not. These cases will be described in the following.

The first author to find the testability properties of Reed-Muller forms was Reddy [109]. The results obtained by Reddy for single SAFs are summarized in the following:

1) *"If the primary inputs leads are fault-free, then there exists a realization for an arbitrary n–variable logic function that requires a fault detection test set with only $n+4$ tests and this test is independent of the function being realized.*

2) *If the primary input leads could be faulty, then only $n+4+2n_e$ tests are required for detecting faults, where n_e is the number of variables appearing in an even number of terms in the Reed-Muller expansion for the function being realized.*

3) *If the primary input leads could be faulty, then by adding an extra observable output and an extra AND gate, the (n + 4) tests of 1) will be sufficient and these tests will again be independent of the function being realized."*

The circuit here is assumed to be composed of a cascade of XOR gates with a secondary input from AND gates each composed of different combinations of variables. An example of the this scheme for the function $f = 1 \oplus x_1x_2 \oplus x_1x_4 \oplus x_1x_2x_3 \oplus x_2x_3x_4 \oplus x_1x_2x_3x_4$ is shown in Figure 7.1.

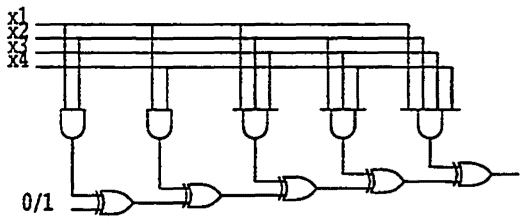

Figure 7.1 The cascade network for *RMC*

In order to detect single SAF in a cascade of XOR gates, it is sufficient to apply a set of tests covering all possible input combinations to each cell [109]. The following test set satisfies this purpose:

$$a_0x_1x_2x_3 \cdots x_n$$

$$T_1 = \begin{Bmatrix} 0\,0\,0\,0\,\ldots\,0 \\ 0\,1\,1\,1\,\ldots\,1 \\ 1\,0\,0\,0\,\ldots\,0 \\ 1\,1\,1\,1\,\ldots\,1 \end{Bmatrix}$$

Any stuck-at-0 fault at any AND gate input or output can be detected by applying either of the test inputs [0111...1, 1111...1]. Similarly, any stuck-at-1 fault at output of any AND gate can be detected by applying either of the test inputs [0000...0, 1000...0]. A stuck-at-1 fault at any input of the AND gates can be detected by the set T_2:

$$T_2 = \left\{ \begin{array}{l} \\ d\ 0\ 1\ 1\ \ldots\ 1 \\ d\ 1\ 0\ 1\ \ldots\ 1 \\ d\ 1\ 1\ 0\ \ldots\ 1 \\ d\ 1\ 1\ 1\ \ldots\ 1 \\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot \\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot \\ d\ 1\ 1\ 1\ \ldots\ 0 \\ \\ \end{array} \right\}$$

where d stands for don't care. The total number of the tests in $T = T_1 \cup T_2$ is then the sum in T_1 and T_2 which is $4 + n$.

The above results are true when none of the primary inputs are faulty. If any of the primary inputs are faulty as well, additional tests are required. As the XOR gate detects an odd number of changes, the $n + 4$ tests above detect faults at those primary inputs that occur in an odd number of AND gates. For the primary inputs that occur in an even number of AND gates, as is suggested by Reddy, it is required to apply two tests for each of these inputs. In this method, the inputs of interest that occur in products with the smallest number of literals are chosen. The stuck-at-one is identified by choosing the inputs occurring in the products one at a time and assigning them zero along with the literals that do not occur in this product. The other variables in the product are assigned the value of 1. Detection of stuck-at-zeros is similar with the difference that the input variable of interest is assigned the value of 1 instead. These tests detect the stuck-at-0 and stuck-at-1 faults in the faulty primary inputs occurring in an even number of times. This leads to the point 2) above.

The idea behind point 3) is that having an additional AND gate with its inputs being the primary inputs that occur in an even number of products can reflect the faults at these primary inputs. The test set in point 1) will then be adequate to detect all the SAFs. In order to make sure the faults detected are not due to the primary inputs, another AND gate can be added exactly the same as above. For further discussion the reader can refer to [109].

The test sets given by Reddy above were shown by Kodandapani [74] to be reducible. Kodandapani has shown that by assigning specific values to the don't cares in the matrix T_2, and a certain scheme of reorganizing the terms, one of the tests in T_1 can be reduced. In this way, the number of tests required to detect any single stuck-at-fault in an AND gate or a single faulty XOR gate can be reduced to $n+3$.

The above results were for the cases where only single SAFs are involved. When there are multiple faults involved, the test set is again shown to be independent of the function. Saluja and Reddy [114] have shown that to detect t faults, $t \geq 1$, only

$$4 + \sum_{i=1}^{\lfloor \log_2 2t \rfloor} \binom{n}{i} \tag{6}$$

tests are required to detect all t-multiple stuck-at-faults (where $\lfloor x \rfloor$ stands for the integer part of x). With addition of an extra AND gate and one observable output, the single stuck-at-faults can be detected as well. Furthermore, Saluja has shown that by addition of extra observable outputs, the same $n+4$ independent tests can detect all single and multiple stuck-at-faults [113].

It can be easily shown that any universal test set generated for detection of single stuck-at-faults of an *RMC* form can be modified for any *CGRM* form by just inverting the test bits for those variables which are of negative polarity in the *CGRM*. As the input to the AND gates are the complements of the test bits for the case of complemented variables, this inversion makes the test bit to be equivalent to the *RMC* network described above. Hence the same results hold true for the *CGRM* networks. This is shown by an example below:

Example 7.1. Let a network be represented by $1 \oplus x_1x_2 \oplus x_1x_4 \oplus x_1x_2x_3 \oplus x_2x_3x_4 \oplus x_1x_2x_3x_4$. This is an *RMC* network with the following universal tests for detection of the stuck-at and bridging faults:

$$a_0 x_1 x_2 x_3 x_4$$

$$T_1 = \left\{ \begin{array}{ccccc} & & & & \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ & & & & \end{array} \right\}$$

$$T_2 = \left\{ \begin{array}{ccccc} & & & & \\ d & 0 & 1 & 1 & 1 \\ d & 1 & 0 & 1 & 1 \\ d & 1 & 1 & 0 & 1 \\ d & 1 & 1 & 1 & 0 \\ & & & & \end{array} \right\}$$

Now let us assume a *CGRM* network has been given in the form $1 \oplus \bar{x}_1 x_2 \oplus \bar{x}_1 \bar{x}_4 \oplus \bar{x}_1 x_2 x_3 \oplus x_2 x_3 \bar{x}_4 \oplus \bar{x}_1 x_2 x_3 \bar{x}_4$. In this form, x_1 and x_4 have negative polarities and thus their respective columns are complemented with respect to the above *RMC* network. The universal tests for detection of the stuck-at and bridging faults of the *CGRM* network will be:

$$a_0 x_1 x_2 x_3 x_4$$

$$T_1 = \left\{ \begin{array}{ccccc} & & & & \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ & & & & \end{array} \right\}$$

$$T_2 = \left\{ \begin{array}{ccccc} & & & & \\ d & 1 & 1 & 1 & 0 \\ d & 0 & 0 & 1 & 0 \\ d & 0 & 1 & 0 & 0 \\ d & 0 & 1 & 1 & 1 \\ & & & & \end{array} \right\}$$

□

If the inversions are to be generated internally, an extra AND gate with observable output can be added to the network. This AND gate should have as input all the variables that appear with negative polarity in the *CGRM* form. This gate will detect the

faults produced by any of the inverters.

