Combinational Logic Synthesis Based on the Dual Form of Reed-Muller Representation

By

Khalid Faraj

B.Sc, M.Sc

A thesis presented in partial fulfilment
of the requirements for the degree of

Doctor of Philosophy

Napier University

School of Engineering

2005

CONTAINS DISKETTE

UNABLE TO COPY

CONTACT UNIVERSITY

IF YOU WISH TO SEE

THIS MATERIAL

Declaration

I declare that no portion of the work referred in this thesis has been submitted in support of an application of another degree, qualification or other academic awards of this or any other university or institution of learning.

Edinburgh, March 2005

Khalid Faraj

Acknowledgements

All praise is due to ALLAH who has been bestowing me with his great bounties and enabled me to complete my thesis.

I am grateful to my research supervisor, Professor A. E. A. Almaini, School of Engineering, Napier University. This thesis would not have been possible without the mentoring encouragement, friendship, constant guidance and weekly meetings.

I would like to thank my second supervisor, Mr. M. MacCallum, for his encouragement of this research.
I would like to thank Professor M. Tariq for his support and encouragement.
Thanks are due to other members of the digital group, Dr. X. Yinshui and Dr. B. Ali.

I would like to thank my parents Mohammad and Zahoa for their unending support. Finally, I would like to thank my wife, Suheir and my lovely children, Mohammad, Omar and Fatima, for their patience and support throughout my research work.

Contents

List of symbols and abbreviations

a	Coefficients of the Sum of Products
AND	AND gate
b	Reed-Muller coefficients
BDD	Binary Decision Diagram
c	Coefficients of the Dual Reed-Muller expressions
CAD	Computer-Aided Design
CPU	Central Processing Unit
d	Coefficients of the Product of Sums
DRM	Dual Reed-Muller
ESOP	Exclusive Sum of Products
EXOR	EXOR gate
FPDRM	Fixed Polarity Dual Reed-Muller
FPGAs	Field-Programmable Gate Arrays
FPRM	Fixed Polarity Reed-Muller
GF(2)	Galios field
GHz	Gigahertz
GRM	Generalize Reed-Muller
HDL	Hardware Description Language
K-M	Karnaugh Map
KRO	Kronecker
LEQ	Logical Equivalence
m	minterm
M	Maxterm
MCNC	Microelectronics Center North Carolina
n	Number of variables

NAND	NANd gate
nD	negative Davio
NOR	NOR gate
OBDD	Ordered Binary Decision Diagram
OR	OR gate
p	polarity number
pD	positive Davio
PLA	Programmable Logic Array
PNR	percentage of non-zero elements
PPRM	Positive Polarity Reed-Muller
PPDRM	Positive Polarity Dual Reed-Muller
PTR	Pointer
R (n)	Reed-Muller transform matrix
RAM	Random-Access Memory
RM	Reed-Muller
ROBDD	Reduced Ordered Binary Decision Diagram
ROM	Read –Only Memory
RTL	Register Transfer Language
S	Shannon
S_i	Sum terms for the Dual Reed-Muller expressions
SOP	Sum of Products
T	Truth vector for the Dual Reed-Muller expressions
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
VHSIC	Very High-Speed Integrated Circuit
VLSI	Very Large-Scale Integration
$\bullet$	AND gate
$\oplus$	EXOR
$\oplus\sum$	Sum of EXORs
$\odot$	XNOR

$\prod$	Products
$++$	Continuous sum operation
$\otimes$	Kronecker product
$*$	Multiplication
$\circ$	Matrix multiplication based on LEQ and OR

List of Figures

List of Tables

Abstract

In certain applications, AND/XOR (Reed-Muller), and OR/XNOR (Dual form of Reed-Muller) logic have shown some attractive advantages over the standard Sum of Products (SOP) and Product of Sums (POS). Bidirectional conversion algorithms between SOP and AND/XOR also between POS and OR/XNOR based on Sparse and partitioning techniques are presented for multiple output Boolean functions. The developed programs are tested for some benchmarks with up to 20 inputs and 40 outputs.

A new direct method is presented to calculate the coefficients of the Fixed Polarity Dual Reed-Muller (FPDRM) from the truth vector of the POS. Any Boolean function can be expressed by FPDRM forms. There are 2^n polarities for an n-variable function and the number of sum terms depends on these polarities. Finding the best polarity is costly interims of CPU time, in order to search for the best polarity which will lead to the minimum number of sums for a particular function. Therefore, an algorithm is developed to compute all the coefficients of the Fixed Polarity Dual Reed-Muller (FPDRM) with polarity p from any polarity q. This technique is used to find the best polarity of FPDRM among the 2^n fixed polarities. The algorithm is based on the Dual- polarity property and the Gray code strategy. Therefore, there is no need to start from POS form to find FPDRM coefficients for all the polarities. The proposed methods are efficient in terms of memory size and CPU time. A fast algorithm is developed and implemented in C language which can convert between POSs and FPDRMs. The program was tested for up to 23 variables. A modified version of the same program was used to find the best polarity. For up to 13 variables the CPU time was less than 42 seconds.

To search for the optimal polarity for large number of variables and to reduce the search time of finding the optimal polarity of the function, two new algorithms are developed and presented in this thesis. The first one is used to convert between POS and Positive Polarity Dual Reed-Muller (PPDRM) forms. The second algorithm will find the optimal fixed polarity for the FPDRM among the 2^n different polarities for large n-variable functions. The most popular minimization criterion of the FPDRM form is obtained by the exhaustive search of the entire polarity vector. A non-exhaustive method for FPDRM expansions is presented. The new algorithms are based on separation of the truth vector (**T**) of POSs around each variable x_i into two groups. Instead of generating all of the polarity sets and searching for the best polarity, this algorithm will find the optimal polarity using the separation and sparse techniques, which will lead to optimal polarity. Time efficiency and computing speed are thus achieved in this technique. The algorithms don't require a large size of memory and don't require a long CPU time. The two algorithms are implemented in C language and tested for some benchmark. The proposed methods are fast and efficient as shown in the experimental results and can be used for large number of variables.

Chapter 1

Introduction

Logic synthesis is the process of converting a high-level description of design into an optimized gate-level representation. Logic synthesis uses standard cell library which has simple cells, such as OR, AND, EXOR, XNOR, NOR, flip-flops, registers. Libraries generally include more complex functions such as multiplexers, adders, decoders, shift registers, and memory (ROM, RAM).

1.1 Logic Synthesis

The increasing complexity of chip designs and the continuous development of smaller size fabrication processes present new challenges to the existing tools. Future synthesis tools are required to handle millions of gates in a realistic time. Computer-Aided Design (CAD) tools became critical for design and verification of Very Large Scale Integrated (VLSI) digital circuits [1]. There was a need for a new standard language to describe digital circuits. Thus, Hardware Description

Language (HDL) came into existence, which is used to develop documents, simulate, and synthesize the design of electronics systems. Hardware Description Languages such as VHDL [2, 3] which stands for VHSIC (Very High Speed Integrated Circuits) Hardware Description Language and Verilog were accepted by academia and industry to describe hardware from the abstract behavioral to the gate level. Computer-aided techniques [4] have provided the enabling methodology to design efficiently and successfully VLSI for a wide range of applications such as processors, telecommunication, etc. A typical VLSI design flow is illustrated in Figure1.1. The key steps are high level synthesis, logic synthesis and optimization, and physical level [5].

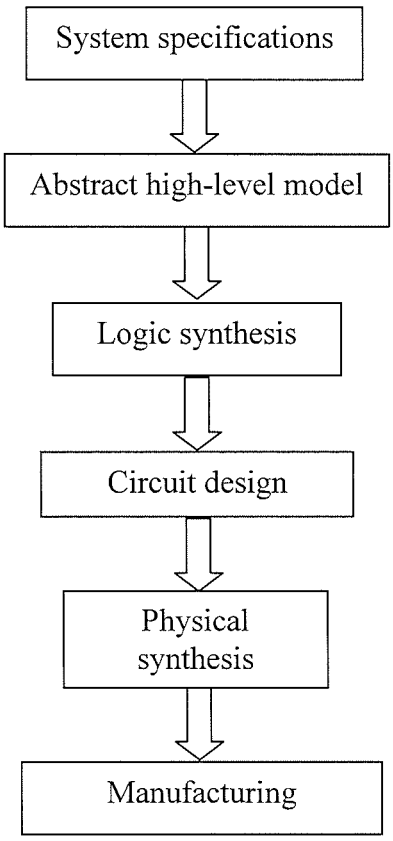


Figure 1.1: General overview of a circuit representation

The main objective of high level synthesis is to transform circuit specification details into a high-level description of the circuit structure, to define major functions to be implemented within the circuit and to realize each function with smaller circuit blocks. For a given abstract behavioral representation of a digital system, output of the high level synthesis phase is a register-transfer level (RTL) structure realizing specified behavior. At this step, circuit consists of functional blocks which are defined in terms of interconnected registers, multiplexers and control elements. Input to the logic synthesis phase is the RTL description of the circuit and a library of logic primitives. Logic primitives, flip-flops and control functions are determined by the selected implementation style and the target technology. Each functional block described in RTL description is transferred into the structure of interconnected logic primitives to minimize either the size or the performance in terms of critical delay or combination of both. The solution of the optimization problem can be measured in terms of cost (or objective) function. The most common quality measures used in a circuit design optimization are the area, and increasingly, power consumption. During the placement phase, the logic gates are assigned to the physical location in the environment selected as a target technology.

Up to now, most of the research has focused on developing algorithms for AND/OR or NAND/NOR circuits [6-9]. Alternatively, any Boolean function can be represented canonically based on AND/EXOR operations, which are called Reed-Muller expansions. This research was first published by Zhegalkin in 1927 [10] in Russia. In 1954, Reed [11] and Muller [12] published their work in the U.S.A. In the last decade synthesis based on AND/EXOR, OR/ XNOR realisations [13, 14] have gained more interest, because these techniques are more compact for certain types of circuits, such as error correcting circuits, and arithmetic circuits. For these reasons implementations based on exclusive-OR gates can be more economical, require less gates [15-17], and have excellent testability [18-21]. A major characteristic of the EXOR logic is the numerous possible canonical

representations of switching functions it provides [22-24]. Also recent progress in circuit technology makes the use of AND/EXOR and OR/XNOR gates feasible [25-28].

Figure 1.2 shows a typical ASIC synthesis design flow based on logic synthesis. The key steps in the ASIC design are: behavioral synthesis which allows the design at higher levels of abstraction by automating the translation and optimization of a behavioral description, or high-level model, into an RTL implementation. Behavioral synthesis tools have been developed which translate the behavioural model to an RTL model [29,30] The register-transfer level (RTL) is often entered textually in a HDL such as VHDL.

Logic synthesis can be divided into there major steps:

1. To convert the description from RTL to logic level, which consist of gates, flip-flops and latches.
2. The logic optimization task is to optimize the description through various procedures in terms of area, speed and testability.
3. Produce a gate level net-list.

Finally, at the physical level, the network is built on a slice of silicon using a complex mapping scheme that translate transistors and wires into fine-line patterns of metals and other substances.

The main objective of this thesis is to concentrate on logical synthesis part. By developing a new optimization techniques and algorithms, which can be used to convert between two-level logic implementations (the product-of-sums form, OR/AND) into two-level Dual Reed-Muller forms (OR/XNOR) and find the optimal form with the minimum number of terms hence, less gates and less area. The Dual Reed-Muller forms (OR/XNOR) can be implemented by using a programmable logic array (PLA) for a two level logic as shown in figure 1.3.

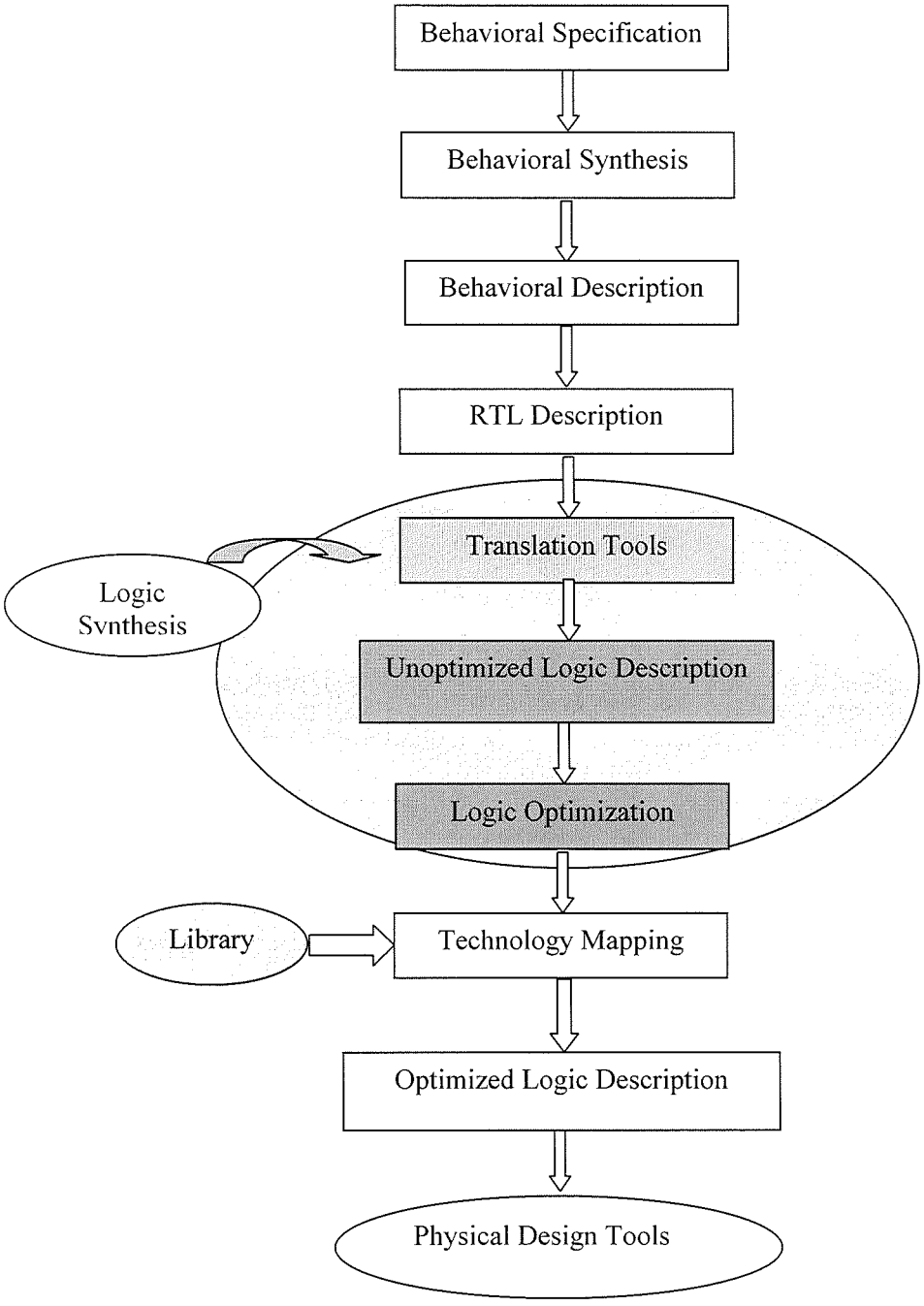


Figure 1.2: General overview of an ASIC design flow

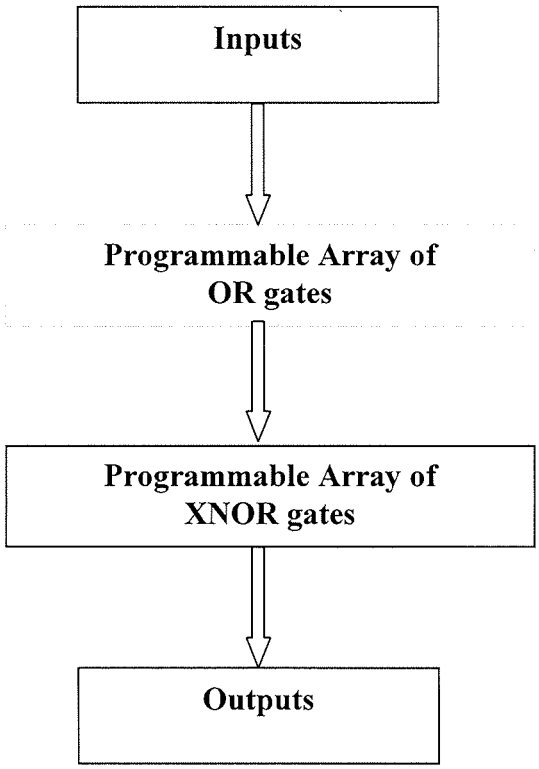


Figure 1.3: Two-Level programmable logic array structure for (OR/XNOR) gates

The work which is developed in this thesis does not replace previous work but complements and enhances it. It gives the designer a larger search space and hence a better chance of finding the ideal solution.

1.2 Background

This section presents the background theory and defines some basic notations that are used throughout this thesis.

There are two standard canonical forms to represent a Boolean function:

1.2.1 Sum of Products

Boolean functions can be expressed by the Sum of Products (SOP) form as given in equation (1.1) [31, 32].

$$f(x_{n-1}x_{n-2}...x_0) = \sum_{i=0}^{2^n-1} a_i m_i \qquad (1.1)$$

Where the subscript i can be expressed in a binary form as $i = (i_{n-1}i_{n-2}...i_0)_2$, '$\Sigma$' is the OR operator, $[a_0,a_1, \ldots,a_{2^n-1}]$ is the truth vector of the function f, $a_i \in \{0,1\}$.

$$a_0 = f(0,0,...,0)$$
$$a_1 = f(0,0,...,1)$$
$$\vdots$$
$$a_{2^n-1} = f(1,1,...,1)$$

The minterm m_i can be represented as $m_i = \dot{x}_{n-1}\dot{x}_{n-2}...\dot{x}_0$

Where

$$\dot{x}_j = \begin{cases} \bar{x}_j, & i_j = 0 \\ x_j, & i_j = 1 \end{cases} \tag{1.2}$$

Where j is from 0 to n-1.

Example 1.1

A three variable function, $f(x_2,x_1,x_0)$ can be expanded by the SOP form as follows:

$$f(x_2,x_1,x_0) = a_{000}\bar{x}_2\bar{x}_1\bar{x}_0 + a_{001}\bar{x}_2\bar{x}_1x_0 + a_{010}\bar{x}_2x_1\bar{x}_0 + a_{011}\bar{x}_2x_1x_0 + a_{100}x_2\bar{x}_1\bar{x}_0 + a_{101}x_2\bar{x}_1x_0 + a_{110}x_2x_1\bar{x}_0 + a_{111}x_2x_1x_0$$

1.2.2 Product of Sums

The same Boolean functions can also be expressed by the Product of Sums (POS) form as given in equation (1.3).

$$f(x_{n-1},x_{n-2},\ldots x_0) = \prod_{i=0}^{2^n-1}(d_i + M_i) \tag{1.3}$$

Where 'Π' represents logical products (AND), the '+' is OR operation and i is a binary n-tuple $i = (i_{n-1}\ i_{n-2}\ \ldots\ i_0)_2$, $[d_0,d_1,\ldots,d_2{}^n{}_{-1}]$ is the truth vector of the function f, $d_i \in \{0,1\}$. If d_i equals to zero then M_i will be retained in the POS, since '0' is the Boolean additive identity such that $0 + M_i = M_i$ [33], and M_i is a sum term (maxterm)

$$M_i = \sum_{k=n\text{-}1}^{0} \overset{\bullet}{x}_k^{\,i_k} = \overset{\bullet}{x}_{n\text{-}1}^{\,i_{n-1}} + \overset{\bullet}{x}_{n\text{-}2}^{\,i_{n-2}} + \ldots + \overset{\bullet}{x}_{0}^{\,i_0} \tag{1.4}$$

Where

$$\overset{\bullet}{x}_k = \begin{cases} x_k & i_k = 0 \\ \overline{x}_k & i_k = 1 \end{cases} \tag{1.5}$$

Where k is from 0 to n-1.

Example 1.2

A two variable function, $f(x_1,x_0)$ can be expanded by the POS form as follows:

$$f(x_1,x_0) = (d_0 + x_1 + x_0)\cdot(d_1 + x_1 + \overline{x}_0)\cdot(d_2 + \overline{x}_1 + x_0)\cdot(d_3 + \overline{x}_1 + \overline{x}_0)$$

$$= \prod_{i=0}^{3} d_i M_i$$

We will refer to the coefficients of SOP form and the coefficients of POS form as a and d respectively.

1.3 Reed-Muller forms based on AND/EXOR operations

Definition 1.1 An n-variable Boolean function can be expressed canonically by a Fixed Polarity Reed-Muller (FPRM) which is also known as Generlized Reed-Muller (GRM) form where each variable can be complemented or un-complemented, but not both, with polarity p expressed in a binary n-tuple, $p = (p_{n-1}\ p_{n-2}\ldots\ p_0)_2$, as follows [34, 35]

$$f(x_{n-1}x_{n-2}...x_0) = \oplus \sum_{k=0}^{2^n-1} b_k \ddot{\Omega}_k \tag{1.6}$$

Where ' $\oplus \sum$ ' is the XOR operator, $\ddot{\Omega}_k = \ddot{x}_{n-1}\ddot{x}_{n-2}...\ddot{x}_0$,

$$\ddot{x}_j = \begin{cases} 1, & k_j = 0 \\ \dot{x}_j, & k_j = 1 \end{cases} \tag{1.7}$$

$$\dot{x}_j = \begin{cases} \overline{x}_j, & p_j = 1 \\ x_j, & p_j = 0 \end{cases}$$

$k = (k_{n-1}\, k_{n-2}...k_0)$, and j is from 0 to n-1. Where $b_k \in \{0,1\}$ indicates the presents or absence of the product terms. This is a Positive Polarity Reed-Muller (PPRM) expression [36].

Sasao [37] shows that there are 7 classes of AND-EXOR expressions, this thesis focus on three major forms of Reed-Muller expansions:

A. The Positive Polarity Reed-Muller (PPRM) form is an EXOR sum of products where each variable is in un-complemented form. This is also called a zero polarity form:

$$f(x_{n-1}x_{n-2}...x_0) = b_0 \oplus b_1x_0 \oplus b_2x_1 \oplus ... \oplus b_{2^n-1}x_{n-1}x_{n-2}...x_0$$

A number of algorithms have been proposed to obtain this form from the sum of products [34, 35, 38-42].

B. The Fixed Polarity Reed-Muller (FPRM) form (GRM) where each variable can be complemented or un-complemented, but not both:

$$f(x_{n-1}x_{n-2}...x_0) = b_0 \oplus b_1\dot{x}_0 \oplus b_2\dot{x}_1 \oplus ... \oplus b_{2^n-1}\dot{x}_{n-1}\dot{x}_{n-2}...\dot{x}_0$$

Where $\dot{x} \in \{x, \bar{x}\}$.

This form can be obtained from the zero polarity form using the identity $\bar{x} = 1 \oplus x$. This give rise to 2^n fixed polarities, many algorithms are available to derive these polarities [31, 37]. To calculate the polarity for any GRM function, each variable is replaced by a 1 or 0 which depends on whether variable x_j is in complemented or true form respectively.

For example, a three-variable function $f(\dot{x}_2\dot{x}_1\dot{x}_0)$:

$$f(x_2x_1x_0) \text{ has polarity } 0$$
$$f(x_2x_1\bar{x}_0) \text{ has polarity } 1$$
$$f(\bar{x}_2x_1x_0) \text{ has polarity } 4$$

The advantage of the GRM is that some minimisation is possible by finding a polarity, which minimise the number of terms. There are some algorithms to find the minimal fixed polarity expansion [43-47].

C. The mixed polarity Reed-Muller form, where each variable can be complemented or un-complemented. For an n variable function, there can be up to $2^{n^{2^{n-1}}}$ different expansions or polarities [48].

1.4 Definitions and identities of EXOR gate

The EXOR operation is defined as follows:

$$A \oplus B = \overline{A}B + A\overline{B}$$

For any Boolean variable x, the following identities are used for EXOR operations:

$$\dot{x} \oplus 1 = \bar{\dot{x}} \qquad \dot{x} \oplus 0 = \dot{x}$$
$$\dot{x} \oplus \dot{x} = 0 \qquad \dot{x} \oplus \bar{\dot{x}} = 1$$
$$\dot{x}_0(1 \oplus \dot{x}_1) = \dot{x}_0 \oplus \dot{x}_0\dot{x}_1$$

Where

$$\dot{x} = \begin{cases} x & true\ form \\ \bar{x} & complemented\ form \end{cases}$$

For any EXOR expression the following properties hold:

$$x_2 \oplus (x_1 \oplus x_0) = (x_2 \oplus x_1) \oplus x_0 = x_2 \oplus x_1 \oplus x_0 \quad (associative)$$
$$x_2(x_1 \oplus x_0) = x_2x_1 \oplus x_2x_0 \quad (distributive)$$
$$x_1 \oplus x_0 = x_0 \oplus x_1 \quad (commutative)$$

The Kronecker product of two matrices ($A_{mA \times nA}$, $B_{mB \times nB}$) is defined as follows:

$$A \otimes B = \begin{bmatrix} a_{11} & . & . & . & a_{1nA} \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ a_{mA1} & . & . & . & a_{mAnA} \end{bmatrix} \otimes B = \begin{bmatrix} a_{11}B & . & . & . & a_{1nA}B \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ a_{mA1}B & . & . & . & a_{mAnA}B \end{bmatrix}$$

Each $a_{i,j}B$ is a block of size mB x nB., and $A \otimes B$ is of the size mA x nA x mB x nB [49,50].

The realisation of Reed-Muller circuits led to computational difficulties for functions of even moderate size, due to the large memory requirements. To overcome these difficulties Reed-Muller functions can be represented using a binary decision diagram, thus reducing the storage requirements. In addition, the computational requirements are also reduced since an efficient method for computing the spectral coefficients is employed.

Binary Decision Diagrams (BDDs) are used as a data structure for Boolean functions. It was introduced by Lee in 1959 [51] and later by Akers [52]. In 1986 Bryant introduced the concept of Ordered Binary Decision Diagrams (OBDD) which, allow canonical representation and efficient manipulation of Boolean functions [53].

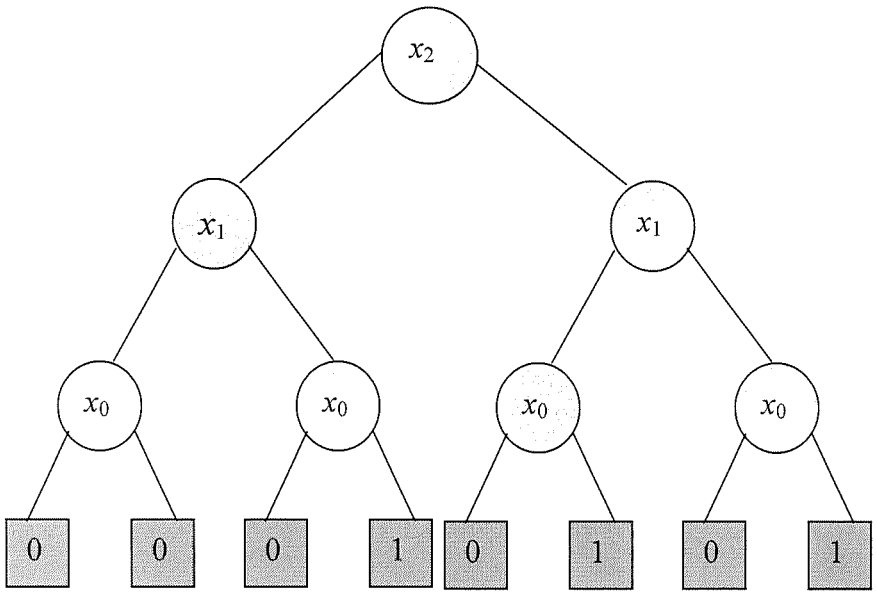
BDD method has been widely used for synthesis, analysis and optimisation of both combinational and sequential logic [54-55]. In addition, BDDs have been used for design verification [56]. Although the technique provides a useful model for large applications, it suffers from the drawback that there is a difficulty in determining

an optimum ordering of the function input variables to achieve the simplest network [57-59].

Definition 1.2 A BDD is a directed acyclic graph representing a Boolean function as shown in Figure 1.1. It can be uniquely defined as a tuple, BDD= (Φ, V, E, {0,1}), Where Φ is the function node (root) [60], V is the set of internal nodes representing the input variables, E is the set of edges, and {0,1} are the terminal nodes. A completely specified function f can be specified by two sets of cubes, an on-set X(on) and an off-set X(off), where f(X(on)) = 1, f(X(off)) = 0.

Examples 1.3

A Boolean function $f(x_2,x_1,x_0) = \sum\{3, 5, 7\}$ can be expressed by a BDD as shown in Figure 1.4.



$$f(x_2, x_1, x_0) = (x_2 x_1 x_0 + x_2 \bar{x}_1 x_0 + \bar{x}_2 x_1 x_0)$$

Figure 1.4: BDD for example 1.3

The following reduction rules are used to reduce BDD to RBDD:

1. Remove duplicate internal nodes: If two nodes v_1 and v_2 have var(v_1) = var(v_2), low(v_1) = low(v_2), and high(v_1) = high (v_2), then eliminate one of the two nodes and redirect all incoming edges to the other internal node.
2. Remove redundant internal node: If an internal node v has low(v) = high(v), then delete node v and redirect all incoming edges to high(v).
3. Remove duplicate terminal: Delete all but one identical terminal and redirect all edges into that terminal [61, 62].

Examples 1.4

Reduce the following Boolean function $f(x_2,x_1,x_0) = \sum\{3, 5, 7\}$ by applying the reduction rules.

1. Merge duplicate nodes, this will lead to:

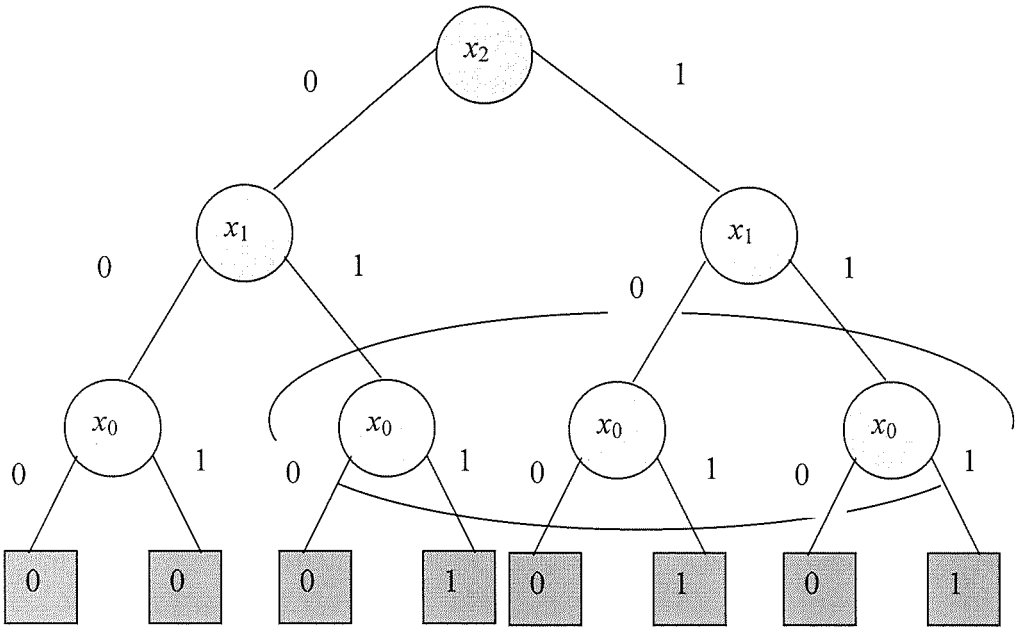


Figure 1.5: Duplicate nodes for example 1.4

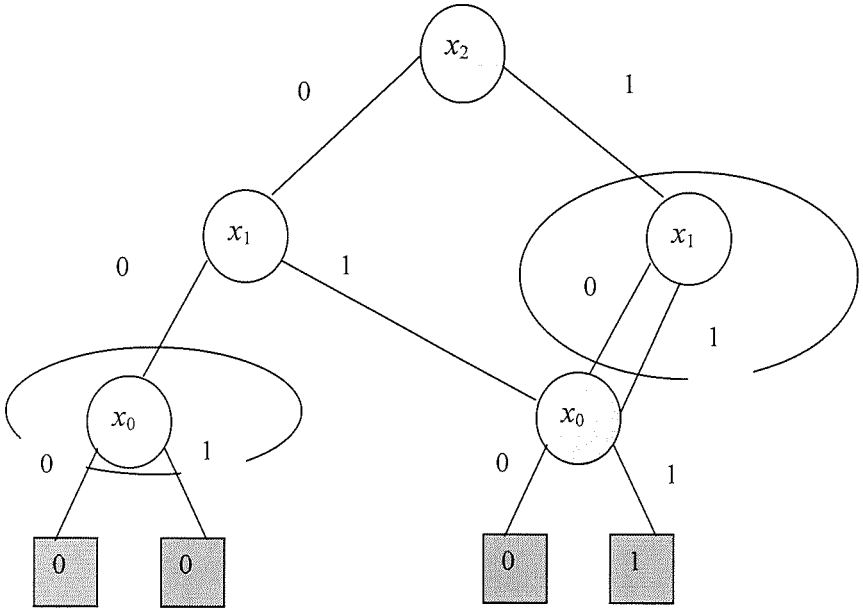
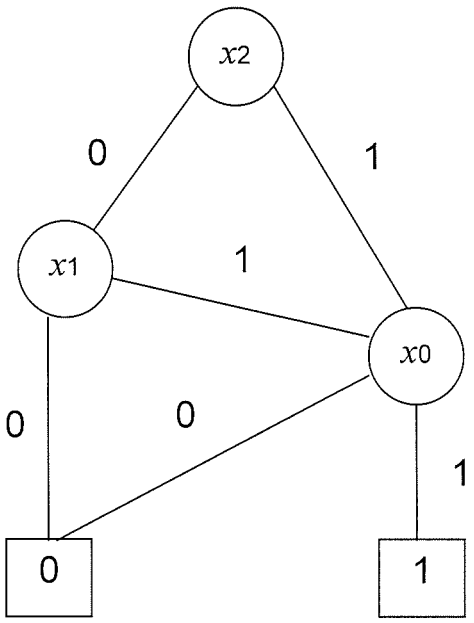


Figure 1.6: Redundant nodes for example 1.4

2. Eliminate redundant nodes, this will lead to:



$$f(x_2, x_1, x_0) = (x_1 x_0 + x_2 x_0)$$

Figure 1.7: ROBDD for example 1.4

1.5 BDD using XOR operators

There are three types of expansions using XOR gates as follows:

$$f = f_0 \oplus x_{n-1}.f_2 \tag{1.8}$$

$$f = \bar{x}_{n-1}.f_2 \oplus f_1 \tag{1.9}$$

$$f = \bar{x}_{n-1}.f_0 \oplus x_{n-1}.f_1 \tag{1.10}$$

Where the cofactors $f_0 = (0, x_{n-2}, \ldots, x_0)$, $f_1 = (1, x_{n-2}, \ldots, x_0)$, and $f_2 = f_0 \oplus f_1$ are independent of the expansion variable x_{n-1} and can be expanded further with respect to the other variables. Equation (1.10) is the Shannon expansion, where the '$\oplus$' is the XOR operator. Equation (1.8) is the positive Davio (pD), it is also known as Positive Polarity Reed-Muller (PPRM) expansion, and each variable appear in the true (un-complemented) form only. Equation (1.9) is the negative Davio (nD) expansion, were the variables appear in the complemented form only [37]. Equation (1.8) can be obtained by replacing $\bar{x}_{n-1}$ with $1 \oplus x_{n-1}$ in equation (1.10) as follows:

$$f = (1 \oplus x_{n-1}).f_0 \oplus x_{n-1}.f_1 = 1.f_0 \oplus x_{n-1}(f_0 \oplus f_1) = 1.f_0 \oplus x_{n-1}f_2$$

Similarly equation (1.9) can be obtained by replacing x_{n-1} with $1 \oplus \bar{x}_{n-1}$ in equation (1.10) as follows:

$$f = \bar{x}_{n-1}.f_0 \oplus (1 \oplus \bar{x}_{n-1}).f_1 = 1.f_1 \oplus \bar{x}_{n-1}(f_0 \oplus f_1) = \bar{x}_{n-1}.f_2 \oplus 1.f_1$$

Example 1.4

Convert the following arbitrary three variables function into PPRM canonical form.

$$f(x_2x_1x_0) = a_{000}\overline{x}_2\overline{x}_1\overline{x}_0 + a_{001}\overline{x}_2\overline{x}_1x_0 + a_{010}\overline{x}_2x_1\overline{x}_0 + a_{011}\overline{x}_2x_1x_0 + a_{100}x_2\overline{x}_1\overline{x}_0 + a_{101}x_2\overline{x}_1x_0 + a_{110}x_2x_1\overline{x}_0 + a_{111}x_2x_1x_0$$

Because the minterms are mutually exclusive [63], the OR can be replaced by the XOR, and by replacing $\overline{x}_{n-1}$ with $1 \oplus x_{n-1}$ the following equation is obtained.

$$f(x_2x_1x_0) = a_{000}(1 \oplus x_2)(1 \oplus x_1)(1 \oplus x_0) \oplus a_{001}(1 \oplus x_2)(1 \oplus x_1)x_0 \oplus a_{010}(1 \oplus x_2)x_1(1 \oplus x_0) \oplus a_{011}(1 \oplus x_2)x_1x_0 \oplus a_{100}x_2(1 \oplus x_1)(1 \oplus x_0) \oplus a_{101}x_2(1 \oplus x_1)x_0 \oplus a_{110}x_2x_1(1 \oplus x_0) \oplus a_{111}x_2x_1x_0$$

Where

$$(1 \oplus x_2)(1 \oplus x_1)(1 \oplus x_0) = 1 \oplus x_0 \oplus x_1 \oplus x_1x_0 \oplus x_2 \oplus x_2x_0 \oplus x_2x_1 \oplus x_2x_1x_0$$

Hence

$$f(x_2x_1x_0) = a_{000}(1 \oplus x_0 \oplus x_1 \oplus x_1x_0 \oplus x_2 \oplus x_2x_0 \oplus x_2x_1 \oplus x_2x_1x_0) \oplus a_{001}(x_0 \oplus x_1x_0 \oplus x_2x_0 \oplus x_2x_1x_0) \oplus a_{010}(x_1 \oplus x_1x_0 \oplus x_2x_1 \oplus x_2x_1x_0) \oplus a_{011}(x_1x_0 \oplus x_2x_1x_0) \oplus a_{100}(x_2 \oplus x_2x_0 \oplus x_2x_1 \oplus x_2x_1x_0) \oplus a_{101}(x_2x_0 \oplus x_2x_1x_0) \oplus a_{110}(x_2x_1 \oplus x_2x_1x_0) \oplus a_{111}x_2x_1x_0$$

By rearranging the terms the following is obtained

$$f(x_2,x_1,x_0) = a_{000}(1) \oplus x_0(a_{000} \oplus a_{001}) \oplus x_1(a_{000} \oplus a_{010}) \oplus x_2(a_{000} \oplus a_{100}) \oplus x_1x_0(a_{000} \oplus a_{001} \oplus a_{010} \oplus a_{011}) \oplus x_2x_0(a_{000} \oplus a_{001} \oplus a_{100} \oplus a_{101}) \oplus x_2x_1(a_{000} \oplus a_{010} \oplus a_{100} \oplus a_{110}) \oplus x_2x_1x_0(a_{000} \oplus a_{001} \oplus a_{010} \oplus a_{011} \oplus a_{100} \oplus a_{101} \oplus a_{110} \oplus a_{111})$$

Figure 1.8 shows the Positive Davio tree for a 3-variable function or the PPRM Form.