It has to be noted that the same tests would detect the stuck-at-faults of a multi-output *CGRM* network. The difference is that the observable points will be the same as the outputs and they have to be examined for each of the functions being realized.

7.3. Detection of Bridging Faults of CGRM Networks

The *RMC* networks not only have universal test sets for detection of stuck-at-faults, but there exist similar schemes for detection of the bridging faults. It has been shown [16] that with certain modifications the same universal tests for detection of stuck-at-faults can be utilized to detect bridging faults of the *RMC* networks.

Before describing the bridging fault detection schemes, certain classification of the bridging faults will be described. Bridging faults result from short connections in the circuits. These shorts can occur at the inputs of logic gates, input lines to different logic gates, or between the lines of the same logic level. These bridging faults are called *intragate*, *intergate*, and *intralevel* respectively. The bridging faults can in turn be either wired-AND or wired-OR function, depending on positive or negative logic.

The results obtained by Bhattacharya, et al regarding the bridging faults detection of *RMC* networks are as follows:

Theorem [Bhattacharya] An *RMC* network of n variable function can be augmented by adding an extra AND gate, with all input variables as its input, so that the universal test set T, of cardinality $n+4$ is sufficient to detect the different intralevel OR-bridging faults, as well as all single stuck-at-faults.

This augmentation for the network in Figure 7.1 is shown in Figure 7.2.

Theorem [Bhattacharya] An *RMC* network of n variable function can be augmented by adding an extra OR gate so that the universal test set T_u, of cardinality $2n+4$ is sufficient to detect the different intralevel AND-bridging faults, as well as all single

stuck-at-faults. $T_u = T \cup T_\alpha$ where

$$T_\alpha = \begin{array}{c} a_0 x_1 x_2 x_3 \cdots x_n \\ \left\{ \begin{array}{l} d\ 1\ 0\ 0\ \ldots\ 0 \\ d\ 0\ 1\ 0\ \ldots\ 0 \\ d\ 0\ 0\ 1\ \ldots\ 0 \\ d\ 0\ 0\ 0\ \ldots\ 0 \\ \cdots\cdots\cdots \\ \cdots\cdots\cdots \\ d\ 0\ 0\ 0\ \ldots\ 1 \end{array} \right. \end{array}$$

The scheme for addition of the OR gate is similar to the one shown in Figure 7.2 with the difference that the augmented AND gate is replaced with an OR gate of all input variables.

For the case of *CGRM* networks, Bhattacharya, et al propose a different circuit augmentation scheme. In this case for detection of OR-bridging faults, it is proposed that three additional gates be added to the network. These gates are one n-input AND gate of all primary inputs, one n-input OR gate of all primary inputs, and an n_1-input AND gate, where n_1 is the number of complemented literals in the *CGRM* expansion. The inputs to this AND gate are derived from the outputs of the inverters producing the negated inputs. The test set devised for the *RMC* network to detect AND-bridging faults can then be applied to detect the OR-bridging faults of the *CGRM* network. A similar augmentation can be devised for detection of the AND-bridging faults for the *CGRM* network.

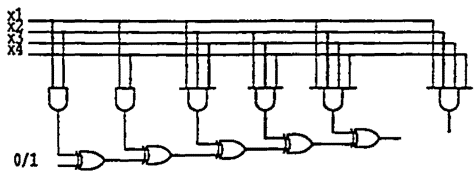

Figure 7.2 Augmentation of the network in Figure 7.1 for detection of OR-bridging faults

If no augmentation is to be incorporated, Damarla and Karpovsky [36] give an

upper bound for the number of test patterns to detect all single Stuck-at-faults and all single detectable AND and OR bridging faults of an *RMC* network. In this scheme for an *RMC* network with k outputs and n inputs ($k \leq 2^n$), at most $3n+5$ test patterns are needed to detect all single stuck-at-faults and both AND and OR bridging faults which are detectable.

7.4. Detection of Stuck-at-Faults in Mixed-Polarity Networks

Pradhan [105] has given a universal test set for multiple fault detection of mixed polarity AND/XOR networks when the inversion of inputs is produced internally using XOR gates. This universal test set which is independent of the function was shown to be of cardinality

$$6+2n+\sum_{e=0}^{j}\binom{n}{e} \tag{7}$$

where n is the number of variables of order j, the maximum number of literals contained in any product term in the AND/XOR expression. This universal test set is comprised of T_3, T_4, and T_5, where

$$a_0 x_1 x_2 x_3 \cdots x_n$$

$$T_{31} = \begin{Bmatrix} 1\ 0\ 0\ 0\ \ldots\ 0 \\ 0\ 1\ 1\ 1\ \ldots\ 1 \end{Bmatrix}$$

$$T_{32} = \begin{Bmatrix} 1\ 1\ 0\ 0\ \ldots\ 0 \\ 1\ 0\ 1\ 0\ \ldots\ 0 \\ 1\ 0\ 0\ 1\ \ldots\ 0 \\ \ldots\ldots\ldots\ 0 \\ \ldots\ldots\ldots \\ \ldots\ldots\ldots \\ 1\ 0\ 0\ 1\ \ldots\ 1 \end{Bmatrix}$$

$$T_{33}=\left\{\begin{array}{l}\\ 0\ 0\ 1\ 1\ \ldots\ 1\\ 0\ 1\ 0\ 1\ \ldots\ 1\\ 0\ 1\ 1\ 0\ \ldots\ 1\\ \ldots\ldots\ldots\\ \ldots\ldots\ldots\\ \ldots\ldots\ldots\\ 0\ 1\ 1\ 1\ \ldots\ 0\\ \\ \end{array}\right\}$$

$$T_4=\left\{\begin{array}{l}\\ 0\ 0\ 0\ 0\ \ldots\ 0\\ 1\ 0\ 0\ 0\ \ldots\ 0\\ 0\ 1\ 1\ 1\ \ldots\ 1\\ 1\ 0\ 0\ 0\ \ldots\ 0\\ \\ \end{array}\right\}$$

and $T_5=T_2{}^j$, where $T_2{}^j$ is the set of n-vectors with the number of 1's in the vectors being less than or equal to j.

7.5. Detection of Stuck-at and Bridging Faults in CRMP Networks

The Canonical Restricted Mixed Polarity AND/XOR networks are the largest class of AND/XOR canonical networks where a product term of certain literals occurs only once in the network. Moreover, a *CRMP* network can be decomposed into its component *CGRM* networks. Each *CGRM* network can be then examined with its universal test sets for detection of stuck-at and bridging faults using the procedures described in previous sections. In this way, it is possible to devise a method for the case of *CRMP* networks which can result in a reduced number of tests as compared to the result obtained by Pradhan in the general case.

In chapter 4, it was mentioned that in *CRMP* form, certain terms include literals of the same polarity and hence can be grouped together as component *CGRM* forms of that *CRMP*. For r component *CGRM* networks, the number of tests required to detect stuck-at and bridging faults can be r times the number of test sets for each component, provided that there are r observable outputs corresponding to the component *CGRMs*. However, as some of the variables occur in different polarities in different component

CGRM networks, there will be some tests which would occur more than once. There will also exist tests which are termed *compatible*. By combining the compatible tests and applying the repeated tests only once, it is possible to reduce the overall number of tests.

Definition two tests are *compatible* if all corresponding bits in the two test vectors are compatible. Two test bits are *compatible* if one of them is don't care or they are both the same.