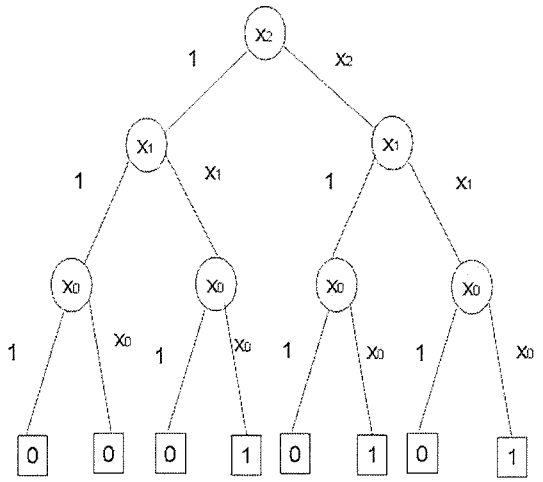


Figure 1.8: Positive Davio tree or (PPRM)

1.6 Reed-Muller representation based on OR/XNOR operations

There exists an alternative algebraic expansion for logical functions, namely the Dual Reed-Muller expansion, which involves the operations of logical equivalence (LEQ) and inclusive-OR to provide a POS form [64-67].

Definition 1.3 Any n-variable function can be expressed by the Dual Reed-Muller (DRM) expression as:

$$f(x_{n-1},x_{n-2},\ldots,x_0) = \odot \prod_{i=0}^{2^n-1} (\mathbf{c}_i + S_i) \tag{1.11}$$

Where ' $\odot$ ' is XNOR operator, $[c_2{}^n{}_{-1}, c_2{}^n{}_{-2}, \ldots, c_0]$ is the truth vector of the function f, $c_i \in \{0,1\}$ and S_i represents a Sum term as

$$S_i = \sum_{k=n-1}^{0} \tilde{x}_k = \tilde{x}_{n-1} + \tilde{x}_{n-2} + \ldots + \tilde{x}_0, \tag{1.12}$$

Where

$$\tilde{x}_k = \begin{cases} 0 & i_k = 0 \\ x_k & i_k = 1 \end{cases} \tag{1.13}$$

Much research has been devoted and focused on AND/EXOR Reed-Muller forms [68, 69]. This thesis focuses on a special class of OR/XNOR circuits, called Dual Reed-Muller (DRM) forms [64- 67]. Dual Reed-Muller forms can be classified according to its polarity. If each variable in the DRM form appears in un-complemented or true form, this form is called Positive Polarity Dual Reed-Muller (PPDRM). The second form is called Fixed Polarity Dual Reed-Muller (FPDRM) forms were each variable appears in un-complemented or complemented form but not both. In recent years there is a growing interest in design of logical functions with XNOR gates [26,27]. Functions realized with such OR/XNOR circuits can have less gates, less connections, occupy less Silicon area, dissipate less power, easily testable, and hence cheaper.

1.7 Aim of the thesis

The aim of this research is to develop a variety of algorithms for synthesis and optimization for the Dual Reed-Muller expressions. The first part of this research will focus on developing efficient and fast algorithms to convert between Product of Sums and Dual Reed-Muller forms. Due to the lack of efficient algorithms to convert between Product of Sums and the Dual Reed-Muller forms and to find the optimal polarity, new transformation algorithms are introduced in this thesis. Several algorithms are introduced in this thesis to convert between POS and FPDRM forms for single and multi output functions. The second and the third part of this research are focused on optimization and finding efficient solutions, to find the best polarity. There are 2^n polarities for each function using FPDRM forms. Therefore, it is required to find the optimal polarity among all the FPDRM forms, to lead to a function with the minimum number of sums. The solutions have to be systematic and efficient so they can be implemented on a computer and can handle large number of variables for larger circuits.

Applications of Dual Reed-Muller implementations have not become popular despite the advantages of using XNOR gates.

1.8 Thesis Overview

Any Boolean function can be expressed by two concepts, they are based on OR/AND and OR/XNOR operations respectively. Figure 1.9 shows the main sections in this thesis, where each section or chapter is described as follows:

Chapter 2 presents a computational technique method for converting Boolean functions in SOP form into Fixed Polarity Reed-Muller (FPRM) expressions, and vice versa. It also converts multi output SOP expressions to multi output Fixed Polarity Reed-Muller expressions. The Reed-Muller Transform matrix is presented in the form of matrix decomposition, as layered vertical and horizontal Kronecker

matrices. Sparse technique is used to store the non-zero elements instead of storing the whole matrix. The conversion technique can be used for single output or multi-output functions.

Chapter 3 defines the basic theory and notations that will be used in the following chapters. It proposes the Dual Reed-Muller expansions (DRM), which are based on OR/XNOR gates. This chapter introduces the transformation techniques to convert a POS, single or multi output, into Dual Reed-Muller in single or multi output functions.

Chapter 4 A Bi-directional conversion algorithm is first proposed to convert between a single output function and Product of Sums (POS) and Positive Polarity Dual Reed-Muller (PPDRM) forms, without using any of the transformation matrices. This algorithm can be used for any polarity.

Chapter 5 presents an algorithm to compute all the coefficients of the Fixed Polarity Dual Reed-Muller (FPDRM) with polarity **p** from any polarity **q**. This technique is used to find the best polarity of FPDRM among the 2^n fixed polarities. The algorithm is based on the dual property and the Gray code strategy. Therefore, there is no need to start from POS form to find FPDRM coefficients for all the polarities. The proposed methods are efficient in terms of memory size and CPU time.

Chapter 6 presents two algorithms, which can be used to convert from POS form to FPDRM form and find the optimal polarity for large number of variables. The first algorithm is used to compute the coefficients of the Positive Polarity Dual Reed-Muller (PPDRM) or FPDRM directly from the truth table of POS, without the use of mapping techniques [65] and without the use of matrix operation [64]. This algorithm is also used to compute the coefficients of POS from PPDRM or

FPDRM. The second algorithm will find the optimal polarity among the 2^n different polarities for large n-variable functions, without generating all of the polarity sets. This algorithm is based on separating the truth vector of POS and the use of sparse techniques, which will lead to optimal polarity. Time efficiency and computing speed are thus achieved in this technique.

Chapter 7 conclusions and future work.

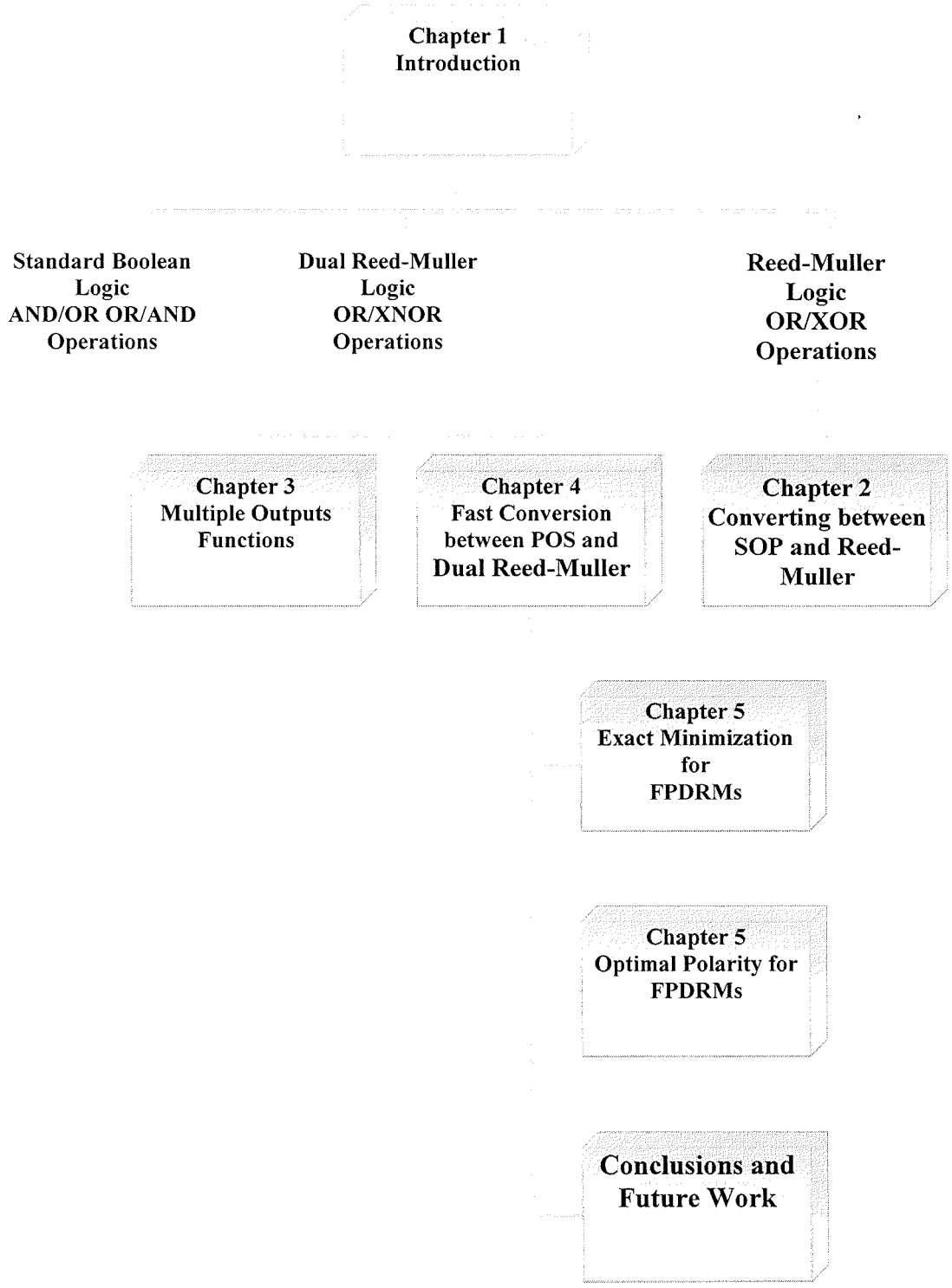


Figure 1.9: Structure of the thesis

Chapter 2

Efficient Polarity Conversion

2.1 Introduction

This chapter presents a computational method for converting Boolean functions in SOP form into Fixed Polarity Reed-Muller (FPRM) expressions, and vice versa [34, 35]. It also converts multi output SOP expressions to multi output FPRM expressions. The Reed-Muller Transform matrix is presented in the form of matrix decomposition, as layered vertical and horizontal Kronecker matrices. Sparse technique is used to store the non-zero elements instead of storing the whole matrix. The conversion technique can be used for single output or multi-output functions. The developed program is tested on personal computers and the results for some benchmark functions of up to 20 inputs and 40 outputs are tested.

A Boolean function can be expressed canonically based on the Reed-Muller (AND/EXOR) expansion. Therefore, conversion methods are needed to convert between SOP and Reed-Muller forms. There are several techniques to convert

between SOP and Reed-Muller [70-73], most of the available techniques are not suitable for multioutput and for large functions. The algorithm in [74] requires less computer memory since it computes from only the on-set coefficients, but it takes much more time when n is greater than 14.

In this chapter a Computer-Algorithm is introduced using the sparse matrix and partitioning technique to convert between SOP and FPRM. When a large matrix is sparse [75], with a high proportion of its entries zero or some other fixes value, it is convenient to store only the non-zero entries of the matrix. The representation of such a sparse matrix is a circular linked list structure [76]. In this representation, each non-zero element belongs to two lists: a list of the non-zero elements of its column and of its row. Each list is ordered according to the appearance of the elements in the left to right or top to bottom travel of the row or, column respectively. Linked lists efficiently represent structures that vary in size [77]. The idea of dividing a large matrix into sub-matrices or blocks arises naturally. The blocks can be treated as if they were the elements of the matrix and the partitioned matrix becomes a matrix of matrices. Partitioning plays an important role in sparse matrix technology because many algorithms designed primarily for matrices of numbers can be generalized to operate on matrices of matrices. The greater flexibility of the concept of partitioning then brings useful computational advantages. Alternatively, partitioning can be considered simply as a data management tool, which helps to organize the transfer of information between main memory and auxiliary devices. Storing a partitioned matrix implies storing a set of sub-matrices. By Partitioning the sparse matrix to many levels to store only one level, and then implement a certain procedure to calculate the reset of the elements without the need to store the whole Reed-Muller transform matrix, which can get very large for large values of input variables n. Since most of the elements in the transformation matrix of Reed-Muller of order $N = 2^n$ are zero. Therefore sparse and partitioning methods would be a good option in this case to save space and time by only storing non-zero elements.

2.2 Polarity conversion for a single output Boolean Functions

Boolean functions can be expressed by the SOP form as given in equation (2.1).

$$f(x_{n-1}x_{n-2}\ldots x_0)=\sum_{i=0}^{2^n-1}a_i m_i \tag{2.1}$$

Alternatively, any n-variable Boolean function can be represented based on Reed-Muller expansion which is based on AND/XOR operators [78].

Definition 2.1 An n-variable Boolean function can be expressed canonically by a Fixed Polarity Reed-Muller (FPRM) form with polarity p expressed in a binary n-tuple, $p=(p_{n-1}\,p_{n-2}\ldots p_0)_2$ [79], as follows

$$f(x_{n-1}x_{n-2}\ldots x_0)=\oplus\sum_{k=0}^{2^n-1}b_k\ddot{\Omega}_k \tag{2.2}$$

Where ' $\oplus\sum$ ' is the XOR operator, $\ddot{\Omega}_k=\ddot{x}_{n-1}\ddot{x}_{n-2}\ldots\ddot{x}_0$,

$$\ddot{x}_j=\begin{cases}1, & k_j=0\\ \dot{x}_j, & k_j=1\end{cases} \tag{2.3}$$

$$\dot{x}_j=\begin{cases}\overline{x}, & p_j=1\\ x_j, & p_j=0\end{cases} \tag{2.4}$$

Where $a_p=[a_0, a_1, \ldots, a_{2^n-1}]^t$ and $b_p=[b_0, b_1, \ldots, b_{2^n-1}]^t$.

The conversion between SOP and a Fixed Polarity Reed-Muller requires constructing the transformation matrix R(n) and substitute in the following equations [37].

$$f = \mathrm{X}_j \dot{\mathrm{R}}(n)\mathrm{a} \tag{2.5}$$

Where the basis vector is defined as follows:

$$\mathrm{X}_j = \begin{cases} [1 \quad \bar{x}_j], & p_j = 1 \\ [1 \quad x_j], & p_j = 0 \end{cases}$$

Therefore, minterms for zero polarity can be identified by expanding a Kronecker product of X_j as follows:

$$\mathrm{X} = \bigotimes_{i=0}^{n-1} [1 \quad x_i]$$

The basic Reed-Muller matrix for $n = 1$ is defined as follows:

$$\dot{\mathrm{R}}(1) = \begin{cases} \mathrm{R}(1) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, & p_j = 0 \\ \mathrm{R}(1) = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, & p_j = 1 \end{cases}$$

For n-variable function the Reed-Muller transformation matrix is calculated as follows:

$$\dot{\mathrm{R}}(n) = \bigotimes_{i=0}^{n-1} \dot{\mathrm{R}}(1) \tag{2.6}$$

Where the ' $\otimes$ 'denotes Kronecker product. The Kronecker product performed as a symbolic computation in X generates the product terms appearing in the FPRMs.

The elements of R(1) and R(n) are 0 and 1, and the calculations are done over GF(2).

Therefore, Reed-Muller expansion for a given truth vector can be deduced by deriving b from a and substituting in the following equation.

$$f(x_{n-1}x_{n-2}...x_0) = \{ [1 \;\; x_{n-1}] \otimes [1 \;\; x_{n-2}] \otimes ... \otimes [1 \;\; x_0] \} * b \qquad (2.7)$$

The Reed-Muller transform matrix R(n) is a self-inverse matrix over GF (2), therefore $(R(n))^{-1} = R(n)$ [73]. Thus, conversion from FPRM to SOP can be accomplished by using the following equation.

$$a = R(n)b \qquad (2.8)$$

Example 2.1

Compute Reed-Muller coefficients with zero polarity for a three-variable function $f(x_2, x_1, x_0) = \Sigma(1,2,4,7)$.

The transformation matrix for p = 0 is calculated using equation (2.6) as follows:

$$R(3) = R(1) \otimes R(1) \otimes R(1)$$

$$R(3) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The coefficients of the PPRM are calculated using the following equation

$$b = R(3)\, a$$

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

Where the coefficients for the vector b are calculated as follows:

$$b_0 = (1 \bullet 0) \oplus (0 \bullet 1) \oplus (0 \bullet 1) \oplus (0 \bullet 0) \oplus (0 \bullet 1) \oplus (0 \bullet 0) \oplus (0 \bullet 0) \oplus (0 \bullet 1) = 0$$
$$b_1 = (1 \bullet 0) \oplus (1 \bullet 1) \oplus (0 \bullet 1) \oplus (0 \bullet 0) \oplus (0 \bullet 1) \oplus (0 \bullet 0) \oplus (0 \bullet 0) \oplus (0 \bullet 1) = 1$$
$$b_2 = (1 \bullet 0) \oplus (0 \bullet 1) \oplus (1 \bullet 1) \oplus (0 \bullet 0) \oplus (0 \bullet 1) \oplus (0 \bullet 0) \oplus (0 \bullet 0) \oplus (0 \bullet 1) = 1$$
$$b_3 = (1 \bullet 0) \oplus (1 \bullet 1) \oplus (1 \bullet 1) \oplus (1 \bullet 0) \oplus (0 \bullet 1) \oplus (0 \bullet 0) \oplus (0 \bullet 0) \oplus (0 \bullet 1) = 0$$
$$b_4 = (1 \bullet 0) \oplus (0 \bullet 1) \oplus (0 \bullet 1) \oplus (0 \bullet 0) \oplus (1 \bullet 1) \oplus (0 \bullet 0) \oplus (0 \bullet 0) \oplus (0 \bullet 1) = 1$$
$$b_5 = (1 \bullet 0) \oplus (1 \bullet 1) \oplus (0 \bullet 1) \oplus (0 \bullet 0) \oplus (1 \bullet 1) \oplus (1 \bullet 0) \oplus (0 \bullet 0) \oplus (0 \bullet 1) = 0$$
$$b_6 = (1 \bullet 0) \oplus (0 \bullet 1) \oplus (1 \bullet 1) \oplus (0 \bullet 0) \oplus (1 \bullet 1) \oplus (0 \bullet 0) \oplus (1 \bullet 0) \oplus (0 \bullet 1) = 0$$
$$b_7 = (1 \bullet 0) \oplus (1 \bullet 1) \oplus (1 \bullet 1) \oplus (1 \bullet 0) \oplus (1 \bullet 1) \oplus (1 \bullet 0) \oplus (1 \bullet 0) \oplus (1 \bullet 1) = 0$$

The product terms for PPRM expression are calculated using the Kronecker product in X as follows:

$$X = \begin{bmatrix} 1 & x_2 \end{bmatrix} \otimes \begin{bmatrix} 1 & x_1 \end{bmatrix} \otimes \begin{bmatrix} 1 & x_0 \end{bmatrix} = \begin{bmatrix} 1 & x_2 \end{bmatrix} \otimes \begin{bmatrix} 1 & x_0 & x_1 & x_1 x_0 \end{bmatrix}$$
$$= 1 \quad x_0 \quad x_1 \quad x_1 x_0 \quad x_2 \quad x_2 x_0 \quad x_2 x_1 \quad x_2 x_1 x_0$$

Therefore,

$$f = Xb = \begin{bmatrix} 1 & x_0 & x_1 & x_1x_0 & x_2 & x_2x_0 & x_2x_1 & x_2\,x_1x_0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= (1 \bullet 0) \oplus (x_0 \bullet 1) \oplus (x_1 \bullet 1) \oplus (x_1x_0 \bullet 0) \oplus (x_2 \bullet 1) \oplus (x_2x_0 \bullet 0) \oplus (x_2x_1 \bullet 0) \oplus (x_2x_1x_0 \bullet 0) = x_0 \oplus x_1 \oplus x_2$$

$$f = x_0 \oplus x_1 \oplus x_2$$

The final circuit for polarity zero is given as follows.

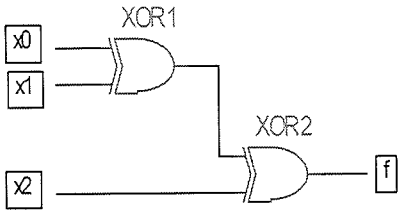


Figure 2.1: Minimal form under polarity 0

Lemma: An n-variable completely specified Boolean function can be uniquely expressed by a 2^n dimensional vector with polarity p, either $a_p = [a_0, a_1, \ldots, a_{2^n-1}]^t$ in SOP format or $b_p = [b_0, b_1, \ldots, b_{2^n-1}]^t$ in Reed-Muller format, then these two vectors can be converted mutually by equation (2.9).

$$a_p = R_n b_p \text{ or } b_p = R_n a_p \qquad (2.9)$$

2.3 Basic theory and algorithms

Converting between SOP and PPRM requires using the transformation matrix (R(n)). For an n-variable Boolean function, the size of R(n) matrix is 2^n by 2^n. Therefore, it requires a huge size of memory to store this matrix. To avoid storing the entire elements in the Reed-Muller matrix, a partitioning technique is developed in this chapter, which will partition the R(n) matrix into sub-matrices. The sub-matrices are smaller in size and require much less memory space than the original transformation matrix. The basic idea for the partitioning technique is to employ two smaller matrices (Key and Basic), which can be multiplied together using Kronecker product to give the original Reed-Muller matrix as follows:

$$\mathrm{R}(n) = \begin{bmatrix} Basic \otimes Key & Basic \otimes Key \\ Basic \otimes Key & Basic \otimes Key \end{bmatrix} \tag{2.10}$$

The matrix in Eq (2.11) is partitioned with the bold arrows into four matrices. By examining those matrices, three of them are identical and the last one consists of zeros. If the same matrix is partitioned into sixteen matrices instead of four as shown in Eq (2.12), nine of these matrices are the same and the reset of the matrices are zeros. Notice that the elements above the main diagonal are always zero.

R(n) matrix can be partitioned into many sections, the number of the sub matrices is determined by using the factor 2^i, where i = 1.... n-1, and n is the number of variables.

Step 1: R(n) matrix is partitioned into sub matrices, the size of each sub matrix is 2^i by 2^i.

Step 2: Each of the identical sub matrices in R(n) matrix is denoted the Key matrix.

Step 3: A new matrix (Basic) is constructed from the Key matrix. The Key matrix is partitioned into sub matrices (Basic), where the size of the Basic matrix is given as:

$$\frac{2^n}{2^i} * \frac{2^n}{2^i}$$

Example 2.3

Let the number of variables n equal to 3.

The size of R(3) = 2^3 by 2^3 = 8 by 8, if we let i = 2, then the size of Key matrix is 2^i by $2^i = 2^2 * 2^2$ as shown in equation (2.13), and the size of the Basic matrix is calculated as follows $8/(2^2)$ by $8/(2^2)$, which is 2 by 2 as shown in equation (2.14).

$$RM(2^3) = \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \quad (2.11)$$

$$RM(2^3) = \left[\begin{array}{cc|cc|cc|cc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \tag{2.12}$$

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

$$Basic = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{2.14}$$

Algorithm 1

To generate the original Reed-Muller matrix $R(2^n)$ using the Key and the Basic matrices, the following steps are followed:

Step 1: The key and the Basic matrices are generated by using equation (2.6).

Step 2: For each element in the Basic matrix generate one Key matrix. (a) If the first element in the Basic matrix is one, then generate one Key matrix. (b) If the element in the Basic matrix is zero, generate another Key matrix, but with all elements equal to zero.

Step 3: Start with the Basic matrix; read the first element of the Basic matrix. If the first element is one construct the first row of the Key matrix. If the first element is zero construct a row where all the elements are zero.

Step 4: Shift the number of columns for this row by the size of the Key matrix 2^i. Read the second element in the first row of the Basic matrix, and repeat step 2.

Step 5: Repeat the same procedure as in step 2 and step 3, till the first row of RM is constructed.

Step 6: Repeat the same procedure for the rest of the rows for the Key matrix.

Step 7: Shift the number of row for the new R again by the size of the Key matrix 2^i, and start all over again, but this time for the second row of the Basic matrix.

Step 8: Repeat as above till the whole Reed-Muller matrix is constructed.

Algorithm 2

The other technique is using sparse matrix, which will store only the ones elements below the main diagonal in the Key and the Basic matrices, to save memory and computing time. The following equation is derived to find the number of non-zero elements in the transformation matrix.

$$NR(1) = (2^n * 2^n) * [1 - \sum_{i=1}^{i=n} 3^{(i-1)} * (1/(2^i * 2^i))] \qquad (2.15)$$

Where n is the number of variables.

According to equation (2.15) the percentage of non-zero elements for R(n) is given as follows:

$$PNR(1) = \frac{NR(1)}{2^n * 2^n} * 100\% \qquad (2.16)$$

For example if $n = 10$, the percentage of non-zero elements

$$(59049/1048576) * 100\ \% = 5.631\%$$

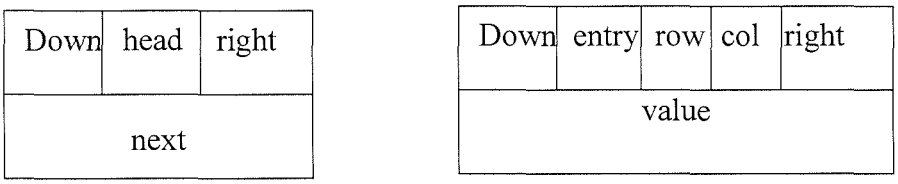
The following table shows the percentage of non-zero elements for different numbers of n.

Table 2.1: Number of variables (n) versus Percentage of ones

n	Size of n matrix	Number of ones	Percentage of one elements
1	4	3	75
2	16	9	56.25
3	64	27	42.18
4	256	81	31.64
5	1024	243	23.73
6	4096	729	17.8
7	16384	2187	13.3
8	65536	6561	10
9	262144	19683	7.5
10	1.04858e 6	59049	5.63

Therefore, a sparse technique would be a good method to store Reed-Muller matrix. A linked list is used to represent the sparse matrix. This will be an efficient

way to represent structures that vary in size. In our data representation, each column of a sparse matrix is represented as a circularly linked list with a head node [79, 80]. A similar representation is used for each row of a sparse matrix. Each head node has three additional fields: down, right, and next as shown in Figure 2.2. The down field is used to link into a column list and the right field is used to link into a row list. The next field links the head nodes together.



(a) Head node (b) Entry node

Figure 2.2: Node structure for sparse matrix

Each element node has five fields: row, column, down, right, value. The right field is used to link to the next zero elements in the same row, and the down field to link to the next zero element in the same column.

Another matrix is stored and designated the Vector matrix by using sparse and linked list technique. This matrix should contain the truth vector (a).

The final results are obtained by, building each row of the RM matrix from the Basic and the Key matrices, as was described in algorithm 2.

To determine the coefficients for the RM (b_i) if it is 0 or 1, the following steps are used:

(a) Set a counter D = 0.

(b) If there is an element in the first row of the RM matrix at (column x) and there is an element in Vector matrix (row x); then increment both pointer for RM and Vector matrix to the next location.

(c) If column x > row x; then while row x is less than column x and row x is not equal to NULL, increment the counter D by one.

(d) Else if row x is greater than column x ; then while column x is less than row x and column x is not equal to NULL, increment the counter D by one.

(e) If RM's pointer is equal to NULL, while Vector's pointer is not equal to NULL, increment the counter D by one and then go to step (g).

(f) If Vector's pointer is equal to NULL, while RM's pointer is not equal to NULL, increment the counter D by one and then go to step (g).

(g) Test if the counter is even or odd. If the counter is odd then d_i equals 1; otherwise d_i is 0 according to the following identity:

$$d_i = \begin{cases} 1 \oplus 1 \oplus \ldots \oplus 1_n \oplus 0 \oplus 0 \oplus \ldots \oplus 0_m = 1, & \textit{if } n \textit{ and } m \textit{ are odd} \\ 1 \oplus 1 \oplus \ldots \oplus 1_n \oplus 0 \oplus 0 \oplus \ldots \oplus 0_m = 0, & \textit{if } n \textit{ and } m \textit{ are even} \end{cases}$$

(d) Repeat steps (a to g) for every row of RM matrix, with the same Vector matrix till the end of the RM matrix.

Example 2.4

Compute the Reed-Muller form of the following 4-variable function

$$f(x_3 x_2 x_1 x_0) = \sum(0,1,3,5,7,9,10,12,13,14)$$

$$R(4) = R(1) \otimes R(1) \otimes R(1) \otimes R(1)$$

Step 1; store this function in the Vector matrix, using a linked list, as shown in figure 2.3.

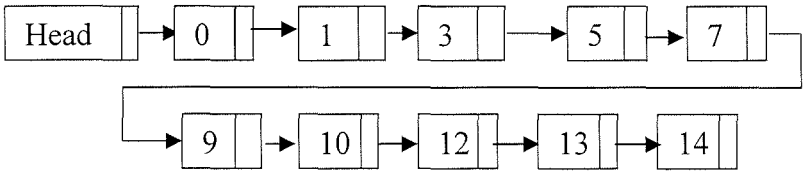


Figure 2.3: Vector Matrix Structure

Step 2; store the Key matrix, using the same technique, as shown in Figure 2.4.

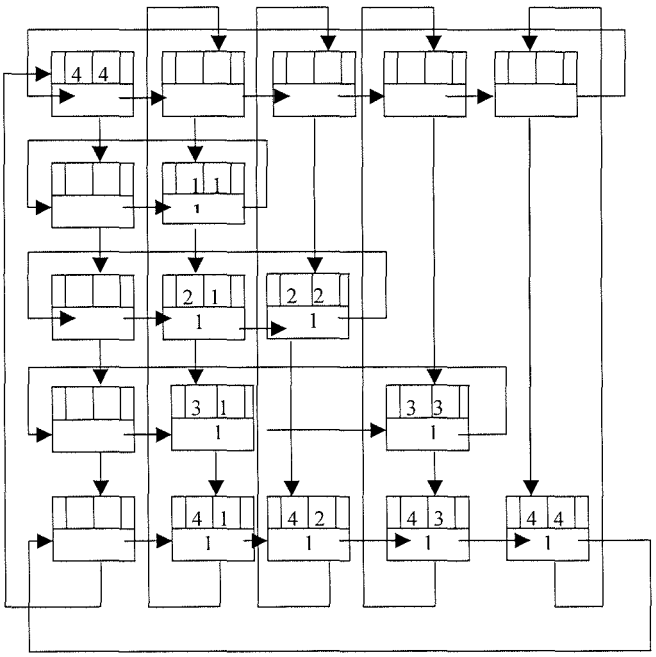


Figure 2.4: Key matrix Structure

Step 3; store the Basic matrix, as shown in Figure 2.5.

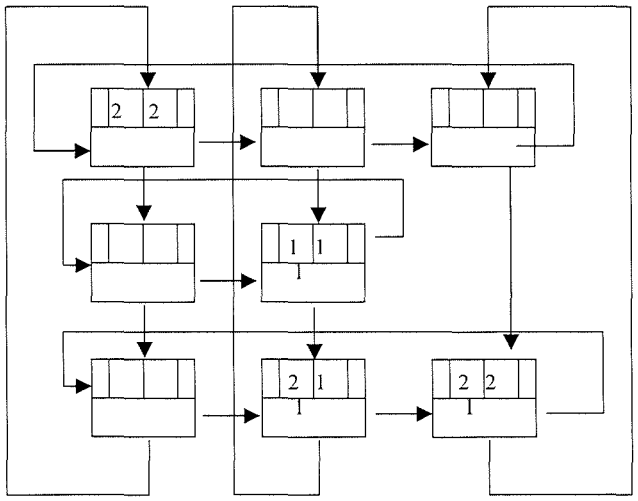


Figure 2.5: Basic Matrix Structure

2.4 Conversion for Multi-Output Functions

Conversion algorithms for multi-outputs SOP into multi-outputs Reed-Muller, and multi-output Reed-Muller into SOP were accomplished by adding a pointer to each node in the Vector matrix, which points to array. This array will store the output functions for that particular input.

Example 2.5

Take a three-variable function and the number of output functions is 4,

x_2	x_1	x_0	f_4	f_3	f_2	f_1
0	0	0	1	0	0	1
1	1	0	0	1	0	1
1	1	1	1	1	0	0

The new Vector matrix is as shown in figure 2.6.

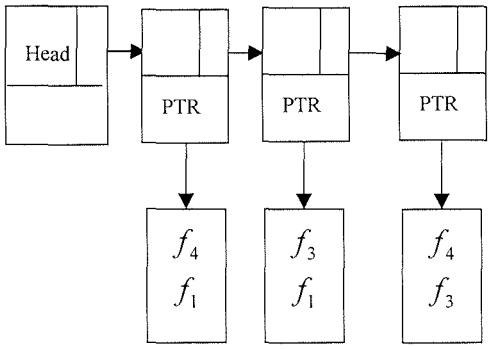


Figure 2.6: Vector matrix for multi output

The rest of the procedure should be the same as in the previous sections, but with some extra steps performed for each extra output function.

Any Boolean function may be represented by a fixed polarity modulo-2 expansion For any n-variable Boolean function there are 2^n distinct FPRMs.

To convert any SOP expansion from polarity q to polarity p, every subscript $i, 0 \le i < 2^{n-1}$ should be converted using equation (2.17), where" $\oplus$ " and "<=" are bitwise XOR an assignment operators respectively [63].

.

$$i <= i \oplus p \tag{2.17}$$

Where p is the polarity number.

Hence, conversion from SOP to any FPRM forms can be done as follows:

1. Convert the on-set minterms for an n-variable function to the on-set minterms with polarity p using Eq (2.17).
2. Use algorithm 1 and 2 to find the corresponding RM coefficients.

2.5 Experimental Results

In this chapter a computer algorithm has been presented for fast bidirectional conversion between sum-of-products and RM for large number of inputs. This method is generalised to large multiple output Boolean functions. This method is developed based on using partitions and the sparse technique. The algorithms were implemented in C language and the program is compiled by using Borland c++ compiler. Then it was tested on a personal computer with Pentium 3, 1GHz CPU and 256M RAM under Window operating system. Experimental results are presented in Table 2.2 where '~0' means that the CPU time is almost zero. The computation time depends on the number of variables (n). For incompletely specified Boolean functions, don't cares are set to off-sets (0) for the outputs (0). This algorithm calculates the Reed-Muller coefficients form the minterms of the SOPs. Although the number of the Reed-Muller coefficients for some circuits is higher than the minterms of the SOPs, this is occurred because we calculated the Reed-Muller coefficients for all the multiple output circuits. Our techniques reported combatable results compared to other techniques, in terms of the variable numbers (n) and output numbers (o). The programme was un-efficient for variable numbers greater than 19, because it requires more memory and we could not run, therefore, it would require some modifications to make the programme handle variables greater than 19.

Table 2.2:

Conversion results of some benchmark functions

Circuit	n	o	#(SOP)	#(RM)	Time (S)
Table5	17	15	158	122129	36.75
B12	15	9	431	31488	4.67
Sao9	10	4	58	883	~0
Misex3c	14	14	305	15739	1.76
Sao2	10	4	58	883	0.06
Apex4	9	19	440	479	~0
T481	16	1	481	50353	11.37
bw	5	28	87	32	~0
pdc	16	40	2810	57860	23.23
Apex4	9	19	440	479	0.06
Random	19	1	8	116737	43.61

2.6 Summary

This chapter has introduced a partition and Sparse methods which can be used in converting between SOP form and RM forms. This method depends on partitioning the transformation Reed-Muller matrix into sub-matrices, where the sub-matrices repeat a special order. The original Reed-Muller matrix can be generated by storing two of the sub matrices and perform algorithm 1. Therefore, storing the whole matrix is not efficient in terms of memory and time. to manipulate the data for calculating the coefficients of Reed-Muller.

The second method that is used for converting between POS and RM forms is the sparse method. This method is based on storing none zero elements in the matrix. Since most of the elements in RM matrix are zeros, therefore, sparse method is very efficient in terms of memory and CPU speed, because it does not require processing every element in the matrix.

Chapter 3

Dual Reed-Muller form

3.1 Introduction

Logic synthesis based on Reed-Muller techniques has shown several advantages over the use of the standard Boolean functions such as SOP. Some of these advantages are high testability, low cost for arithmetic and parity checker. Several conversion algorithms to convert between SOP and Fixed polarity Reed-Muller exist [38, 45, 63,81-86]. Alternatively Reed-Muller form can be presented using the Dual Reed-Muller (DRM) form which was introduced by Green [64-67]. This form is based on the use of OR/XNOR gates. The XNOR gate plays a major role in various circuits especially circuits used in arithmetic process such as full adders, comparators [87-89]. Another feature for XNOR is to have a small number of transistors to implement [25-28,90]. Any n-variable Boolean function in the POS form can be expressed by 2^n

Fixed Polarity Dual Reed-Muller form (FPDRM). Extensive research has been done on developing techniques and algorithms to convert between SOP and Reed-Muller and very little has been done on converting from Product of Sums and the Dual Reed-Muller form or even to find the best or the optimal polarity which will lead to the minimal function with the least number of Sums.

This chapter covers the basic theory and notations which will be used in the Dual Reed-Muller form. It also introduces new operations which can be used to describe the Dual Reed-Muller form and convert the POS to DRM for single output function, and multi output functions. When designing complex circuits for mass production, it is worth it to try many possible solutions such as RM, DRM, etc to find a good solution to reduce components and cost.

3.2 DRM expansion of logical functions

Definition 2.1 Any n-variable function can be expressed by the Dual Reed-Muller (DRM) expression as:

$$f(x_{n-1},x_{n-2},\ldots,x_0) = \odot \prod_{i=0}^{2^n-1} (c_i + S_i) \tag{3.1}$$

Where '$\odot$' is XNOR operator, $[c_2{}^n{}_{-1},c_2{}^n{}_{-2},\ldots,c_0]$ is the truth vector of the function f, $c_i \in\{0,1\}$ and S_i represents a Sum term as

$$S_i = \sum_{k=n-1}^{0} \tilde{x}_k = \tilde{x}_{n-1} + \tilde{x}_{n-2} + \ldots + \tilde{x}_0, \tag{3.2}$$

Where

$$\tilde{x}_k = \begin{cases} 0 & i_k = 0 \\ x_k & i_k = 1 \end{cases} \tag{3.3}$$

Table (3.1) illustrate the basic the XNOR operations.