Example 7.2. 10-0100 and 1-0-1-0 are compatible while 10-0100 and 1010000 are not. □

With identification of the compatible tests, it is possible to construct a compatibility graph. In this graph the nodes represent the tests and the nodes that are compatible will be adjacent. The problem of reducing the number of tests for a *CRMP* network can then be formulated as follows:

- Decompose the *CRMP* network into its component *CGRM* networks.
- Generate the test sets for each of the component *CGRM* networks.
- Remove the repeated occurrences of the same tests.
- Construct the compatibility graph of the tests.
- Find the disjoint covering of the graph with maximum cliques.

Once the covering is chosen, a single test is created for each group of compatible tests by combining them (0 and d gives 0, 1 and d gives 1). As an example, the test used for the two compatible tests d10 and 1dd will be 110.

The test reduction method will be shown by Example 7.3 below:

Example 7.3. Let a *CRMP* network be represented by $1 \oplus \bar{x}_1x_2 \oplus x_1\bar{x}_4 \oplus x_1x_2x_3 \oplus \bar{x}_2x_3\bar{x}_4 \oplus \bar{x}_1x_2x_3x_4$. This network can clearly be represented by 3 component *CGRM* networks given by: $C_1 = 1 \oplus \bar{x}_1x_2 \oplus \bar{x}_1x_2x_3x_4$, $C_2 = x_1\bar{x}_4 \oplus \bar{x}_2x_3\bar{x}_4$, and $C_3 = x_1x_2x_3$. The test sets for each of the component *CGRM* networks are then:

$$C_1:\begin{Bmatrix} 0\,1\,0\,0\,0 \\ 0\,0\,1\,1\,1 \\ 1\,1\,0\,0\,0 \\ 1\,0\,1\,1\,1 \\ d\,1\,1\,1\,1 \\ d\,0\,0\,1\,1 \\ d\,0\,1\,0\,1 \\ d\,0\,1\,1\,0 \end{Bmatrix} \quad C_2:\begin{Bmatrix} d\,0\,1\,0\,1 \\ d\,1\,0\,1\,0 \\ d\,0\,0\,1\,0 \\ d\,1\,1\,1\,0 \\ d\,1\,0\,0\,0 \\ d\,1\,0\,1\,1 \end{Bmatrix} \quad C_3:\begin{Bmatrix} d\,0\,0\,0\,d \\ d\,1\,1\,1\,d \\ d\,0\,1\,1\,d \\ d\,1\,0\,1\,d \\ d\,1\,1\,0\,d \\ d\,1\,1\,1\,d \end{Bmatrix}$$

It can be seen that d111d in C_3 occurs twice and one of them will be used. It is the same case with d0101 in C_1 and the one in C_2. The compatible tests are {01000, d1000}, {00111, d011d}, {11000, d1000}, {10111, d011d}, {d1111, d111d}, {d0110, d011d}, {d1010, d101d}, {d1110, d111d}, and {d1011, d101d} while d0011, d0101, d0010, and d110d are not compatible with any other tests. Now, a minimal number of non-redundant tests can be devised where only one test will be created from each set of compatible tests. □

As it can be seen, in this method the test sets would no longer be universal: however, depending on the number of component *CGRM* networks, the variables present and their polarities, it may be possible to reduce the number of tests drastically. The merits of the method vary with the type of the *CRMP* network and will not always yield the same gains. It also requires as many observable points as the number of component *CGRM* networks.

7.6. Detection of Stuck-at-Faults in Reed-Muller Trees

The basic tree structure realizing the Reed-Muller trees is shown in Figure 7.3. In this structure, the inputs to any AND gate is one of the input variables and the output of one XOR gate from a previous level. The inputs to an XOR gate is either an AND gate or another XOR gate. The tree expansion terminates with 0 or 1 at certain branches. In this case, the constants will be the input to one of the XOR gates and that branch will stop from expanding.

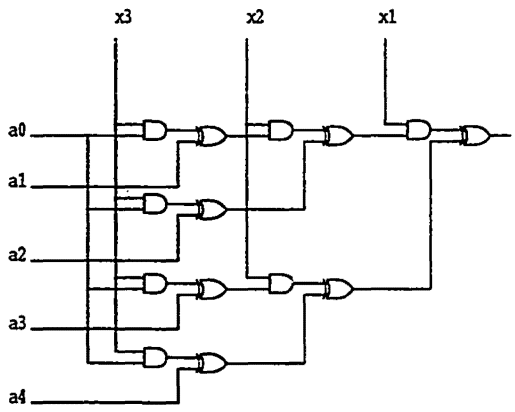

Figure 7.3 The Reed-Muller Tree for a 3-variable function

Let us observe that applying 1s to the input variables will result in all the AND gates to become transparent; i.e. the tree will be one of XOR-tree. It has been shown [61] that four tests are necessary and sufficient to detect all single stuck-at-faults of an XOR-tree. Moreover, it has been shown [135] that at most $\lfloor 3n/2 \rfloor + 1$ tests would detect all multiple-stuck-at faults of XOR-trees. For detecting stuck-at-faults of the AND gates, it can be seen that applying 0s for all the constant inputs to the XOR gates and 1s for all input variables and constant inputs to the AND gates will detect stuck-at-0s at the AND input gates. Detecting stuck-at-1 faults at the inputs to AND gates requires a set of $n + 1$ tests shown below:

$$
\begin{array}{c}
\begin{matrix} a_1 & \overbrace{\cdots\ a_m\ a_p\ x_1 x_2 \cdots}^{} & x_n \end{matrix} \\
T_\tau = \left\{ \begin{array}{l}
1\ .\ .\ 0\ 1\ 1\ .\ .\ 1 \\
1\ .\ .\ 1\ 0\ 1\ .\ .\ 1 \\
1\ .\ .\ 1\ 1\ 0\ .\ .\ 1 \\
1\ .\ .\ 1\ 1\ 1\ .\ .\ 1 \\
.\ .\ .\ .\ .\ .\ .\ .\ . \\
.\ .\ .\ .\ .\ .\ .\ .\ . \\
1\ .\ .\ 1\ 1\ 1\ .\ .\ 0
\end{array} \right.
\end{array}
$$

where *a 1* through *a m* represent the constant inputs to the XOR gates and *a p* represents the constant to the last level AND gate.

Similar to the *CGRM* forms, for the case of *CGRM* trees, there are some input variables that will be in negative polarities but the structure is similar to the Reed-Muller tree. Here, for those variables that occur in negative polarity, the tests need to be the bit-wise opposite to the corresponding test bit in the Reed-Muller tree. If the inversions are to occur inside the chip, an extra AND gate with observable output having all the negated inputs as its input would be required.

7.7. Summary

In this chapter, the testability of AND/XOR forms was reiterated and extended for certain of these forms. Specifically, it was shown for the first time that the universal test vector for detection of stuck-at-faults and bridging faults in Reed-Muller cascades can be directly used in fixed polarity forms with slight modification. In addition, a scheme for identification of a minimal test vector for Generalized AND/XOR forms was introduced. While investigation of the testability of multi-level XOR forms is new, a step in this direction was introduced by this author for the case of Reed-Muller trees.

Chapter 8

Conclusion

This dissertation introduced new concepts in synthesis and mapping for CA-type FPGAs based on XOR logic which have applications for logic synthesis in general. While the synthesis and mapping to LUT-Type and row based FPGAs have attracted considerable attention, the CA-Type FPGAs have been essentially lacking adequate synthesis methods. This is in contrast to the increasing attention the CA-Type FPGAs have received as major FPGA architectures. In this dissertation, the author developed several new methodologies for synthesis and mapping to these fine-grained FPGAs with mostly local interconnections.