Table 3.1:

Truth table for XNOR

$A\ B$	$A \oplus B$	$A \odot \mathbf{B}$
0 0	0	1
0 1	1	0
1 0	1	0
1 1	0	1

The following XNOR operations and identities are defined

The XNOR '$\odot$'operation is defined as follows:

$$A \odot B = \overline{A}\,\overline{B} + AB \tag{3.4}$$

For any Boolean variable x, the following identities are used for XNOR operations:

$$\begin{aligned} x \odot 1 &= x & \overline{x} \odot 1 &= \overline{x} \\ x \odot 0 &= \overline{x} & \overline{x} \odot 0 &= x \\ x \odot x &= 1 & x \odot \overline{x} &= 0 \end{aligned} \tag{3.5}$$

For any XNOR expression the following properties hold:

$$x_2 \odot (x_1 \odot x_0) = (x_2 \odot x_1) \odot x_0 = x_2 \odot x_1 \odot x_0 \qquad (\textit{associative}) \tag{3.6}$$

$$x_2 + (x_1 \odot x_0) = (x_2 + x_1) \odot (x_2 + x_0) \qquad (\textit{distributive}) \tag{3.7}$$

$$x_1 \odot x_0 = x_0 \odot x_1 \qquad (commutative) \qquad (3.8)$$

$$x_1 \bullet x_0 = x_1 \odot x_0 \odot (x_1 + x_0) \qquad (3.9)$$

An arbitrary 2-variable function can be presented by the canonical POS as follows:

$$f(x_1, x_0) = (f(0,0) + x_1 + x_0) \bullet (f(0,1) + x_1 + \bar{x}_0) \bullet (f(1,0) + \bar{x}_1 + x_0) \bullet (f(1,1) + \bar{x}_1 + \bar{x}_0)$$

The ANDs ($\bullet$) operators are replaced by $\odot$ operators, since if all the Maxterms are ANDed together the answer is zero and if all the Maxterms are XNORed together the answer is zero too.

$$f(x_1, x_0) = (f(0,0) + x_1 + x_0) \odot (f(0,1) + x_1 + \bar{x}_0) \odot (f(1,0) + \bar{x}_1 + x_0) \odot (f(1,1) + \bar{x}_1 + \bar{x}_0)$$

By applying the following identity to replace $x \odot 0 = \bar{x}$, this will give the following result:

$$f(x_1,x_0) = (f(0,0) + x_1 + x_0) \odot (f(0,1) + x_1 + (0 \odot x_0)) \odot (f(1,0) + (0 \odot x_1) + x_1)) \odot (f(1,1) + (0 \odot x_1) + (0 \odot x_0))$$

Using the distributive law, the above expression is modified to

$$f(x_1,x_0) = (f(0,0) + x_1 + x_0) \odot (f(0,1) + x_1) \odot (f(0,1) + x_1 + x_0) \odot (f(1,0) + x_1) \odot (f(1,0) + x_1 + x_0)) \odot f(1,1) \odot (f(1,1) + x_1) \odot (f(1,1) + x_0) \odot (f(1,1) + x_1 + x_0)$$

By rearranging and grouping the common terms the following expression is obtained

$$f(x_1,x_0) = f(1,1) \odot (x_0 + (f(1,0) \odot f(1,1))) \odot (x_1 + (f(0,1) \odot f(1,1)))$$
$$\odot (x_1 + x_0 + (f(0,0) \odot f(0,1) \odot f(1,0) \odot f(1,1))) \quad (3.10)$$

For simplicity, c_j coefficients are introduced to express the coefficients of the DRM expansion, where j corresponds to the decimal equivalent of the binary subscript of f_{klm}, and equation (3.10) can be written as:

$$f(x_1,x_0) = c_3 \odot (x_0 + c_2) \odot (x_1 + c_1)$$
$$\odot (x_1 + x_0 + c_0) \quad (3.11)$$

This is the Dual Reed-Muller expansion based on OR/XNOR operation [64-66]. Taking a 3-variable function as an example:

We can also obtain the DRM expansion of an n-variable function as follows:

$$f(x_{n-1} \sim x_0) = c_{2^n-1} \odot (x_0 + c_{2^n-2}) \odot (x_1 + c_{2^n-3}) \odot \ldots \odot$$
$$(x_{n-1} + \cdots + x_1 + c_1) \odot (x_{n-1} + \cdots + x_0 + c_0) \quad (3.12)$$

Let us now introduce two new operations to simplify the DRM expansion of n-variable function. Their definitions are as follows:

Continuous sum operation [65]

The Continues Sum (++) of two matrices ($A_{mA \times nA}$, $B_{mB \times nB}$) is defined as follows:

$$A++B = \begin{bmatrix} a_{11} & . & . & . & a_{1nA} \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ a_{mA1} & . & . & . & a_{mAnA} \end{bmatrix} ++B = \begin{bmatrix} a_{11}+B & . & . & . & a_{1nA}+B \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ a_{mA1}+B & . & . & . & a_{mAnA}+B \end{bmatrix} \quad (3.13)$$

Each $a_{ij}B$ is a block of size $_{mB\ x\ nB}$. The size of the new matrix (A++ B) is equal to the size of ($_{mA\ x\ nA}$ x $_{mB\ x\ nB}$).

For example:

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} ++ \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1+b_1 & a_1+b_2 & a_2+b_1 & a_2+b_2 \\ a_1+b_3 & a_1+b_4 & a_2+b_3 & a_2+b_4 \\ a_3+b_1 & a_3+b_2 & a_4+b_1 & a_4+b_2 \\ a_3+b_3 & a_3+b_4 & a_4+b_3 & a_4+b_4 \end{bmatrix}$$

The continuous sum operation meets the associative law

$$[A] ++ \{[B] ++ [c]\} = \{[A] ++ [B]\} ++ [C] = [A] ++ [B] ++ [C],$$

But it does not meet the commutative law

$$[A] ++ [B] \neq [B] ++ [A]$$

Added coincidence operation or Matrix multiplication based on logical equivalence (LEQ) [64-65] is defined as follows:

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} a_{11} & . & . & a_{14} \\ . & . & & . \\ . & & . & . \\ a_{31} & . & . & a_{44} \end{bmatrix} \circ \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \tag{3.14}$$

Where '$\circ$' represents matrix multiplication based on OR/XNOR. Thus

$$c_1 = (a_{11} + b_1) \odot (a_{12} + b_2) \odot (a_{13} + b_3) \odot (a_{14} + b_4)$$
$$c_2 = (a_{21} + b_1) \odot (a_{22} + b_2) \odot (a_{23} + b_3) \odot (a_{24} + b_4)$$
$$c_3 = (a_{31} + b_1) \odot (a_{32} + b_2) \odot (a_{33} + b_3) \odot (a_{34} + b_4)$$
$$c_4 = (a_{41} + b_1) \odot (a_{42} + b_2) \odot (a_{43} + b_3) \odot (a_{44} + b_4)$$

We can obtain the following relationship between the f_j coefficients and c_j coefficients for n-variable functions.

$$c = [T_n] \circ f \tag{3.15}$$

$$f = [T_n] \circ c \tag{3.16}$$

Where

$$T_n = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} + + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} + + ... + + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \; n \text{ times} \tag{3.17}$$

with

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

and

$$c = [c_0 \; c_1 .... c_2{}^n{}_{-1}]^t$$

$$f = [f_0 \; f_1 .... f_2{}^n{}_{-1}]^t$$

For a 3-variable function, equations (3.15) and (3.16) can be written as

$$
\left.
\begin{aligned}
c_0 &= f_0 \odot f1 \odot \ldots \odot f_7 \\
c_1 &= f_1 \odot f_3 \odot f_5 \odot f_7 \\
c_2 &= f_2 \odot f_3 \odot f_6 \odot f_7 \\
c_3 &= f_3 \odot f_7 \\
c_4 &= f_4 \odot f_5 \odot f_6 \odot f_7 \\
c_5 &= f_5 \odot f_7 \\
c_6 &= f_6 \odot f_7 \\
c_7 &= f_7
\end{aligned}
\right\} \qquad (3.18)
$$

$$
\left.
\begin{aligned}
f_0 &= c_0 \odot c_1 \odot \ldots\ldots \odot c_7 \\
f_1 &= c_1 \odot c_3 \odot c_5 \odot c_7 \\
f_2 &= c_2 \odot c_3 \odot c_6 \odot c_7 \\
f_3 &= c_3 \odot c_7 \\
f_4 &= c_4 \odot c_5 \odot c_6 \odot c_7 \\
f_5 &= c_5 \odot c_7 \\
f_6 &= c_6 \odot c_7 \\
f_7 &= c_7
\end{aligned}
\right\} \qquad (3.19)
$$

Example: Apply the above method to get DRM expansion of a 4-variable function $f(x_3, x_2, x_1, x_0) = \prod(1,2,4,7,8,11,13,14)$.

$$
T(4) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} + + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} + + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} + + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}
$$

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \\ c_{10} \\ c_{11} \\ c_{12} \\ c_{13} \\ c_{14} \\ c_{15} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$c_0 = (0+1)\odot(0+0)\odot(0+0)\odot(0+1)\odot(0+0)\odot(0+1)\odot(0+1)\odot(0+0)\odot(0+0)$$
$$\odot(0+1)\odot(0+1)\odot(0+0)\odot(0+1)\odot(0+0)\odot(0+0)\odot(0+1) = 1$$

$$c_1 = (1+0)\odot(0+0)\odot(1+0)\odot(0+1)\odot(1+0)\odot(0+1)\odot(1+1)\odot(0+0)\odot(1+0)$$
$$\odot(0+1)\odot(1+1)\odot(0+0)\odot(1+1)\odot(0+0)\odot(1+0)\odot(0+1) = 1$$

$$c_2 = (1+1)\odot(1+0)\odot(0+0)\odot(0+1)\odot(1+0)\odot(1+1)\odot(0+1)\odot(0+0)\odot(1+0)$$
$$\odot(1+1)\odot(0+1)\odot(0+0)\odot(1+1)\odot(1+0)\odot(0+0)\odot(0+1) = 1$$

$$c_3 = (1+1)\odot(1+0)\odot(1+0)\odot(0+1)\odot(1+0)\odot(1+1)\odot(1+1)\odot(0+0)\odot(1+0)$$
$$\odot(1+1)\odot(1+1)\odot(0+0)\odot(1+1)\odot(1+0)\odot(1+0)\odot(0+1) = 1$$

$$c_4 = (1+1)\odot(1+0)\odot(1+0)\odot(1+1)\odot(0+0)\odot(0+1)\odot(0+1)\odot(0+0)\odot(1+0)$$
$$\odot(1+1)\odot(1+1)\odot(1+0)\odot(0+1)\odot(0+0)\odot(0+0)\odot(0+1) = 1$$

Similarly for c_5-c_{15}

$$c_{15} = (1+1)\odot(1+0)\odot(1+0)\odot(1+1)\odot(1+0)\odot(1+0)\odot(1+1)\odot(1+0)\odot(1+0)$$
$$\odot(1+1)\odot(1+1)\odot(1+0)\odot(1+1)\odot(1+0)\odot(1+0)\odot(0+1) = 1$$

To generate the sum terms the following basis vectors are used for $n = 4$ [61, 62].

$$[0 \quad x_3] + +[0 \quad x_2] + +[0 \quad x_1] + +[0 \quad x_0] =$$
$$= [0 \quad x_0 \quad x_1 \quad (x_1 + x_0) \quad x_2 \quad (x_2 + x_0) \quad (x_2 + x_1) \quad (x_2 + x_1 + x_0) \quad x_3 \quad (x_3 + x_0)$$
$$(x_3 + x_1) \quad (x_3 + x_1 + x_0) \quad (x_3 + x_2) \quad (x_3 + x_2 + x_0) \quad (x_3 + x_2 + x_1)$$
$$(x_3 + x_2 + x_1 + x_0)]$$

Finally the dual Reed-Muller form is generated as follows:

$$f(x_3, x_2, x_1, x_0) = [0 \quad x_3] + +[0 \quad x_2] + +[0 \quad x_1] + +[0 \quad x_0] \circ c$$
$$f(x_3, x_2, x_1, x_0) = x_0 \odot x_1 \odot x_2 \odot x_3$$

3.3 Generalization for large functions

An algorithm has been d eveloped based on equations (3.15) and (3.16) to convert between Product of Sums and Dual Reed-Muller. Algorithms one and two from Chapter two were adopted in this Chapter. The adopted algorithms were based on Sparse and partition techniques [66, 80]. In order to use the algorithms for converting between Product of Sums and Dual Reed-Muller, the following changes have been used.

To construct the Dual Reed-Muller transform matrix (T_n) from the Reed-Muller transform matrix (RM_n), which is defined by equation (3.13), the following steps are performed. Recall that the Reed-Muller transform is applied over GF (2) [92].

Step 1: Construct the transformation matrix (RM_n) using Kronecker product '$\otimes$' [92, 93] as follows:

$$RM_n = \bigotimes_{i=1}^{n} M^1 \tag{3.20}$$

where

$$M^1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{3.21}$$

Step 2: Transpose the transformation matrix (RM_n) by replacing the rows by the columns, complement the elements of (RM_n) matrix by changing the zeros to ones, and the ones to zeros.

Example

For a three-variable function construct the Dual Reed-Muller matrix (T_n) by using the Reed-Muller matrix (RM).

RM matrix is constructed using equation (3.20), which gives the following result.

$$RM(2^3) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{3.22}$$

$$RM = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{3.23}$$

The Dual Reed-Muller matrix (T_n) is generated from equation (3.23) by taking the transpose of equation (3.23) and changing each zero and one elements to one and zero respectively. Therefore, (T_n) is

$$T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{3.24}$$

Sparse and partitioning techniques are similar to the methods used in Chapter two, except that the zero elements are used here instead of the one elements. Hence the T matrix from equation (3.24) is partitioned as follows:

$$T = \left[\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}\right] \tag{3.25}$$

Examining equation (3.25), three of sub matrices are identical while the last one consists of zeros, and the elements below the main diagonal are just ones. Therefore, the two matrices that are needed to generate the matrix in equation (3.25) are:

$$\text{Key} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

and

$$\text{Basic} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

The only elements that are needed from the Key and Basic matrices in order to generate equation (3.25) are the zero elements.

The following equation is derived to find the total number of zero elements in the transformation matrix (T_n) that are needed to convert between Product of Sums and the Dual Reed-Muller form.

$$NR(0) = (2^n * 2^n) - (2^n * 2^n) * [\sum_{i=1}^{i=n} 3^{(i-1)} * (1/(2^i * 2^i))] \qquad (3.26)$$

Therefore, a big saving in terms of memory size is achieved by using the Sparse and partitioning techniques.

3.4 Conversion for Multi-Output Functions

Conversion algorithms for multi output POS into multi output DRM, and multi output DRM into POS are accomplished by adding a pointer to each node in the Vector matrix. This pointer will point to array. Each element in the array will include the i^{th} output function for that particular sum. A unique counter will be associated with each output function. Each counter will be incremented by one for that particular sum.

The rest of the procedure should be the same as in the previous sections.

3.5 Experimental Results

Algorithms are implemented using C language; the program is compiled using Borland C++ compiler. Then it is tested on a personal computer with Pentium 3, 1GHz CPU and 256M RAM under Window operating system. The experimental results are shown in Table 2 where '~0' means that the CPU time is almost zero. Factors on which the computation time depends are the number of variables (n), number of Sums, and the number of outputs (o). For incompletely specified Boolean functions, don't cares are set to on-sets (1) for the outputs.

To our knowledge there are no experimental results published in this topic to compare with. Green [64] has introduced theoretical approach to convert between POSs and Dual Reed-Muller coefficients. This technique is suitable for small number of variables, because it requires building and storing the transformation matrix. The size of the transformation matrix is 2^n by 2^n, therefore, it requires a huge size of memory to store. Hence, [64] technique is not efficient it terms of memory.

This algorithm calculates the Reed-Muller coefficients form the minterms of the SOPs. Although the number of the Reed-Muller coefficients for some circuits is higher than the minterms of the SOPs, this is occurred because we calculated the Reed-Muller coefficients for all the multiple output circuits. Our techniques reported

combatable results compared to other techniques, in terms of the variable numbers (n) and output numbers (o). The programme was un-efficient for variable numbers greater than 20, because it requires more memory and we could not run, therefore, it would require some modifications to make the programme handle variables greater than 19.

Table 3.2:

Conversion results of some functions from MCNC Benchmark

	n	o	Init.terms for POS	DRM.terms	Time (S)
Misex3c	14	14	305	4605	1.59
Clip	9	5	167	111	0.06
B12	15	9	431	41	3.3
Clip1	4	4	4	5	~0
Alu4	14	8	1030	1850	2.19
Pdc	16	1	2810	881	16.86
Apex4	9	19	440	506	~0
Spla	16	1	2310	482	15.49
Misex1	8	7	32	15	~0
Table5	17	1	158	2240	28.4
Ex1010	10	10	1023	1023	0.11
Con1	7	2	10	12	~0
Rd84	8	4	257	253	~0
Inc	7	9	34	57	~0
Random	20	1	8	4379	5.44

3.6 Summary

In this chapter we have introduced and explained the mathematical operations, such as continuous sum, that are needed for converting POS to Dual Reed-Muller expressions. This is needed to calculate the Dual Reed-Muller matrix. This matrix is required to calculate the coefficients of the Dual Reed-Muller form.

Sparse and partition algorithms have been introduced also in this chapter as in chapter two, but with some modifications to deal with the Dual Reed-Muller expressions. Sparse and partition algorithms have been programmed using C language to convert for a single and multi ouput functions.

Chapter 4

Fast transformation between POS and DRM functions

4.1 Introduction

To derive Fixed Polarity Dual Reed-Muller (FPDRM) coefficients from POS coefficients using the transformation matrix would be very costly in terms of memory and CPU time. The transformation matrix requires the construction and storing of the matrix TM_n which has a size of 2^n by 2^n for n-variables. This is overcome by introducing a fast transformation equations for computing all the coefficients of the Dual Reed-Muller forms from the coefficients of the product of sums. The coefficients of Dual Reed-Muller terms are derived without the need to generate or store the transformation matrix of the Dual Reed-Muller terms.

4.2 Definitions and representations of DRM expressions

Definition 4.1 An n-variable Boolean function can be expressed as:

$$f(x_{n-1}, x_{n-2}, ..., x_0) = \prod_{i=0}^{2^n-1} (\mathrm{d}_i + \mathrm{M}_i) \tag{4.1}$$

$$\mathrm{M}_i = \sum_{k=n-1}^{0} \overset{\bullet i_k}{x_k} = \overset{\bullet i_{n-1}}{x_{n-1}} + \overset{\bullet i_{n-2}}{x_{n-2}} + ... + \overset{\bullet i_0}{x_0} \tag{4.2}$$

Where

$$\overset{\bullet}{x}_k = \begin{cases} x_k & i_k = 0 \\ \overline{x}_k & i_k = 1 \end{cases} \tag{4.3}$$

Definition 4.2 Alternatively, any n-variable function can be expressed by the fixed polarity Dual Reed-Muller expression as:

$$f(x_{n-1}, x_{n-2}, \ldots, x_0) = \odot \prod_{i=0}^{2^n-1} (\mathrm{c}_i + \mathrm{S}_i) \tag{4.4}$$

$$\mathrm{S}_i = \sum_{k=n-1}^{0} \overset{\sim i_k}{x_k} = \overset{\sim i_{n-1}}{x_{n-1}} + \overset{\sim i_{n-2}}{x_{n-2}} + ... + \overset{\sim i_0}{x_0}, \tag{4.5}$$

Where

$$\tilde{x}_k = \begin{cases} 0 & i_k = 0 \\ x_k & i_k = 1 \end{cases} \tag{4.6}$$

Definition 4.3 Polarity vector $(p_{n-1},p_{n-2},\ldots,p_0)$ for a FPDRM of an n-variable Boolean function is a binary vector with n elements, where $p_i = 0$ indicates the variable x_i in an un-complemented form (x_i), while $p_i = 1$ indicates the variable x_i in the complemented form ($\bar{x}_i$).