As mentioned earlier, the main characteristics of the CA-type FPGAs is their fine granularity and their emphasis in local communications. The logic blocks in most of these architectures are capable of realizing a large number of functions of two - or three inputs. Therefore, the architecture supports a much wider array of logic operations than the simple AND/OR logic. Furthermore, the local communications among the cells create restrictions on the placement and routing which need to be considered. In this dissertation, XOR logic and the concept of regularity were used by this author to address the fine granularity and local interconnections of these type of FPGAs.

XOR logic has long been known to be, in general, more compact than AND/OR and possess high design for testability properties. This knowledge was extended in this dissertation to reaffirm the compactness of the XOR logic on new synthesis methods and provide new synthesis tools to make the utilization of this logic more practical.

This author introduced a new concept of Universal XOR Forms which provides the framework for investigation of all possible XOR canonical representations of the Boolean functions. Up to now, only AND/XOR representations of functions were known. It was long known that the set of -variable Boolean functions under addition mod-2 forms a space over the Galois field of two elements, GF(2). It was also known that different AND/XOR canonical forms form bases in this vector space. However, this study had remained confined to AND/XOR bases only. In this dissertation, this approach was taken to its logical conclusion and was extended to all possible XOR canonical forms including various AND/OR/XOR canonical forms.

In terms of the logic blocks in CA-Type FPGAs and many architectures in general, AND, OR, and XOR are more available than other gates. Hence, among all UXF, various AND/OR/XOR canonical forms were more concentrated on. The author has shown ity and Generalized AND/XOR canonical forms.

A fast method for realization of functions in fixed polarity AND/XOR canonical form was introduced. These forms are the basis for many other forms and are the most easily testable of all forms. The method utilized special characteristics of functions given as an array of disjoint cubes in order to identify a minimal polarity. Both exhaustive and heuristic methods were presented. Various operations and techniques in this chapter were presented which are new and are all solely based on the works of this author.

In addition, the larger class of Generalized AND/XOR forms were presented and the utilization of the fixed polarity forms in minimization of functions in these forms were shown. The Generalized AND/XOR forms comprise a large class of forms and their testability is of smaller cardinality than the ESOP representations of functions. This author has contributed to the minimization methodology and the test generation where the testing properties are totally new.

While the Boolean techniques for identification of a minimal UXF representation of a function could result in more compact representation, these techniques are usually very slow. For this purpose, algebraic methods were also investigated. Based on the former work of Song [143], this author introduced an algorithm which identifies a minimal restricted factored representation of the functions. The generalizations to the complex terms introduced in [143] are also new here. Next, it was shown experimentally by this author that through the factored forms, it is possible to devise a multi-level AND/OR/XOR representation of functions which is more compact than ESOPs, which are generally themselves more compact than SOPs. The advantage for CA-Type FPGAs is that these factored forms would directly map to them while general factorization will be difficult to rout.

All of the above synthesis methods fit within the framework of Complex Maitra Logic Arrays. In this approach, which is a generalization of PLA and XPLAs, the two stages of logic synthesis and physical design are combined alleviating the routing stage which is of problem in CA-Type FPGAs. The CMLA was presented as an OR or XOR of complex terms which include AND, OR, and XOR of literals. With limited interconnection among the cells, this approach, while being multi-level, provides a direct mapping strategy which is the main problem with CA-Type FPGAs. This author has also contributed to the concept of CMLA, and in particular, he is the author of a general folding technique which results in further compaction of the mapping. Experimental results of the algorithms developed by this author illuminated the advantage of these techniques.

A different regular structure-based synthesis method was that of the Ordered Kronecker Functional Decision Diagrams. In this dissertation the concept of the Kronecker Functional Decision Diagrams was presented as a generalization of the Binary Decision Diagrams and Functional Decision Diagrams. The main contribution of this author was the application of the dynamic variable ordering with sifting, introduced by

Rudell, to the KFDDs. I have also contributed to various topological heuristic techniques for the identification of a minimal KFDD only partly discussed here and partly in [117]. The latter reference first popularized these decision diagrams in the research community. By using the sifting technique, it was possible to evaluate the compactness of this representation as compared to BDDs. While, compactness of KFDDs was known intuitively, the theoretical work by Becker et al. [41] has illuminated their advantages. They showed that there exists a class of functions for which BDDs are exponential while FDDs are polynomial and vice versa. Hence, using KFDDs as a generalization of both of these is more advantageous than using only one. Furthermore, they showed that there exists a class of functions for which both BDDs and FDDs are exponential while KFDDs are polynomial.

For, the KFDDs to have practical use, it is important to be of ease of construction and manipulation. While the iterative generation of the Davio nodes was independently developed by the author and the colleagues at J. W. Goethe University, the generation of the package is due to these colleagues. Utilizing the dynamic variable ordering with sifting, it was possible to experimentally substantiate the advantages of the KFDDs. For large benchmark functions, it was shown that on average KFDDs are 35% more compact than BDDs with reductions of up to 75% being observed.

As KFDDs are canonical, easy to construct and manipulate and more compact than BDDs, they can potentially be of major value in logic synthesis in general. As there exist hard functions that have not been able to be represented by BDDs, it could be of interest to evaluate KFDDs whether they are able to represent these functions. In addition, it is possible to develop more efficient applications in the areas where BDDs have been generally used. These include areas in synthesis, verification, modeling, testing, etc.

The MUX and AND/XOR nodes of KFDDs are available in most CA-Type FPGAs. Hence, it is possible to use KFDDs for synthesis and utilize graph embedding

techniques for mapping to these FPGAs. In this dissertation, the synthesis aspect was emphasized on.

The logic synthesis techniques also possess high testability properties. These properties were illuminated in the dissertation by introducing testability schemes for some of these forms. Specifically, it was shown for the first time in this dissertation that the universal test vector for detection of stuck-at-faults and bridging faults in Reed-Muller cascades can be directly used in fixed polarity forms with slight modification. In addition, a scheme for identification of a minimal test vector for Generalized AND/XOR forms was introduced. While investigation of the testability of multi-level XOR forms is new, a step in this direction was introduced for the case of Reed-Muller trees. This result by the author is among the very few recent studies on testability of multi-level AND/XOR networks.

The methodologies presented in this dissertation can be still extended further, however, they provide new frameworks which can contribute to the investigation of CA-Type FPGA synthesis and synthesis with XOR. It can be claimed that utilization of XORs has gained much more credibility already from the time that the work on this dissertation started. This can be observed from the new groups that work in this area as well as the organizing of the first workshop devoted to this logic. It can further be claimed that the work of our group and this author as part of the group has had some impact in the popularization of the concepts of XOR synthesis. This dissertation is just illuminating the areas which are of high research potential in future.

In summary, this dissertation provided general methodologies for CA-Type FPGA synthesis which until now were less than adequate. As mentioned in the introduction, the available methods with one exception of the lexicographical ordering method were very inefficient and could not handle general purpose functions. The methodologies introduced here have addressed this problem. In addition, as a result of this research, general

synthesis tools and concepts were developed which are of value to research community at large. Kronecker Functional Decision Diagrams, the concept of UXFs, and the approaches presented for the minimization of Boolean functions in fixed polarity AND/XOR canonical forms can be considered the major contributions by this author.