Maxterms can be identified by applying a Continues Sum (++) of n basis vector of the form [0 x_i] for a '0' polarity and [0 $\bar{x}_i$] for a '1' polarity .

The coefficient vector can be derived from the truth vector d using the transform matrix as given in equation (4.9). The transform matrices for a '0' and a '1' polarity are given in equations (4.7) and (4.8) respectively.

$$TM_0 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{4.7}$$

$$TM_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.8}$$

$$\mathbf{c} = TM_n \circ \mathbf{d} \tag{4.9}$$

Where **d** is the truth vector of the POS, and **c** is the truth vector of the Dual Reed-Muller form.

In general the transformation matrix (TM_n) for a PPDRM is given as follows:

$$TM_n = TM_0 ++ TM_0 ++ \ldots ++ TM_0 \quad n \text{ times.} \tag{4.10}$$

Where '++' is the Continuous Sum. Furthermore $TM_n^{-1} = TM_n$ is a self-inverse matrix in GF (2) [93].

The FPDRM can be deduced by substituting the coefficient vector **c** in equation (4.11) for a zero polarity [64-67].

$$f(x_{n-1},x_{n-2},\ldots,x_0) = \{[0 \;\; x_{n-1}] ++ [0 \;\; x_{n-2}] ++ \ldots ++ [0 \;\; x_0]\} \circ \mathbf{c} \qquad (4.11)$$

Example 4.1

Given the two-variable function in POS form, find the fixed Dual Reed-Muller form with polarity number equals 0, using the matrix operations.

The transformation matrix for **p** = 0 is calculated using equation (4.8) as follows:

$$TM(2) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} ++ \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Using equation (4.9) the truth vector **c** is obtained as follows:

$$\begin{bmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} d_3 \\ d_2 \\ d_1 \\ d_0 \end{bmatrix}$$

Provided the order of the elements in **d** and **c** vectores are reversed [64, 66].

Using the following identities for XNOR

$$0 \odot 0 = 1$$

$$0 \odot 1 = 0$$

$$1 \odot 0 = 0$$

$$1 \odot 1 = 1$$

Hence

$c_3 = (0+d_3) \odot (1+ d_2) \odot (1+ d_1) \odot (1+ d_0) = d_3 \odot 1 \odot 1 \odot 1 = d_3 \odot 1$

$c_2 = (0+d_3) \odot (0+ d_2) \odot (1+ d_1) \odot (1+ d_0) = d_3 \odot d_2 \odot 1 \odot 1 = d_3 \odot d_2 \odot 1$

$c_1 = (0+d_3) \odot (1+ d_2) \odot (0+ d_1) \odot (1+ d_0) = d_3 \odot 1 \odot d_1 \odot 1 = d_3 \odot d_1 \odot 1$

$c_0 = (0+d_3) \odot (0+ d_2) \odot (0+ d_1) \odot (0+ d_0) = d_3 \odot d_2 \odot d_1 \odot d_0$

Finally the FPDRM function is calculated using equation (4.11) as follow:

$$f(x_1, x_0) = \{[0 \;\; x_1] + +[0 \;\; x_0]\} \circ \mathbf{c}$$

$$= \{[0 \;\; x_0 \;\; x_1 \;\; x_1 + x_0]\} \circ \mathbf{c}$$

$$= (c_3 + 0) \odot (c_2 + x_0) \odot (c_1 + x_1) \odot (c_0 + x_1 + x_0)$$

4.3 Conversion from POS to PPDRM

Equation (4.1) can be rewriting as follows

$$f(x_{n-1}, x_{n-2}, \ldots, x_0) = \odot \prod_{i=0}^{2^n - 1} (d_i + M_i) \qquad (4.12)$$

Where the AND gate is replaced by XNOR gate.

Example 4.2

When n is equal to '2', $f(x_1, x_0)$ can be expanded using equation (4.1) as the following.

$$f(x_1, x_0) = (d_0 + x_1 + x_0) \cdot (d_1 + x_1 + \bar{x}_0) \cdot (d_2 + \bar{x}_1 + x_0) \cdot (d_3 + \bar{x}_1 + \bar{x}_0)$$

This is can be rewritten using equation (4.12) as the following

$$f(x_1,x_0) = (d_0 + x_1 + x_0) \odot (d_1 + x_1 + \bar{x}_0) \odot (d_2 + \bar{x}_1 + x_0) \odot (d_3 + \bar{x}_1 + \bar{x}_0)$$

The following theory is introduced in order to convert Product of Sums to Positive Polarity Dual Reed-Muller form. Each complemented variable ($\bar{x}_i$) in equation (4.12) is replaced by the following identity $\bar{x} = 0 \odot x$ [66]. For any Dual Reed-Muller coefficient c_i, where $i = (i_{n-1} i_{n-2} \ldots i_0)$, if i_k is 0, then there is a constant '0' in S_i according to equation (4.6). Since x_k can be obtained by both x_k and $\bar{x}_k$ in M_i, therefore, i_k can be both '0' and '1' in d according to equation (4.3). If i_k is '1', then there is x_k in S_i according to equation (4.6). Because '0' can only be created by $\bar{x}_k$ in M_i, i_k can only be '1' in d according to equation (4.3). This can be formulated as equation (4.13).

$$c_i = c_{i_{n-1}\ i_{n-2}\ \ldots\ i_0} = \bigodot_{l} \prod d_{l_{n-1}\ l_{n-2}\ \ldots\ l_0} \tag{4.13}$$

Where $l = (l_{n-1}\ l_{n-2} \ldots\ l_0)$,

$$l_k = \begin{cases} y & i_k = 0 \\ 1 & i_k = 1 \end{cases} \tag{4.14}$$

Where 'y' is the notation for both 1 and 0, $k \in \{0,1,\ldots,n-1\}$.

Example 4.3

Let n equals two then according to equations (4.13) and (4.14) the truth vector **c** can be calculated as follows.

$$c_3 = c_{11} = \odot \prod_k d_{11} = d_{11} = d_3$$

$$c_2 = c_{10} = \odot \prod_k d_{1y} = d_{10} \odot d_{11} = d_2 \odot d_3$$

$$c_1 = c_{01} = \odot \prod_k d_{y1} = d_{01} \odot d_{11} = d_1 \odot d_3$$

$$c_0 = c_{00} = \odot \prod_k d_{yy} = d_{11} \odot d_{10} \odot d_{01} \odot d_{00} = d_3 \odot d_2 \odot d_1 \odot d_0$$

Similarly this method can be used to convert from **c** to **d** with zero polarity using the identity $0 = x \odot \bar{x}$ in each term in equation (4.4). Hence the following equation is derived:

$$d_i = d_{i_{n-1}\ i_{n-2}\ \ldots\ i_0} = \odot \prod_l c_{l_{n-1}\ l_{n-2}\ \ldots l_0} \tag{4.15}$$

$$kj = \begin{cases} y & i_j = 0 \\ 1 & i_j = 1 \end{cases} \tag{4.16}$$

Observation:

If the number of off-set coefficients d_i in equation (4.13) for the corresponding S_i coefficient is odd (odd number of zeros), then coefficient c_i should be included for that DRM expansion. Otherwise it should not be included, because $0 \odot 0 = 1$. Therefore, the zero coefficients should be included for DRM expansion only.

Example 4.4

Calculate the truth vector **c** for the following three-variable function $f(x_2, x_1, x_0) = \Pi M(0,4,6,7)$.

The DRM coefficients are calculated using equations (4.13, 4.14).

$$c_7 = c_{111} = \odot \prod_k d_{111} = d_7 = 0$$

$$c_6 = c_{110} = \odot \prod_k d_{11y} = d_{111} \odot d_{110} = 0 \odot 0 = 1$$

$$c_5 = c_{101} = \odot \prod_k d_{1y1} = d_{111} \odot d_{101} = 0 \odot 1 = 0$$

$$c_4 = c_{100} = \odot \prod_k d_{1yy} = d_{111} \odot d_{110} \odot d_{101} \odot d_{100} = 0 \odot 0 \odot 1 \odot 0 = 0$$

$$c_3 = c_{011} = \odot \prod_k d_{y11} = d_{111} \odot d_{011} = 0 \odot 1 = 0$$

$$c_2 = c_{010} = \odot \prod_k d_{y1y} = d_{111} \odot d_{110} \odot d_{011} \odot d_{010} = 0 \odot 0 \odot 1 \odot 1 = 1$$

$$c_1 = c_{001} = \odot \prod_k d_{yy1} = d_{111} \odot d_{101} \odot d_{011} \odot d_{001} = 0 \odot 1 \odot 1 \odot 1 = 0$$

$$c_0 = c_{000} = \odot \prod_k d_{yyy} = d_{111} \odot d_{110} \odot d_{101} \odot d_{100} \odot d_{011} \odot d_{010} \odot d_{001} \odot d_{000}$$

$$= 0 \odot 0 \odot 1 \odot 0 \odot 1 \odot 1 \odot 1 \odot 0 = 1$$

Since the following coefficients are equal to zero (7, 5, 4, 3, 1) therefore, the DRM is

$$f(x_2, x_1, x_0) = \odot \prod(7,5,4,3,1)$$

$$= 0 \odot (x_1) \odot (x_1 + x_0) \odot (x_2) \odot (x_2 + x_1)$$

4.4 Conversion between d and c with any fixed polarity

To facilitate the use of equations (4.14, 4.16) for large Boolean functions the bitwise relationship between the subscripts of **d** and **c** is represented by using truth Table (4.1). The following equation is obtained using Karnaugh map of POS for the standard function $\psi_j = (\psi_{n-1}\ \psi_{n-2} \ldots\ \psi_0)$. A loop of 0-cells in a Karnaugh map generates a sum term.

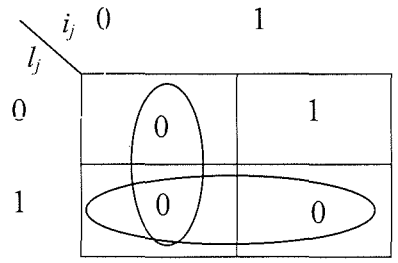
$$\Psi_j = i_j . \bar{l}_j \tag{4.17}$$

Where '.' is the normal AND operator and $\bar{l}$ represents the complement of the off-sets.

When ψ_j is equal to '0', that is all of its binary bits are '0', then equations (4.14) and (4.16) are satisfied. Using equation (4.17), it is possible to decide if a particular coefficient should be included for the conversion or not. Besides, only the off-set coefficients need to be calculated since $x = 1 \odot x$.

Table 4.1:

Map of the standard function ψ_j



Example 4.5

Calculate the truth vector **c** for the Dual Reed-Muller with zero polarity for the following three-variable function $f(x_2, x_1, x_0) = \Pi M(0,4,6,7)$.

The following coefficients for the truth vector **c** are calculated using equations (4.13) and (4.17).

$c_0 = c_{000} = 0 \cdot \bar{0} \odot 0 \cdot \bar{4} \odot 0 \cdot \bar{6} \odot 0 \cdot \bar{7} = 0 \odot 0 \odot 0 \odot 0 = 1$

$c_1 = c_{001} = 1 \cdot \bar{0} \odot 1 \cdot \bar{4} \odot 1 \cdot \bar{6} \odot 1 \cdot \bar{7} = 1 \cdot 7 \odot 1 \cdot 3 \odot 1 \cdot 1 \odot 1 \cdot 0 = 1 \odot 1 \odot 1 \odot 0 = 0$

$c_2 = c_{010} = 2\cdot\bar{0} \odot 2\cdot\bar{4} \odot 2\cdot\bar{6} \odot 2\cdot\bar{7} = 2\cdot7 \odot 2\cdot3 \odot 2\cdot1 \odot 2\cdot0 = 2\odot2\odot0 \odot0 = 1$

$c_3 = c_{011} = 3\cdot\bar{0} \odot 3\cdot\bar{4} \odot 3\cdot\bar{6} \odot 3\cdot\bar{7} = 3\cdot7 \odot 3\cdot3 \odot 3\cdot1 \odot 3\cdot0 = 3\odot3\odot1\odot0 = 0$

$c_4 = c_{100} = 4\cdot\bar{0} \odot 4\cdot\bar{4} \odot 4\cdot\bar{6} \odot 4\cdot\bar{7} = 4\cdot7 \odot 4\cdot3 \odot 4\cdot1 \odot 4\cdot0 = 4\odot0\odot0\odot0 = 0$

$c_5 = c_{101} = 5\cdot\bar{0} \odot 5\cdot\bar{4} \odot 5\cdot\bar{6} \odot 5\cdot\bar{7} = 5\cdot7 \odot 5\cdot3 \odot 5\cdot1 \odot 5\cdot0 = 5\odot1\odot1\odot0 = 0$

$c_6 = c_{110} = 6\cdot\bar{0} \odot 6\cdot\bar{4} \odot 6\cdot\bar{6} \odot 6\cdot\bar{7} = 6\cdot7 \odot 6\cdot3 \odot 6\cdot1 \odot 6\cdot0 = 6 \odot2\odot0\odot0 = 1$

$c_7 = c_{111} = 7\cdot\bar{0} \odot 7\cdot\bar{4} \odot 7\cdot\bar{6} \odot 7\cdot\bar{7} = 7\cdot7 \odot 7\cdot3 \odot 7\cdot1 \odot 7\cdot0 = 7 \odot3\odot1\odot0 = 0$

Where the decimal numbers are the value of the c's coefficients and the complemented decimal numbers are the complement of POS coefficients.
The results in example (4.5) and example (4.4) are equal. Therefore equation (4.17) can be used directly to find all the coefficients for zero polarity Dual Reed-Muller form.

Observation If the numbers of the included zeros are odd for a certain coefficients then c_i should be included and the value for this coefficient is zero; otherwise c_i is 1 and it should not be included.

Theorem 4.1 Given an off-set Dual Reed-Muller coefficient set R_p for an n-variable function with polarity **p**. A coefficient set with any other polarity **q** can be established through the process on R_p itself, using the following equation [63].

$$i \Leftarrow i \odot \mathrm{p} \odot \mathrm{p}'$$

Or

$$i \Leftarrow i \oplus p \tag{4.18}$$

Where '$\odot$' is bitwise XNOR operator, '$\oplus$' is the bitwise XOR operator and '$\Leftarrow$' is the assignment operator and p' = 0.

Equation (4.18) can also be used to convert from DRM to POS to find the d_i s from the c_i s coefficients.

Algorithm 4.1

The following steps can lead to the PPDRM expression from POS form.

Step 1: Store the complement of each term in the off-set.

Step 2: Use AND operation to AND each term from step '1' with the decimal value of each coefficient c_i.

Step 3: Count the number of zeros of each term for each coefficient c_i from step '2', if the total number is odd then that coefficient c_i should be included, otherwise it shouldn't be included.

Step 4: Repeat steps 2 and 3 for the rest of the coefficients.

Example 4.6

Convert a 3-variable function $f(x_2,x_1,x_0) = \Pi$ M(0,4,6,7) from POS form to the fixed polarity DRM with polarity **p** = 1= (001) in binary form.

Step 1, this function is converted to the POS expansion with polarity 1. The following new coefficients are obtained using equation (4.18).

$$0 \Leftarrow 0 \oplus 1 = 1$$
$$4 \Leftarrow 4 \oplus 1 = 5$$
$$6 \Leftarrow 6 \oplus 1 = 7$$
$$7 \Leftarrow 7 \oplus 1 = 6$$

Hence the new coefficients are:

$$f(x_2,x_1,\bar{x}_0) = \Pi\, M(1,5,7,6)$$

Step 2, Π M(1,5,7,6) is converted to DRM using equation (4.17).

$c_0 = c_{000} = 0\cdot\bar{1} \odot 0\cdot\bar{5} \odot 0\cdot\bar{7} \odot 0\cdot\bar{6} = 0 \odot 0 \odot 0 \odot 0 = 1$

$c_1 = c_{001} = 1\cdot\bar{1} \odot 1\cdot\bar{5} \odot 1\cdot\bar{7} \odot 1\cdot\bar{6} = 1\cdot 6 \odot 1\cdot 2 \odot 1\cdot 0 \odot 1\cdot 1 = 1 \odot 1 \odot 1 \odot 0 = 0$

$c_2 = c_{010} = 2\cdot\bar{1} \odot 2\cdot\bar{5} \odot 2\cdot\bar{7} \odot 2\cdot\bar{6} = 2\cdot 6 \odot 2\cdot 2 \odot 2\cdot 0 \odot 2\cdot 1 = 2 \odot 2 \odot 0 \odot 0 = 1$

$c_3 = c_{011} = 3\cdot\bar{1} \odot 3\cdot\bar{5} \odot 3\cdot\bar{7} \odot 3\cdot\bar{6} = 3\cdot 6 \odot 3\cdot 2 \odot 3\cdot 0 \odot 3\cdot 1 = 2 \odot 2 \odot 0 \odot 1 = 0$

$c_4 = c_{100} = 4\cdot\bar{1} \odot 4\cdot\bar{5} \odot 4\cdot\bar{7} \odot 4\cdot\bar{6} = 4\cdot 6 \odot 4\cdot 2 \odot 4\cdot 0 \odot 4\cdot 1 = 4 \odot 0 \odot 0 \odot 0 = 0$

$c_5 = c_{101} = 5\cdot\bar{1} \odot 5\cdot\bar{5} \odot 5\cdot\bar{7} \odot 5\cdot\bar{6} = 5\cdot 6 \odot 5\cdot 2 \odot 5\cdot 0 \odot 5\cdot 1 = 4 \odot 0 \odot 0 \odot 1 = 1$

$c_6 = c_{110} = 6\cdot\bar{1} \odot 6\cdot\bar{5} \odot 6\cdot\bar{7} \odot 6\cdot\bar{6} = 6\cdot 6 \odot 6\cdot 2 \odot 6\cdot 0 \odot 6\cdot 1 = 6 \odot 2 \odot 0 \odot 0 = 1$

$c_7 = c_{111} = 7\cdot\bar{1} \odot 7\cdot\bar{5} \odot 7\cdot\bar{7} \odot 7\cdot\bar{6} = 7\cdot 6 \odot 7\cdot 2 \odot 7\cdot 0 \odot 7\cdot 1 = 6 \odot 2 \odot 0 \odot 1 = 0$

$f(x_2,x_1,\bar{x}_0) = \odot\Pi\,(7,4,3,1)$

The sum terms in this canonical form can be generated using equation (3.10) as follows.

$$[0\ \ x_2] +\!+ [0\ \ x_1] +\!+ [0\ \ \bar{x}_0]$$

$$= \left[0 \quad \bar{x}_0 \quad x_1 \quad x_1+\bar{x}_0 \quad x_2 \quad x_2+\bar{x}_0 \quad x_2+x_1 \quad x_2+x_1+\bar{x}_0\right]$$

Finally, the Fixed Polarity Dual Reed-Muller form with polarity (p = 1) is obtained by using equation (4.11) as follows:

$$f(x_2,x_1,\bar{x}_0) = 0 \odot x_1+\bar{x}_0 \odot x_2 \odot x_2+x_1$$

Example 4.7

For a three-variable Boolean function $f\ (x_2,x_1,x_0)$ = Π M(0,4,6,7) find the expression of PPDRM **p** = 0.

Applying Algorithm 1

Step 1: Store the complement of each term in the off-set; this gives the following results.

$\bar{0},\bar{4},\bar{6},\bar{7} = 7,3,1,0$

Step 2: The first coefficient of PPDRM (c_0) is calculated using equations (4.17) as follows:

$$c_0 = 0\cdot\bar{0} \odot 0\cdot\bar{4} \odot 0\cdot\bar{6} \odot 0\cdot\bar{7} = 0\cdot7 \odot 0\cdot3 \odot 0\cdot1 \odot 0\cdot0 = 0 \odot 0 \odot 0 \odot 0$$

Step 3: Count the number of zeros in the last expression, which is four. Since the number of zeros is even then this coefficient (c_0) should not be included for PPDRM. Hence

$c_0 = 1$

Similarly, the rest of the coefficients are obtained as follows:

$$c_1 = 1\cdot\bar{0} \odot 1\cdot\bar{4} \odot 1\cdot\bar{6} \odot 1\cdot\bar{7} = 1\cdot7 \odot 1\cdot3 \odot 1\cdot1 \odot 1\cdot0 = 1 \odot 1 \odot 1 \odot 0 = 0$$

$$c_2 = 2\cdot\bar{0} \odot 2\cdot\bar{4} \odot 2\cdot\bar{6} \odot 2\cdot\bar{7} = 2\cdot7 \odot 2\cdot3 \odot 2\cdot1 \odot 2\cdot0 = 2 \odot 2 \odot 0 \odot 0 = 1$$

$$c_3 = 3\cdot\bar{0} \odot 3\cdot\bar{4} \odot 3\cdot\bar{6} \odot 3\cdot\bar{7} = 3\cdot7 \odot 3\cdot3 \odot 3\cdot1 \odot 3\cdot0 = 3 \odot 3 \odot 1 \odot 0 = 0$$

$$c_4 = 4\cdot\bar{0} \odot 4\cdot\bar{4} \odot 4\cdot\bar{6} \odot 4\cdot\bar{7} = 4\cdot7 \odot 4\cdot3 \odot 4\cdot1 \odot 4\cdot0 = 4 \odot 0 \odot 0 \odot 0 = 0$$

$$c_5 = 5\cdot\bar{0} \odot 5\cdot\bar{4} \odot 5\cdot\bar{6} \odot 5\cdot\bar{7} = 5\cdot7 \odot 5\cdot3 \odot 5\cdot1 \odot 5\cdot0 = 5 \odot 1 \odot 1 \odot 0 = 0$$

$$c_6 = 6\cdot\bar{0} \odot 6\cdot\bar{4} \odot 6\cdot\bar{6} \odot 6\cdot\bar{7} = 6\cdot7 \odot 6\cdot3 \odot 6\cdot1 \odot 6\cdot0 = 6 \odot 2 \odot 0 \odot 0 = 1$$

$$c_7 = 7\cdot\bar{0} \odot 7\cdot\bar{4} \odot 7\cdot\bar{6} \odot 7\cdot\bar{7} = 7\cdot7 \odot 7\cdot3 \odot 7\cdot1 \odot 7\cdot0 = 7 \odot 3 \odot 1 \odot 0 = 0$$

The truth vector **c** for zero polarity is given as follows:

$$\mathbf{c} = [7, 5, 4, 3, 1]$$

The sum terms in this canonical form can be generated using equation (4.11) as follows:

$$[0\ x_2] ++ [0\ x_1] ++ [0\ x_0] = [0\ x_0\ \ x_1\ \ x_1 + x_0\ \ x_2\ \ x_2 + x_0\ \ x_2 + x_1\ \ x_2 + x_1 + x_0]$$

Using equation (4.11) to get the PPDRM form will lead to

$$f(x_2,x_1,x_0) = (0) \odot (x_1) \odot (x_1+x_0) \odot (x_2) \odot (x_2+x_1)$$

Example 4.8

Compute Reed-Muller coefficients with zero polarity for a four-variable Boolean function $f(x_3,x_2,x_1,x_0) = \Pi$ M(0,3,4,5,6,8,10,11,13,14,15) and find the PPDRM expression.

c_0 can be calculated by using equation (4.17), for the first off-set coefficient '1', we have

$$\Psi = 0.\bar{0}$$
$$\Psi = 0000.1111 = 0000$$

Because all of Ψ_j are 0, this coefficient should be included. Then move to the second off-set coefficient which is 2, we have

$$\Psi = 0.\bar{3}$$
$$\Psi = 0000.1100 = 0000$$

Repeat the same procedure for the rest of off-set coefficients. After finishing all the 11 off-set coefficients, count the number of zero for each operation. This gives 11, since this odd number therefore, c_0 is included because it is equals to 0. Repeat the same procedure for the reset of the coefficients. The final Dual Reed-Muller form is:

$$f(x_3, x_2, x_1, x_0) = \odot \prod(15,12,9,7,6,5,3,1,0)$$

This same example was run using the programmed develop in this chapter, the following results were obtained.

$$f(x_3, x_2, x_1, x_0) = \odot \prod(15,12,9,7,6,5,3,1,0)$$

This result agrees with the previous results.

4.5 Results

In this section, experimental results are presented using algorithm 4.1. The proposed algorithm is implemented in C language and the programs compiled using Borland C++ compiler. The program was tested on a personal computer with Pentium 4 processor of 2.4 GHz CPU and 512 MB of RAM under Window operating system. The algorithm was applied to several MCNC benchmarks and some random functions. Table 4.2 shows the results obtained from converting POS coefficients into DRM coefficients. Where name denotes the name of circuit, n denotes the number of variables, Init terms denote the number of terms in POS form, DRM terms denote the number of terms in DRM form, the execution time (CPU Time (s)) is time required to calculate the coefficients of the Fixed Polarity Dual Reed-Muller form the coefficients of the Product of Sums and it is given in seconds. For most of the circuits with n less than 16 the CPU time is less than 1 seconds. The CPU time depends on the variable number n and the initial number of terms (number of off-set coefficients). For large Boolean functions, there are many coefficients and they should be accessed once. This algorithm can be improved by ordering the off-set coefficients in advance. This is can be achieved by introducing a multiple segment technique, then the execution time will be improved and it could handle large number of variables.

For incompletely specified Boolean functions,' don't care' are set to '1'. The experimental results obtained in Table 4.2 reflect the efficiency of the algorithm.

Table 4.2:

Conversion results from POS to PPDRM form

Name	*n*	Init. terms in POS	DRM Terms	CPU Time (s)
Con1	7	88	9	0.000
Rd84	8	136	37	0.050
Apex4	9	534	181	0.000
Clip	9	480	92	0.000
Ex1010	10	142	480	0.010
F12†	12	1984	365	0.02
F13†	13	4152	127	0.100
F14†	14	16172	625	0.711
F15†	15	5792	3100	0.540
spla	16	5348	517	0.931
Table5	17	28552	3359	9.845

4.6 Summary

This chapter has introduced the basics matrices that are needed to convert between Product of Sums and Positive and Fixed polarity Dual Reed-Muller forms. Such matrices are needed to construct the transformation matrix for the Dual Reed-Muller expressions. This chapter has introduced also new formulas to calculate the coefficients of the Dual Reed-Muller directly from the off-set of the Product of Sums. Therefore, this technique is fast and does not need to store the transformation matrix.

Chapter 5

Exact minimization of Dual Reed-Muller expressions

5.1 Introduction

In the optimization of FPDRM expansions, functions with different polarities are usually calculated directly from POS expressions [64, 65]. A new algorithm is presented in this chapter to generate all the polarity sets from any polarity set **q** for a single output Boolean function. This technique is used to find the best polarity of FPDRM among the 2^n fixed polarities. The algorithm is based on the dual property and the Gray code strategy. Time efficiency and computing speed are achieved in this technique because the information in finding FPDRM expansion of one polarity is utilized by others. Two-fixed polarities can be derived from each other without the need to go back to the original Boolean function in the POS form, if the two polarities are dual.

Definition 5.1 Polarity vector $(p_{n-1},p_{n-2},\ldots,p_0)$ for a FPDRM of an n-variable Boolean function is a binary vector with n elements, where $p_i = 0$ indicates the variable x_i in

an un-complemented form (x_i), while $p_i = 1$ indicates the variable x_i in the complemented form ($\bar{x}_i$).

Definition 5.2 Two polarities are defined to be dual polarities if they reveal the following property: the n-bit binary strings of these two polarities have n-1 bits in common and only one bit is different [95, 96].

5.2 Exact minimization of the Fixed Polarity DRM forms

A new algorithm is presented in this chapter to generate all of the polarity sets from any polarity set **q**, without using the direct method in converting from POS to FPDRM forms [64, 65]. Time efficiency is achieved in this technique because the information utilized in finding DRM expansion of one polarity is utilized by others. Two-fixed polarities can be derived from each other without the need to go back to the original Boolean function, if the two polarities are dual [95, 96].

Corollary 5.1 The DRM with a fixed polarity q_j can be derived from DRM with a fixed polarity p_j, where p_j, q_j are dual polarities and j is the permuting bit.

Proof

From equation (5.1) with polarity $\mathbf{p} = p_j$, DRM is given as:

$$f(x_{n-1},x_{n-2},\ldots,x_0) = \odot \prod_{i=0}^{2^n-1} (c_i + S_i) \tag{5.1}$$

$$f^{(p_j)}(x_{n-1}, x_{n-2}, \ldots, x_0) = \odot \prod_{i=0}^{2^n-1} (c_i^{p_j} + S_i^{p_j}) \tag{5.2}$$

Where

$$S_i^{p_j} = \sum_{k=n-1}^{0} \bullet^{i_k} x_k = x_{n-1} \bullet^{i_{n-1}} + x_{n-2} \bullet^{i_{n-2}} + \ldots + x_0 \bullet^{i_0} \tag{5.3}$$

p_j, is any given polarity = (0,..., 2^n-1), $S_i^{(p_j)}$ are the sum terms for the particular polarity p_j.

In order to change polarity from p_j to q_j, equation (5.3) can be expressed as

$$S_i^{p_j} = \sum_{k=n-1}^{j+1} (\dot{x}_k)^{i_k} + (\dot{x}_j)^{i_j} + \sum_{k=j-1}^{0} (\dot{x}_k)^{i_k} \tag{5.4}$$

Each variable x_k in equation (5.4) will remain as it is, except variable x_j is replaced by the following identity $(\dot{x}_j)^{i_j} = 0 \odot (\bar{\dot{x}}_j)^{i_j}$. Therefore, equation (5.4) becomes the following equation for $S_i^{q_j}$

$$S_i^{q_j} = \sum_{k=n-1}^{j+1} (\dot{x}_k)^{i_k} + (0 \odot (\bar{\dot{x}}_j)^{i_j}) + \sum_{k=j-1}^{0} (\dot{x}_k)^{i_k} \tag{5.5}$$

By using the following property:

$$A + (0 \odot \bar{B}) + C = (A+C) + (0 \odot \bar{B}) = (A+C+0) \odot (A+C+\bar{B})$$

Hence

$$S_i^{q_j} = (\sum_{k=n-1}^{j+1} (\dot{x}_k)^{i_k} + \sum_{k=j-1}^{0} (\dot{x}_k)^{i_k}) \odot (\sum_{k=n-1}^{j+1} (\dot{x}_k)^{i_k} + (\bar{\dot{x}}_j)^{i_j} + \sum_{k=j-1}^{0} (\dot{x}_k)^{i_k}) \tag{5.6}$$

By substituting (5.6) in (5.2), we obtain the following FPDRM expansion for q_j with $\mathbf{p} = q_j$.

$$f^{q_j} = \odot \prod_{i=0}^{2^n-1} (c_i^{q_j} + [(\sum_{k=n-1}^{j+1} (\dot{x}_k)^{i_k} + \sum_{k=j-1}^{0} (\dot{x}_k)^{i_k}) \odot (\sum_{k=n-1}^{j+1} (\dot{x}_k)^{i_k} + (\bar{\dot{x}}_j)^{i_j} + \sum_{k=j-1}^{0} (\dot{x}_k)^{i_k})])$$

To convert polarity p_j to polarity q_j, each sum in the FPDRM with polarity p_j is converted into a binary string. A zero is placed in the binary string if the variable is present and a one if the variable is absent. The new term is generated, by copying all the binary string except for bit j. If bit j is zero change it to one. Duplicate terms are deleted according to the rule B $\odot$ B = 1. Based on this, the following algorithm is developed.

5.3 Conversion from polarity p to polarity q

Algorithm 5.1

This algorithm converts between the polarities and identifies the polarity number with the least number of Sum terms in the FPDRM functions. The following steps shall be used to derive the coefficient set Ψq from the dual set Φp. The steps are repeated for the rest of the polarities till the best polarity is obtained.

Step1: Use Algorithm 4.1 to calculate the coefficients for PPDRM function. Set Φ_{min} = the number of off-set coefficients for polarity p_j.

Step 2: Determine the next polarity q_j in Gray code order, where polarities p_j and q_j are dual and differ in bit j only.

Step 3: Converts the sum terms for polarity p_j into a binary string. By replacing each variable by '0' if the variable is present in the sum term or by '1' if it is absent.

Step 4: For each term in polarity p_j, generate a new term if bit j of the binary string is '0'. Replace bit j with '1' and copy all others bits to generate the new term.

Step 5: Delete common pairs between original strings and newly generated strings because B $\odot$ B = 1.

Step 6: The unaffected strings are the product terms of the new polarity q_j.

Step 7: Count the total number of zero coefficients Ψ_{min} for polarity q_j.
If $\Psi_{min} < \Phi_{min}$ then $\Phi_{min} := \Psi_{min}$.

Step 8: Stop if all polarities have been checked. Otherwise go to step 2.

Example 5.1
Find the best polarity for a three-variable function $f(x_2,x_1,x_0) = \Pi$ M(0,4,6,7).

Step 1, the following results are obtained for PPDRM using algorithm 4.1.

$$f(x_2,x_1,x_0) = \odot\Pi\,(7,5,4,3,1) \tag{5.7}$$

In order to find all polarities for DRM expansion, a Gray code sequence is generated for $n = 3$, then Algorithm 5.1 is applied to find the best polarity with the least number of Sum terms.
The following Gray code is generated for $n = 3$:

000 – 001 – 011 – 010 – 110 – 111 – 101 - 100

Count the number of coefficients from step 1, and set $\Phi p = 5$ since this the first step in the procedure set also Φ_{min} to Φp.
Step 2, since polarity '0' = (000) and polarity '1' = (001) are dual polarities. Hence DRM in polarity '1' can be derived directly from equation (5.7) using Algorithm 5.1 as shown in Table 5.1 where the altered bit is at j equals 0.

Table 5.1:

Derivation of DRM for Polarity 1

Polarity 0 $x_2x_1x_0$	New terms	Polarity 1 $x_2x_1\bar{x}_0$
111		111
~~101~~		100
100	~~101~~	011
011		001
001		

Therefore, the coefficients for the FPDRM with **p** = 1 are the following:

$$f(x_2,x_1,\bar{x}_0) = \odot\Pi\,(7,4,3,1)$$

Where the number of terms is Ψ_{min} = 4, since $\Psi_{min} < \Phi_{min}$ then record the corresponding polarity as $P_{min} = q_j = 1$ and $\Phi_{min} = 4$.

Repeat step 2 to convert from polarity '1'= (001) to '3' = (011), with the altered bit j = 1 as shown in Table 5.2.

Table 5.2:

Derivation of DRM for Polarity 3

Polarity 1 $x_2x_1\overline{x}_0$	New terms	Polarity 3 $x_2\overline{x}_1\overline{x}_0$
111		111
100	110	110
~~011~~		100
001	~~011~~	001

Therefore, the coefficients for the FPDRM with **p** = 3 are the following:

$$f(x_2, \overline{x}_1, \overline{x}_0) = \odot\Pi\ (7,6,4,1)$$

Where the number of terms is Ψ_{min} = 4, since Ψ_{min} = Φ_{min} then go to the next polarity.

Repeat step 2 to convert from polarity '3'= (011) to '2' = (010), with the altered bit j = 0 as shown in Table 5.3.

Table 5.3:

Derivation of DRM for Polarity 2

Polarity 3 $x_2\overline{x}_1\overline{x}_0$	New terms	Polarity 2 $x_2\overline{x}_1x_0$
~~111~~		110
110	~~111~~	101
100	101	100
001		001

Therefore, the coefficients for the FPDRM with **p** = 2 are the following

$$f(x_2, \bar{x}_1, x_0) = \odot\Pi\,(6,5,4,1)$$

$\Psi_{min} = 4$, since $\Psi_{min} = \Phi_{min}$ then go to the next polarity.

Repeat step 2 to convert from polarity '2'= (010) to '6' = (110), with the altered bit j = 2 as shown in Table 5.4.

Table 5.4:

Derivation of DRM for Polarity 6

Polarity 2 $x_2\bar{x}_1x_0$	New terms	Polarity 6 $\bar{x}_2\bar{x}_1x_0$
110		110
~~101~~		100
100		001
001	~~101~~	

Therefore, the coefficients for the FPDRM with **p** = 6 are the following

$$f(\bar{x}_2, \bar{x}_1, x_0) = \odot\Pi\,(6,4,1)$$

$\Psi_{min} = 3$, since $\Psi_{min} < \Phi_{min}$ then record the corresponding polarity as $P_{min} = q_j = 6$ and change Φ_{min} to 3.

Repeat step 2 to convert from polarity '6'= (110) to '7' = (111), with the altered bit j = 0 as shown in Table 5.5.

Table 5.5:

Derivation of DRM for Polarity 7

Polarity 6 $\bar{x}_2\bar{x}_1x_0$	New terms	Polarity 7 $\bar{x}_2\bar{x}_1\bar{x}_0$
110	111	111
100	101	110
001		101
		100
		001

Therefore, the coefficients for the FPDRM with **p** = 7 are the following

$$f(\bar{x}_2,\bar{x}_1,\bar{x}_0) = \odot\Pi\,(7,6,5,4,1)$$

Ψ_{min} = 5, since $\Psi_{min} > \Phi_{min}$ then go to the next polarity.

Repeat step 2 to convert from polarity '7'= (111) to '5' = (101), with the altered bit j = 1 as shown in Table 5.6.

Table 5.6:

Derivation of DRM for Polarity 5

Polarity 7 $\bar{x}_2\bar{x}_1\bar{x}_0$	New terms	Polarity 5 $\bar{x}_2 x_1 \bar{x}_0$
~~111~~		101
~~110~~		100
101	~~111~~	011
100	~~110~~	001
001	011	

Therefore, the coefficients for the FPDRM with **p** = 5 are the following

$$f(\bar{x}_2, x_1, \bar{x}_0) = \odot\Pi\,(5,4,3,1)$$

Repeat step 2 to convert from polarity '5'= (101) to '4' = (100), with the altered bit j = 0 as shown in Table 5.7.