References

1. S. B. Akers, "On a Theory of Boolean Functions," *J. of SIAM*, vol. 7, pp. 487-498, 1959.

2. S. B. Akers, "A Rectangular Logic Array," *IEEE Trans. on Comput.*, vol. C-21, No. 8, pp. 848-857, 1972.

3. S. B. Akers, "Binary Decision Diagrams," *IEEE Trans. on Comput.*, vol. 27, pp. 509-516, 1978.

4. E. Artin, *Geometric Algebra*, Interscience Publishers, Inc., 1957.

5. A. Aziz, F. Balarin, R. K. Brayton, S. Cheng, R. Hojati, S. C. Krishnan, R. K. Ranjan, A. Sangiovanni-Vincentelli, and T. R. Shiple, "HSIS: A BDD-Based Environment for Formal Verification," *Proc. 31st ACM/IEEE Design Automation Conf.*, 1994.

6. P. Ashar, A. Ghosh, S. Devadas and A. R. Newton, "Combinational and Sequential Logic Verification Using General Binary Decision Diagrams," *IEEE Int. Workshop on Logic Synthesis*, 1991.

7. ATMEL Corporation, CMOS Integrated Circuit Data Book, 1994.

8. B. Becker, R. Drechsler, and R. Werchner, "On the Relation Between BDDs and FDDs," *Technical report, Universität Frankfurt, Fachbereich Informatik*, 1993.

9. B. Becker, R. Drechsler, and M. Theobald, "On the Implementation of a Package for Efficient Representation and Manipulation of Functional Decision Diagrams," *Proc. of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pp. 162-169, Hamburg, Germany, Sept. 1993.

10. J. Bentley and H. T. Kung, "A Tree Machine for Searching Problems," *Proc. IEEE Int. Conf. Parallel Processing*, pp. 257-266, 1979.

11. J. Bern, J. Gregov, C. Meinel, and A. Slobodova, "Boolean Manipulation with Free BDDs - First Experimental Results," *Proc. IEEE Europ. Conf. on Design Automation*, 1994.

12. Ph. W. Besslich and M. W. Riege, "An efficient Program for Logic Synthesis of Mod-2 Sum Expressions," *Proc. of IEEE EUROASIC*, pp. 136-141, 1991.

13. Ph. W. Besslich and E. A. Trachtenberg, *A Three-Valued Quasi-Linear Transformation for Logic Synthesis*, C. Muroga and R. Creutzburg (eds.), Spectral Techniques: Theory and Applications, Elsevier, North Holland, 1992.

14. T. Besson, H. Bousouzou, M. Crastes, I. Floricica, and G. Saucier, "Synthesis of Multiplexer-based FPGA Using BDD," *Proc. IEEE Int. Conf. on Comput. Design*,

pp. 163-167, October 1992.

15. S. N. Bhatt and C. E. Leiserson, "How to Assemble Tree Machines ," *Proc. 14th ACM. Symp. Theory Comput.*, pp. 77-84, 1982.

16. 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," *Proc. IEE Pt. E.*, vol. 132, No. 3, pp. 155-161, 1985.

17. G. Bioul, and M. Davio, "Taylor Expansions of Boolean Functions and of their Derivatives," *Philips Res. Rep.*, vol. 27, No. 1, pp. 1-6, 1972.

18. D. Bostick, G. Hachtel, R. Jacoby, M. R. Lightner, P. Moceyunas, C. R. Morrison, and D. Ravenscroft, "The Boulder Optimal Logic Design System," *Proc. IEEE Int. Conf. on CAD*, pp. 62-65, 1987.

19. K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," *Proc. of the 27th ACM/IEEE Design Automation Conf.*, pp. 40-45, 1990.

20. K. S. Brand and T. Sasao, "Minimization of AND-EXOR Expressions Using Rewrite Rules," *IEEE Trans. on Comput.*, vol. C-42, No. 5, pp. 568-576, 1993.

21. R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

22. R. K. Brayton, R. L. 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, 1989.

23. R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Comput.*, vol. C-35, No. 8, pp. 667-691, 1986.

24. R. E. Bryant, "On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication," *IEEE Trans. on Comput.*, vol. C-40, pp. 205-213, 1991.

25. R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Comput. Surveys*, vol. 24, pp. 293-318, 1992.

26. K. M. Burch, "Multiplier Verification Using BDDs," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 408-412, 1991.

27. 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," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 417-420, 1991.

28. A. Calazans, Q. Zhang, R. Jacobi, B. Yernaux, and A. M. Trullemans, "Advanced Ordering and Manipulation Techniques for Binary Decision Diagrams," *Proc. IEEE Europ. Conf. on Design Automation*, pp. 452-457, 1992.

29. P. Calingaert, "Switching Function Canonical Forms Based on Commutative and Associative Binary Operations," *Trans. of AIEE*, vol. 80, pp. 217-224, 1961.

30. M. Cohn, "Inconsistent Canonical Forms of Switching Functions," *IRE Trans. on Elec. Comput.*, vol. EC-11, pp. 284, 1962.

31. Concurrent Logic, Inc., A Seminar at Portland State University, Portland Oregon, Nov. 1992.

32. M. Crastes, K. Sakouti, and G. Saucier, "A Technology Mapping Method Based on Perfect and Semi-Perfect Matchings," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 93-98, San Francisco, CA, June 1991.

33. L. Csanky, *On the Generalized Reed-Muller Canonical Form of Boolean Functions*, M.S. Thesis, University of California, Berkeley, 1972.

34. L. Csanky, M. A. Perkowski, and I. Schäfer, "Canonical Restricted Mixed Polarity Exclusive-OR Sums of Products and the Efficient Algorithm for their Minimization," *Proc. IEE Pt. E.*, vol. 140, No. 1, pp. 69-77, 1993.

35. J. Darringer, D. Brand, J. Gerby, W. Joyner, and L. Trevillyan, "LSS: A System for Production Logic Synthesis," *IBM J. Res. Develop.*, pp. 537-545, 1984.

36. T. Damarla and M. Karpovksy, "Detection of Stuck-at and Bridging Faults in Reed-Muller Canonical (RMC) Networks," *Proc. IEE Pt. E.*, vol. 136, No. 5, pp. 430-433, 1989.

37. M. Davio, "Ring-Sum Expansions of Boolean Functions," *Proceedings of the Symposium on Computers and Automata*, pp. 411-418, Polytechnic Institute of Brooklyn, 1971.

38. M. Davio, J. P. Deschamps, and A. Thayse, *Discrete and Switching Functions*, McGraw-Hill, 1978.

39. R. Drechsler, M. Theobald, and B. Becker, "Fast FDD Based Minimization of Generalized Reed-Muller Forms," *Personal Communication*, 1993.

40. R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski, "Efficient Representation and Manipulation of Switching Functions Based on Ordered Kronecker Functional Decision Diagrams," *Proc. 31st ACM/IEEE Design Automation Conf.*, 1994.

41. R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski, "On the Computational Power of Ordered Kronecker Functional Decision Diagrams," *Submitted to IEEE Int. Conf. on CAD*, 1994.

42. V. Dvorak, "A Two-Rail Cascade Synthesis of Boolean Functions," *IEEE Trans. on Comput.*, vol. C-17, No. 6, pp. 592-596, 1968.

43. B. Elspas, *The Theory of Multirail Cascades*, pp. 315-367, Recent Developments in Switching Theory, Ed. A. Mukhopadhyay, Academic Press, 1971.

44. B. J. Falkowski, I. Schäfer, and M. A. Perkowski, "Fast Computer Algorithm for the Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions," *IEEE 33rd Midwest Symp. on Circuits & Systems*, 1990.

45. B. J. Falkowski, *Spectral Methods for Boolean and Multiple-Valued Input Logic Functions*, PhD Dissertation, Portland State University, Portland Oregon, 1991.

46. L. T. Fisher, "Unateness Properties of AND-Exclusive-OR Logic Circuits," *IEEE Trans. on Comput.*, vol. C-23, No. 2, pp. 166-172, 1974.

47. R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 227-233, San Francisco, CA, June 1991.

48. S. J. Friedman and K. J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," *Proc. of the 24th ACM/IEEE Design Automation Conf.*, pp. 348-356, 1987.

49. S. J. Friedman, *Efficient Data Structures for Boolean Function Representation*, PhD Dissertation, Dept. of Comput. Sciences, Princeton University, 1990.

50. Froessl, J. and Eschermann, B., "Module Generation for AND/XOR-Fields (XPLAs)," *Proc. of Int Conf. on Comput. Design*, pp. 26-29, 1991.

51. M. Fujita, Y. Matsunga and T. Kakuda, "On Variable Ordering of Binary Decision Diagrams for the Application of Multi-Level Synthesis," *Proc. IEEE Europ. Conf. on Design Automation*, pp. 50-54, 1991.

52. H. Fujiwara, *Logic Testing and Design for Testability*, MIT Press, 1985.

53. D. Gordon, I. Koren, and G. M. Silberman, "Embedding Tree Structures in VLSI Hexagonal Arrays ," *IEEE Trans. on Comput.*, vol. C-33, pp. 104-107, 1984.

54. D. Gordon, "Efficient Embeddings of Binary Trees in VLSI Arrays ," *IEEE Trans. on Comput.*, vol. C-36, pp. 1009-1018, 1987.

55. D. Green and P. W. Foulk, "Adaptive Logic Trees for Use in Multilevel-Circuit Design," *Electron. Letters*, vol. 5, pp. 83-84, 1969.

56. D. Green, "Reed-Muller Expansions of Incompletely Specified Functions," *Proc. IEE Pt. E.*, vol. 134, No. 5, pp. 228-236, 1987.

57. D. Green, "Reed-Muller Canonical Forms With Mixed Polarity and Their Manipulations," *Proc. IEE Pt. E.*, vol. 137, No. 1, pp. 103-113, 1990.

58. D. Green, "Families of Reed-Muller canonical forms," *Int. J. Elect.*, pp. 259-280, Feb. 1991.

59. D. Gregory, K. Bartlett, A. de Geus and G. Hachtel, "Socrates: A System for Automatically Synthesizing and Optimizing Combinational Logic," *Proc. of the 23rd ACM/IEEE Design Automation Conf.*, pp. 79-85, 1986.

60. J. Gregov and C. Meinel, *Efficient Analysis and Manipulation of OBDDs Can be Extended to Read-Once-Only Branching Programs*, WG'92, LNCS, 1992.

61. Hayes, J. P., "On Realization of Boolean Functions Requiring a Minimal or Near-Minimal Number of Tests," *IEEE Trans. on Comput.*, vol. C-20, No. 12, pp. 1506-1513, 1971.

62. L. Hellerman, "A Measure of Computational Work," *IEEE Trans. on Comput.*, vol. C-21, 1972.

63. M. Helliwell and M. A. Perkowski, "A Fast Algorithm to Minimize Multi-Output Mixed-Polarity Generalized Reed-Muller Forms," *Proc. of the 25th ACM/IEEE Design Automation Conf.*, pp. 427-432, 1988.

64. E. Horowitz and A. Zorat, "The Binary Tree as an Interconnection Network: Applications to Multiprocessor Systems and VLSI," *IEEE Trans. on Comput.*, vol. C-30, pp. 247-253, 1981.

65. S. L. Hurst, *The Logical Processing of Digital Signals*, pp. 364-391, Crane Russak, 1978.

66. S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic*, Academic Press, 1985.

67. S. J. Hong, R. G. Cain, and D. L. Ostapko, "Mini: A Heuristic Approach for Logic Minimization," *IBM J. Res. Develop.*, pp. 443-458, 1974.

68. N. Ishiura, H. Sawada, and S. Yajima, "Minimization of Binary Decision Diagrams Based on Exchanges of Variables," *Proc. IEEE Int. Conf. on CAD*, pp. 472-475, 1991.

69. J. Jain, M. Abadir, J. Bitner, D. S. Fussell and J. A. Abraham, "An Efficient Functional Representation for Digital Circuits," *Proc. IEEE Europ. Conf. on Design Automation*, pp. 440-446, 1992.

70. S. Jeong, B. Plessier, G. Hachtel, D. S. Fussell, and F. Somenzi, "Extended BDDs: Trading off Canonicity for Structure in Verification Algorithms," *Proc. IEEE Int. Conf. on CAD*, pp. 464-467, 1991.

71. S. Jeong, B. Plessier, G. Hachtel, D. S. Fussell and F. Somenzi, "Variable Ordering for Binary Decision Diagrams," *Proc. of IEEE Europ. Design Automation Conf.*, pp. 447-451, 1992.

72. U. Kebcshull, E. Schubert, and W. Rosenstiel, "Multilevel Logic Synthesis Based on Functional Decision Diagrams," *Proc. of IEEE Europ. Design Automation Conf.*, pp. 43-47, 1992.

73. G. J. Klir, *An Approach to General Systems Theory,* pp. 281-285, Van Nostrand Reinhold Company, 1969.

74. K. L. Kodandapani, "A Note on Easily Testable Realizations for Logic Functions," *IEEE Trans. on Comput.*, vol. C-23, pp. 332-333, 1974.

75. K. L. Kodandapani, and R. V. Setlur, "A Note on Minimal Reed-Muller Canonical Forms of Switching Functions," *IEEE Trans. on Comput.*, vol. C-26, pp. 310-313, 1977.

76. S. N. Kukreja and I. Chen, "Combinational and Sequential Cellular Structures," *IEEE Trans. on Comput.*, vol. C-22, No. 9, pp. 813-823, 1973.

77. C. Y. Lee, "Representation of Switching Circuits by Binary-Decision Programs," *Bell System Technical J.*, vol. 38, pp. 985-999, 1959.

78. M. K. K. Maitra, "Cascaded Switching Networks of Two-Input Flexible Cells," *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 136-143, 1962.

79. S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment," *Proc. of Int. Conf. on CAD*, pp. 6-9, 1988.

80. S. B. Marinkovic, and Z. Tosic, "Algorithm for Minimal Polarized Polynomial Form Determination," *IEEE Trans. on Comput.*, vol. C-23, No. 12, pp. 1313-1315, 1974.

81. S. Minato, N. Ishiura, and S. Yajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," *Proc. of the 27th ACM/IEEE Design Automation Conf.*, pp. 52-57, 1990.

82. S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," *Proc. of the 30th ACM/IEEE Design Automation Conf.*, pp. 272-277, 1993.

83. R. C. Minnick, "Cutpoint Cellular Logic," *IEEE Trans. on Electron. Comput.*, vol. EC-13, pp. 685-698, 1964.

84. R. C. Minnick, "Cobweb Cellular Arrays," *AFIPS Conf. Proc.*, 1965.

85. R. C. Minnick, "A Survey of Microcellular Research," *J. of ACM*, vol. 14, No. 2, pp. 203-241, 1967.

86. A. Mukhopadhyay, "Unate Cellular Logic," *IEEE Trans. on Comput.*, vol. C-18, No. 2, pp. 114-121, 1969.

87. A. Mukhopadhyay and G. Schmitz, "Minimization of Exclusive-OR and Logical Equivalence Switching Circuits," *IEEE Trans. on Comput.*, vol. C-19, pp. 132-140, February 1970.

88. A. Mukhopadhyay and H. S. Stone, *Cellular Logic*, pp. 281-285, Recent Developments in Switching Theory, Ed. A. Mukhopadhyay, Academic Press, 1971.

89. D. E. Muller, "Application of Boolean Algebra to Switching Circuit Design and to Error Detection," *IRE Trans. on Elec. Comput.*, vol. EC-3, pp. 6-12, 1954.

90. R. Murgai, N. Shenoy, K. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *Proc. IEEE Int. Conf. on CAD*, pp. 564-567, 1991.

91. R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "An Improved Synthesis Algorithm for Multiplexor-based FPGAs," *29th ACM/IEEE Design Automation Conf.*, pp. 380-386, June, 1992.

92. S. Muroga, Y. Kambayashi, H. C. Lai and J. N. Culliney, "The Transduction Method - Design of Logic Networks Based on Permissible Functions," *IEEE Trans. on Comput.*, vol. C-38, pp. 1404-1424, 1989.

93. L. B. Nguyen, M. A. Perkowski and N. B. Goldstein, "PALMINI - Fast Boolean Minimizer for Personal Computers," *24th ACM/IEEE Design Automation Conf.*, pp. 615-621, 1987.

94. E. W. Page, "Minimally Testable Reed-Muller Canonical Forms," *IEEE Trans. on Comput.*, vol. C-29, No. 8, pp. 746-750, 1980.

95. G. K. Papakonstantinou, "A Synthesis Method for Cutpoint Cellular Arrays," *IEEE Trans. on Comput.*, vol. EC-21, No. 12, pp. 1286-1292, 1972.

96. M. S. Paterson, "On Razborov's Result for Bounded Depth Circuits over $\{\oplus, ^{=}\}$," *Technical report, University Warwick*, Warwick, 1986.

97. M. A. Perkowski, M. Driscoll, J. Liu, D. Smith, J. Brown, L. Yang, A. Shamsapour, M. Helliwell, B. Falkowski, P. Wu, M. Ciesielski, and A. Sarabi, "Integration of Logic Synthesis and High-Level Synthesis into the DIADES Design Automation System," *Proc. of IEEE Int. Symp. on Circuits and Systems* , pp. 748-751, Portland, OR, May 1989.

98. M. A. Perkowski, A. Sarabi, and I. Schäfer, "Application of Orthogonal Transforms in Image Processing," *Proc. of Northcon* , pp. 307-309, Portland, OR, October 1991.

99. M. A. Perkowski, and P. D. Johnson, "Canonical Multi-Valued Input Reed-Muller Trees and Forms," *Proc. 3rd NASA Symposium on VLSI Design*, pp. 11.3.1-11.3.13, Moscow, Idaho, October 1991.

100. M. A. Perkowski, "The Generalized Orthonormal Expansion of Functions With Multiple-Valued Inputs and Some of its Applications," *Proceedings of the 22nd ISMVL*, pp. 442-450, 1992.

101. M. A. Perkowski, L. Csanky, A. Sarabi, and I. Schäfer, "Fast Minimization of Mixed-Polarity AND/XOR Canonical Networks," *Proc. IEEE Int. Conf. on Comput. Design*, pp. 33-36, 1992.

102. M. A. Perkowski, A. Sarabi, and F. R. Beyl, "XOR Canonical Forms of Switching Functions," *Proc. of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pp. 27-32, Hamburg, Germany, Sept. 1993.

103. M. A. Perkowski, I. Schäfer, A. Sarabi, and M. Chrzanowska-Jeske, "Multi-level Synthesis Based on Kronecker Decision Diagrams and Boolean Ternary Decision Diagrams for Incompletely Specified Functions," *Accepted for Publication in VLSI J.*, 1994.

104. J. Poswig, "Disjoint Decomposition of Boolean Functions," *Proc. IEE Pt. E.*, vol. 138, No. 1, pp. 48-56, 1993.

105. D. K. Pradhan, "Universal Test Sets for Multiple Fault Detection in AND-EXOR Arrays," *IEEE Trans. on Comput.*, vol. C-27, No. 2, pp. 181-187, 1978.

106. F. P. Preparata, "State-Logic Relations for Autonomous Sequential Networks," *IEEE Trans. on Elec. Comput.*, vol. EC-13, pp. 542-548, 1964.

107. S. Purwar, "An Efficient Method of Computing Generalized Reed-Muller Expansions from Binary Decision Diagram," *IEEE Trans. on Comput.*, vol. C-40, No. 11, pp. 1298-1301, 1991.

108. A. A. Razborov, "Lower Bounds on the Size of Bounded Depth Networks over the Basis $\{\oplus, \neg\}$," *Technical report, Moscow State University*, Moscow, 1986.

109. S. M. Reddy, "Easily Testable Realization for Logic Functions," *IEEE Trans. on Comput.*, vol. C-21, No. 11, pp. 1183-1188, 1972.

110. I. S. Reed, "A Class of Multiple-Error-Correcting Codes and Their Decoding Scheme," *IRE Trans. on Inf. Theory*, vol. PGIT-4, pp. 38-49, 1954.

111. R. L. Rudell, and A. Sangiovanni-Vincentelli, "Multiple-Valued Minimization for PLA Optimization," *IEEE Trans. on CAD*, pp. 727-750, 1987.

112. R. L. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *Proc. IEEE Int. Conf. on CAD*, pp. 42-47, 1993.

113. K. K. Saluja, *A Study of Combinational Networks Based on Reed-Muller Canonic Forms*, PhD Dissertation, University of Iowa, Iowa City, Iowa, 1973.

114. K. K. Saluja and S. M. Reddy, "Fault Detection Test Set for Reed-Muller Canonic Networks," *IEEE Trans. on Comput.*, vol. C-24, No. 10, pp. 995-998, 1975.

115. A. Sarabi and M. A. Perkowski, "Fast Exact and Quasi-Minimal Minimization of Highly Testable Fixed-Polarity AND/XOR Canonical Networks," *Proc. of the 29th ACM/IEEE Design Automation Conf.*, pp. 30-35, 1992.

116. A. Sarabi and M. A. Perkowski, "Cube Based Method for Optimal and Quasi-Optimal Minimization of Consistent Generalized Reed-Muller Expansions," *Submitted to IEEE Trans. on Comput.*, 1992.

117. A. Sarabi, P. F. Ho, K. Iravani, W. R. Daasch, and M. A. Perkowski, "Minimal Multi-level Realization of Switching Functions Based on Kronecker Functional Decision Diagrams," *IEEE Int. Workshop Logic Synthesis*, pp. P3-1, 1993.

118. A. Sarabi and M. A. Perkowski, "Design for Testability Properties of AND/XOR Networks," *Proc. of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pp. 147-153, Hamburg, Germany, Sept. 1993.

119. A. Sarabi, N. Song, M. Chrzanowska-Jeske, and M. A. Perkowski, "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays ," *Proc. 31st ACM/IEEE Design Automation Conf.*, 1994.

120. T. Sasao and Ph. Besslich, "On the Complexity of MOD-2 Sum PLAs," *IEEE Trans. on Comput.*, vol. C-39, No. 2, pp. 262-266, 1990.

121. T. Sasao and T. Amada, "A Design Method of AND-OR-EXOR circuits," *Int. Symp. on Logic Synthesis and Microprocessor Arch.*, 1992.

122. T. Sasao, "EXMIN 2: A Simplification Algorithm for Exclusive-Sum-Of-Products Expressions for Multiple-Valued-Input Two-Valued-Output Functions," *IEEE Trans. on CAD*, vol. 12, No. 5, pp. 621-632, 1993.

123. T. Sasao, *AND-EXOR Expressions and their Optimization, in Sasao(ed.), Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

124. N. Sasao, "Ternary Decision Diagrams and Their Application,," *IEEE Int. Workshop Logic Synthesis*, p. 6c, 1993.

125. G. Saucier, J. From, and P. Abouzeid, "Lexicographical Expressions of Boolean Functions with Application to Multilevel Synthesis," *IEEE Trans. on CAD*, vol. 12, No. 11, pp. 1642-1654, 1993.

126. J. M. Saul, "An Improved Algorithm for the Minimization of Mixed Polarity Reed-Muller Representation," *Proc. IEEE Int. Conf. on Comput. Design*, pp. 372-

375, Cambridge, MA, Oct. 1990.

127. J. M. Saul, "Towards a Mixed Exclusive/Inclusive OR Factored Form," *Proc. of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pp. 2-5, Hamburg, Germany, Sept. 1993.

128. I. Schäfer, *An Efficient Cube Comparison Method for Discrete Spectral Transformations of Logic Functions*, M.S. Thesis, Portland State University, Portland, OR, 1990.

129. I. Schäfer and M. A. Perkowski, "Multiple-Valued Input Generalized Reed-Muller Forms," *Proc. of IEEE Int. Symp. on Multi-Valued Logic*, pp. 40-48, 1991.

130. I. Schäfer, *Orthogonal and Nonorthogonal Expansions for Multi-Level Logic Synthesis for Nearly Linear Functions and their Application to Field Programmable Gate Array Mapping*, PhD Dissertation, Portland State University, Portland Oregon, 1992.

131. I. Schäfer, M. A. Perkowski, and H. Wu, "Multilevel Logic Synthesis for Cellular FPGAs Based on Orthogonal Expansions," *Proc. of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pp. 42-51, Hamburg, Germany, Sept. 1993.

132. E. Schubert, U. Kebcshull, and W. Rosenstiel, "FDD Based Technology Mapping for FPGA," *Proc. of IEEE EUROASIC*, pp. 14-18, 1992.

133. E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," *Proc. IEEE Int. Conf. on Comput. Design*, pp. 328-323, October 1992.

134. S. Seshu and F. E. Hohn, "Symmetric Polynomials in Boolean Algebras," *Harvard Comp. Lab. Annals*, vol. XXX, pp. 225-234, 1957.

135. S. C. Seth, K. L. Kodandapani, "Diagnosis of Faults in Linear Tree Networks," *IEEE Trans. on Comput.*, vol. C-26, No. 1, pp. 29-33, 1977.

136. C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans. AIEE*, vol. 57, pp. 713-723, 1938.

137. H. M. Sheffer, "A Set of Five Independent Postulates for Boolean Algebras, With Application to Logical Constants," *Trans. American Math. Society*, vol. 14, pp. 481-488, 1913.

138. A. Shen, S. Devadas, and A. Ghosh, "Probabilistic Construction and Manipulation of Free Boolean Diagrams," *Proc. IEEE Int. Conf. on CAD*, pp. 544-549, 1993.

139. R. A. Short, "Two-Rail Cellular Arrays," *AFIPS Conf. Proc.*, vol. 27, pt. 1, pp. 355-369, 1965.

140. A. Shukla, "A Set of Axioms for the Propositional Calculus with Implication and Converse Non-Implication," *Notre Dame J. of Formal Logic*, vol. V, pp. 123-128, 1965.

141. A. Shukla, "A Set of Axioms for the Propositional Calculus with Implication and Non-Equivalence," *Notre Dame J. of Formal Logic*, vol. VII, No. 3, pp. 281-286, 1966.

142. N. Song and M. A. Perkowski, "EXORCISM-MV-2: Minimization of Exclusive Sum of Products Expressions for Multiple-Valued Input Incompletely Specified Functions," *Proc. of IEEE Int. Symp. on Multi-Valued Logic*, pp. 132-137, 1993.

143. N. Song and M. A. Perkowski, "A Method for Logic Mapping for Fine Grain FPGAs," *IEEE Int. Workshop Logic Synthesis*, p. P9a, 1993.

144. N. Song, M. A. Perkowski, M. Chrzanowska-Jeske, and A. Sarabi, "A New Design Methodology for Two-Dimensional Logic Arrays," *Accepted for Publication in VLSI J.*, 1994.

145. M. H. Stone, "The Representation of Boolean Algebras," *Proc. Nat. Acad. Sci.*, vol. 22, pp. 37-111, 1936.

146. H. S. Stone and A. J. Korenjak, "Canonical Form and Synthesis of Cellular Cascades," *IRE Trans. Electron. Comput.*, vol. EC-14, No. 6, pp. 852-862, 1965.

147. S. Swamy, "On Generalized Reed-Muller Expansion," *IEEE Trans. on Comput.*, vol. C-21, pp. 1008-1009, 1972.

148. C. C. Tsai and M. Marek-Sadowska, "Efficient Minimization Algorithms for Fixed Polarity AND/XOR Canonical Networks," *Great Lake Symp. VLSI*, pp. 76-79, 1993.

149. D. Varma, and E. A. Trachtenberg, "Design Automation Tools for Efficient Implementation of Logic Functions by Decomposition," *Proc. IEEE Int. Conf. on CAD*, pp. 901-916, 1989.

150. W. Wan and M. A. Perkowski, "A New Approach to the Decomposition of Incompletely Specified Multi-Output Functions Based on Graph-Coloring and Local Transformations and its Application to FPGA Mapping," *Proc. European Design Automation Conf.*, pp. 293-298, Hamburg, Germany, Sept. 1992.

151. I. Wegener, "On the Complexity of Branching Programs and Decision Trees for Clique Functions," *J. of ACM*, vol. 35, No. 2, pp. 461-471, 1988.

152. C. D. Weiss, "Optimal Synthesis of Arbitrary Switching Functions with Regular Arrays of 2-input 1-output Switching Elements," *IEEE Trans. on Comput.*, vol. C-18, No. 9, pp. 839-856, 1969.

153. W. Wernick, "Complete Set of Logical Functions," *Trans. American Math. Society*, vol. 51, pp. 117-132, 1942.

154. S. G. Williamson, *Combinatorics for Computer Science*, Computer Science Press, Inc., 1985.

155. L. Wu and M. A. Perkowski, *Minimization of Permuted Reed-Muller Trees for Cellular Logic Programmable Gate Array*, pp. 78-87, H. Gruenbacher and R. Hartenstein (eds.), LNCS, No. 705, Springer Verlag, 1993.

156. S. V. Yablonskii, "Functional Constructions in K-valued Logic," *Trudy mathematichekovo instituta im. Steklova*, vol. 51, pp. 5-142, 1958.

157. M. Yoeli, "A Group-Theoretic Approach to Two-Rail Cascades," *IRE Trans. Electron. Comput.*, vol. EC-14, pp. 815-822, 1965.

158. I. L. Zhegalkin, "Arifmetizatsiya simbolicheskoĭ logiki (Arithmetization of Symbolic Logic)," *Matematicheskii Sbornik*, vol. 35, 36, pp. 311-373, 205-338, 1928, 1929.