Table 5.7:

Derivation of DRM for Polarity 4

Polarity 5 $\bar{x}_2 x_1 \bar{x}_0$	New terms	Polarity 4 $\bar{x}_2 x_1 x_0$
~~101~~		100
100	~~101~~	011
011		001
001		

The process is terminated at this point, because $\Psi min > \Phi_{min}$ (4 > 3), ttherefore the best polarity for this function is **p** = 6 = (110) with 3 terms.

$$f(\bar{x}_2, \bar{x}_1, x_0) = \odot\Pi\,(6,4,1)$$

The sum terms for this canonical form can be generated as follows:

$$[0 \;\; \bar{x}_2] + +[0 \;\; \bar{x}_1] + +[0 \;\; x_0] = [0 \;\; x_0 \;\; \bar{x}_1 \;\; (\bar{x}_1 + x_0) \;\; \bar{x}_2 \, (\bar{x}_2 + x_0) \;\; (\bar{x}_2 + \bar{x}_1)$$

$$(\bar{x}_2 + \bar{x}_1) \;\; (\bar{x}_2 + \bar{x}_1 + x_0)]$$

Using equation (4.11) the following equation is obtained for FPDRM form with polarity number is equal to 6.

$$f(\bar{x}_2, \bar{x}_1, x_0) = [0 \; x_0 \;\; \bar{x}_1 \;\; \bar{x}_1 + x_0 \;\; \bar{x}_2 \;\; \bar{x}_2 + x_0 \;\; \bar{x}_2 + \bar{x}_1 \;\; \bar{x}_2 + \bar{x}_1 + x_0]^{\circ} \mathbf{c}$$

Where the truth vector **c** = [1 0 1 0 1 1 0 1]
Hence,

$$f(\bar{x}_2, \bar{x}_1, x_0) = x_0 \odot (\bar{x}_1 + x_0) \odot (\bar{x}_2 + \bar{x}_1)$$

5.4 Results

In this section, experimental results are presented using algorithms (4.1) and (5.1). The proposed algorithms are implemented in C language and the programs compiled using Borland C++ compiler. The programs were tested on a personal computer with Pentium 4 processor of 2.4 GHz CPU and 512 MB of RAM under Window operating system. The algorithms were applied to several Random functions as well as MCNC benchmarks. Table 5.8 shows the results obtained from converting PPDRM (p = 0)

coefficients into FPDRM coefficients. Where name denotes the name of circuit, Input No denotes the number of variables, Init terms denote the number of terms in PPDRM form, Terms denotes the minimum number of terms required for a fixed polarity Dual Reed-Muller forms (FPDRM) terms, the execution time (CPU Time (s)) is time required to find the best polarity starting from the zero polarity and Improv (%) represents the average saving in terms between polarity zero and the best polarity which is based on the following formula:

$$\text{Improv.Terms} = \frac{\text{Initi.Terms} - \text{DRM.Terms}}{\text{Initi.Terms}} \times 100\%$$

The execution time (CPU Time (s)) depends on the variable number n as well as the initial number of terms. For incompletely specified Boolean functions,' don't care' are set to '1'. The experimental results obtained in Table 5.8 reflect the efficiency of the algorithm. The average saving in terms of number of terms is about 30 percent. We are able to minimize functions with up to 14 variables. Memory requirements make it impossible to minimize functions with more than 14 variables. To our knowledge there are no publications on this topic to compare with.

Table 5.8:

Optimization results based on algorithm 2

Name	Input No.	Terms under polarity 0	FPDRM		Improv. (%)	CPU Time (s)
			Best polarity	Terms		
Con1	7	9	7	8	11	0.000
Rd84	8	37	1	29	21	0.000
Apex4	9	181	2	170	6	0.100
Clip	9	92	496	63	32	0.031
Ex1010	10	480	944	413	14	1.061
F12†	12	365	804	127	65	3.455
F13†	13	127	704	65	48	4.16
F14†	14	625	5587	66	89	260.07

†Randomly generated Boolean functions.

5.5 Summary

Boolean functions in Product of Sums forms can be represented by Fixed Polarity Dual Reed-Muller forms. Each FPDRM form can be identified with a distinct polarity, were each variable appears in the complemented or un- complemented form but not both. Therefore, an algorithm is required to find the best polarity of the FPDRM forms among the 2^n fixed polarities, without converting directly between POS and FPDRM forms for each polarity. Hence a new algorithm is presented in this chapter. The algorithm is used to generate all the polarity sets from any polarity set **q** for a single output Boolean function. This algorithm is based on the dual property and the Gray code strategy. Two polarities are dual if they have n-1 bits in common and only one bit is different.

Hence, all of the fixed polarities can be derived from each other without starting from the original Boolean function in the POS form. Therefore, time efficiency and computing speed are achieved in this technique.

Chapter 6

Optimal Polarity for Dual Reed-Muller Expressions

6.1 Introduction

In this chapter we present two algorithms, which can be used to convert from POS to FPDRM and find the optimal polarity for large number of variables. The first algorithm is used to compute the coefficients of PPDRM or FPDRM directly from the truth table of POS, without the use of mapping techniques [65] and without the use of matrix operation [64]. This algorithm is also used to compute the coefficients of POS from PPDRM or FPDRM. The second algorithm will find the optimal polarity among the 2^n different polarities for large n-variable functions, without generating all of the polarity sets. This algorithm is based on separating the truth vector of POS and the use of sparse techniques, which will lead to the optimal polarity. Time efficiency and computing speed are thus achieved in this technique.

Definition 6.1 Polarity vector $(p_{n-1},p_{n-2},\ldots,p_0)$ for a FPDRM of an n-variable Boolean function is a binary vector with n elements, where $p_i = 0$ indicates the variable x_i in

an un-complemented form (x_i), while $p_i = 1$ indicates the variable $\bar{x}_i$ in the complemented form x_i.

Property 6.1 For an n-variable Boolean function, there are 2^n FPDRM expansions corresponding to 2^n different polarity numbers. Each of such expansions is a canonical representation of a completely specified Boolean function.

6.2 Conversion Algorithms

6.2.1 Conversion from POS to FPDRM

To compute the coefficients for FPDRM expansion (**c**) from the coefficients of the POS expansion (**d**), the following principles and derivation are developed. An n-variable Boolean function can be expressed as:

$$f(x_{n-1}, x_{n-2}, ..., x_0) = \prod_{i=0}^{2^n-1} (d_i + M_i) \tag{6.1}$$

Equation (6.1) can be represented as

$$f(x_{n-1},x_{n-2},\ldots,x_0) = (d_0 + x_{n-1} + x_{n-2} + \ldots x_0) \cdot (d_1 + x_{n-1} + x_{n-2} + \ldots \bar{x}_0) \cdot (d_2 + x_{n-1} + x_{n-2} + \ldots \bar{x}_1 + x_0) \cdot \ldots \cdot (d_{2^n-1} + \bar{x}_{n-1} + \bar{x}_{n-2} + \ldots \bar{x}_0) \tag{6.2}$$

In equation (6.1) if all Maxterms are ANDed for each different combination of the inputs the result will be '0' and if all Maxterms are XNORed for each different combination of the inputs variables the result will be also a '0', because for each combination of the inputs one of the Maxtrems will be '0' and the rest will be '1'. Hence equation (6.1) can be written as in Equation (6.3) by replacing each AND gate by XNOR gate.

$$f(x_{n-1},x_{n-2},\ldots,x_0) = \odot \prod_{i=0}^{2^n-1} (d_i + M_i) \tag{6.3}$$

$$f(x_{n-1},x_{n-2},\ldots,x_0) = (d_0 + x_{n-1} + x_{n-2} + \ldots x_0) \odot (d_1 + x_{n-1} + x_{n-2} + \ldots \bar{x}_0)$$

$$\odot (d_2 + x_{n-1} + x_{n-2} + \ldots \bar{x}_1 + x_0) \odot \ldots \tag{6.4}$$

$$\odot (d_2{}^n{}_{-1} + \bar{x}_{n-1} + \bar{x}_{n-2} + \ldots \bar{x}_0)$$

Equation (6.4) can be described in terms of a coefficient truth vector. The coefficient vector for an n-variable Boolean function can be represented as:

$$\mathbf{T} = [d_0, d_1, ..., d_{2^n-1}] \tag{6.5}$$

The elements of the truth vector (**T**) are placed in the order of decimal equivalent binary coding of the sum terms.

Examining the general form in equation (6.4), half of the sum terms in the truth vector **T** include variable x_i in true form and the second half include variable x_i in complemented form. Therefore, the truth vector (**T**) for any Boolean function in POS form can be separated into two rows for each variable x_i and the result is stored in the separation matrix $T(x_i)$. The first row of the separation matrix $T(x_i)$ contains Maxterms with variable x_i in un-complemented form, while the second row of $T(x_i)$ contains Maxterms with variable x_i in complemented [97]. The elements in the truth vector and the separation matrix $T(x_i)$ are arranged into groups of four bits for convenient. The following example illustrates the separation process.

Example 6.1

Construct the truth vector **T** for a 4-variable function $f(x_3,x_2,x_1,x_0) = \Pi$ M(0,4,6,7,11,15) and use the truth vector **T** to generate the separation matrix for each variable x_i.

The truth vector **T** has 2^n elements. Each Maxterm correspond to '0's in the truth vector **T**. Hence **T** is presented as follows:

$$\mathbf{T} = [0111\ 0100\ 1110\ 1110]$$

To generate the first matrix $T(x_3)$, the truth vector **T** is separated around variable x_3 int two equal parts. The first part corresponds to un-complemented part, while the second part to the complemented part. This is can be done according to the following formula:

$$\text{Number of Divisions} = \frac{2^n}{2^{n-i}}$$

Where n is the number of variables and i is the number for variable x_i.

Therefore, $n = 4$ and $i = 3$.

Hence,

The un- complemented for x_3 is:

$$[0111\ 0100]$$

And for the complemented is:

$$[1110\ 1110]$$

Therefore,

$$T(x_3) = \begin{bmatrix} 0111 & 0100 \\ 1110 & 1110 \end{bmatrix} \begin{matrix} x_3 \\ \bar{x}_3 \end{matrix}$$

For x_2 the truth vector is divided as follows:

$$\text{Number of Divisions} = 2^4/2^2 = 4$$

Therefore, the truth vector is divided into four equal parts.
The un- complemented for x_2 is:

[1110] and [1110]

This gives:

[1110 1110]

While for the complemented part is:

[0100] and [1110]

Therefore,

$$T(x_2) = \begin{bmatrix} 0111 & 1110 \\ 0100 & 1110 \end{bmatrix} \begin{matrix} x_2 \\ \bar{x}_2 \end{matrix}$$

Similarly the separation matrices for x_1 and x_0 are as follows:

$$T(x_1) = \begin{bmatrix} 0101 & 1111 \\ 1100 & 1010 \end{bmatrix} \begin{matrix} x_1 \\ \bar{x}_1 \end{matrix}$$

$$T(x_0) = \begin{bmatrix} 0100 & 1111 \\ 1110 & 1010 \end{bmatrix} \begin{matrix} x_0 \\ \bar{x}_0 \end{matrix}$$

To replace any complemented variable $\bar{x}_i$ by un-complemented variable x_i in equation (6.4) the following identity $\bar{x}_i = (0 \odot x_i)$ is used. The following result is obtained

$$(a + \bar{x}_i) \odot (b + x_i) = [a + (0 \odot x_i)] \odot (b + x_i)$$

$$= [(a + 0) \odot (a + x_i)] \odot (b + x_i)$$

$$= a \odot [(a + x_i) \odot (b + x_i)]$$

However

$$[(a + x_i) \odot (b + x_i)] = [(a \odot b) + x_i]$$

This can be verified as follows

$$\overline{[(a + x_i) \odot (b + x_i)]} = \bar{a}\,\bar{x}_i \oplus \bar{b}\,\bar{x}_i$$

$$= \bar{x}_i\,(\bar{a} \oplus \bar{b})$$

Where '$\oplus$' is XOR operator.

Complementing the last expression, the following is obtained

$$\overline{\bar{x}_i\,(\bar{a} \oplus \bar{b})} = [(a \odot b) + x_i]$$

Therefore,

$$[(a + x_i) \odot (b + x_i)] = [(a \odot b) + x_i]$$

Hence

$$(a + \bar{x}_i) \odot (b + x_i) = a \odot [(a \odot b) + x_i] \tag{6.6}$$

Examining equation (6.6), the coefficients of the un-complemented part of variable x_i take a new form. The new coefficient is (a XNOR b), while the coefficient for the complemented part will remain the same. Similarly, to convert un-complemented form to complemented form the following principal is applied.

Each un-complemented variable x_i is replaced by $0 \odot \bar{x}_i$.

$$(a + \bar{x}_i) \odot (b + x_i) = (a + \bar{x}_i) \odot [(b + (0 \odot \bar{x}_i)]$$

$$= (a + \bar{x}_i) \odot [(b + 0) \odot (b + \bar{x}_i)]$$

$$= [(a + \bar{x}_i) \odot (b + \bar{x}_i)] \odot b$$

By taking the complement of the following expression

$$\overline{[(a + \bar{x}_i) \odot (b + \bar{x}_i)]} = \bar{a}x_i \oplus \bar{b}x_i$$

$$= x_i (\bar{a} \oplus \bar{b})$$

Taking the complement for the last expression will give the following result

$$\overline{x_i (\bar{a} \oplus \bar{b})} = \bar{x}_i + (a \odot b)$$

Hence

$$[(a + \bar{x}_i) \odot (b + \bar{x}_i)] = \bar{x}_i + (a \odot b)$$

Therefore,

$$(a + \bar{x}_i) \odot (b + x_i) = b \odot [(a \odot b) + \bar{x}_i] \quad (6.7)$$

Inspecting equation (6.7), the coefficient of the true form stays as it is while the coefficient of the complemented form is replaced by (a XNOR b).

Algorithm 6.1

A computer algorithm has been developed based on the previous theory as shown in the following steps.

Converting from POS to FPDRM

Algorithm 6.1.a

The following steps are used to from POS to FPDRM forms:

Step 1: Store the coefficients of the POS in the truth vector (**T**).

Step 2: Construct T(x_i) matrix from **T** vector for each variable x_i. The first row of T(x_i) matrix contains the coefficients of the Maxterms for variable x_i in un-complemented form. While the second row of T(x_i) contains the coefficients of the Maxterms with variable x_i in the complemented form.

Step 3: The elements in the first and second rows of T(x_i) matrix are group together using XNOR operation and the result is stored in vector (**N**).

Step 4: If the required polarity for x_i variable is '0' then replace the contents of each true variable x_i in the truth vector **T** by the contents of vector **N**.

Step 5: If the required polarity for x_i variable is '1' then replace the contents of each complemented part of the $\overline{x_i}$ variable in the truth vector **T** by the contents of the un-complemented part of the x_i variable and store the result **N** in place of un-complemented part of x_i variable in **T**.

Step 6: Repeat the previous steps for the rest of the variables by using the new truth vector from step '5' or '6' depending on the polarity.

Step 7: The zero elements stored in the last **T** vector are the coefficients for that particular polarity of the FPDRM.

6.2.2 Conversion from FPDRM to POS

Algorithm 6.1.b

To find the POS's coefficients from the FPDRM's coefficients, step '5' in Algorithm 6.1.a is changed to the following step:

If the required polarity for x_i variable is '1' then replace the contents of each un-complemented part of the $\bar{x}_i$ variable in the truth vector **T** by the contents of the complemented part of the x_i variable and store the result **N** in place of complemented part of x_i variable in **T**.

The following examples will illustrate Algorithm 6.1.a and Algorithm 6.1.b.

Example 6.2

Convert a 4-variable function $f(x_3,x_2,x_1,x_0) = \Pi\ M(0,4,7,11,15)$ from POS form to polarity 7 DRM.

Store the coefficients of Maxterms in the truth vector **T**.

T = [0111 0110 1110 1110]

Separate **T** vector around variable x_3 to obtain $T(x_3)$ matrix and XNOR each element in the first row with the elements in the second row.

$$T(x_3) = \begin{bmatrix} 0111\ 0110 \\ 1110\ 1110 \end{bmatrix} \begin{matrix} x_3 \\ \bar{x}_3 \end{matrix} \quad (x_3 \text{ XNOR } \bar{x}_3)$$

$$N = [0110\ 0111]$$

Since the polarity is '0' for variable x_3, replace the un-complemented part of x_3 variable in **T** by the **N** vector results. Therefore, the new **T** vector is

T = [0110 0111 1110 1110]

Separate the new vector **T** around variable x_2 to obtain T(x_2) matrix as follows:

$$T(x_2) = \begin{bmatrix} 0110 & 1110 \\ 0111 & 1110 \end{bmatrix} \begin{matrix} x_2 \\ \bar{x}_2 \end{matrix} \quad (x_2 \text{ XNOR } \bar{x}_2)$$

$$N = [1110 \quad 1111]$$

Since the polarity is '1' for variable x_2 apply step '5', the new truth vector is

$$\mathbf{T} = [1110 \ 0110 \ 1111 \ 1110].$$

Similarly for variable x_1 and x_0

$$T(x_1) = \begin{bmatrix} 1101 & 1111 \\ 1010 & 1110 \end{bmatrix} \begin{matrix} x_1 \\ \bar{x}_1 \end{matrix} \quad (x_1 \text{ XNOR } \bar{x}_1)$$

$$N = [1000 \quad 1110]$$

$$\mathbf{T} = [1011 \ 0001 \ 1111 \ 1011]$$

$$T(x_0) = \begin{bmatrix} 1100 & 1111 \\ 0101 & 1101 \end{bmatrix} \begin{matrix} x_0 \\ \bar{x}_0 \end{matrix} \quad (x_0 \text{ XNOR } \bar{x}_0)$$

$$N = [0110 \quad 1101]$$

The final **T** vector is

$$\mathbf{T} = [0111 \ 1000 \ 1111 \ 0111] = \{0,5,6,7,12\}$$

The sum terms in this canonical can be generated by using the basis vector

$$[0\ \ x_3]++[0\ \bar{x}_2]++[0\ \bar{x}_1]++[0\ \bar{x}_0]=[0\ \ \bar{x}_o\ \ \bar{x}_1\ \ (\bar{x}_1+\bar{x}_0)\ \ \bar{x}_2\ \ (\bar{x}_2+\bar{x}_o)$$
$$(\bar{x}_2+\bar{x}_1)\ \ (\bar{x}_2+\bar{x}_1+\bar{x}_o)\ \ (x_3+\bar{x}_0)\ \ (x_3+\bar{x}_1)$$
$$(x_3+\bar{x}_1+\bar{x}_0)\ \ (x_3+\bar{x}_2)\ \ (x_3+\bar{x}_2+\bar{x}_0)$$
$$(x_3+\bar{x}_2+\bar{x}_1)\ \ (x_3+\bar{x}_2+\bar{x}_1+\bar{x}_0)]$$

The FPDRM can be generated using by substituting the coefficient vector **c** in the following general equation.

$$f(x_{n-1},x_{n-2},\ldots,x_0)=\{[0\ \ x_{n-1}]++[0\ \ x_{n-2}]++\ldots++[0\ \ x_0]\}\ ^{o}\ \mathbf{c} \tag{6.8}$$

Hence

$$f(x_3,x_2,x_1,x_0)=0\odot(\bar{x}_2+\bar{x}_0)\odot(\bar{x}_2+\bar{x}_1)\odot(\bar{x}_2+\bar{x}_1+\bar{x}_0)\odot(x_3+\bar{x}_2+\bar{x}_0)$$

Example 6.3

Convert a 4-variable function $f(x_3,x_2,x_1,x_0) = \odot\Pi\ (0,5,6,7,12)$ from FPDRM form to POS form by using polarity $\mathbf{p} = 7 = (0111)$.

Store the truth vector in **T** matrix.

$$\mathbf{T} = [0111\ \ 1000\ \ 1111\ \ 0111]$$

Separate **T** vector around variable x_3 to obtain $T(x_3)$ matrix.

$$T(x_3)=\begin{bmatrix}0111 & 1000\\ 1111 & 0111\end{bmatrix}\begin{matrix}x_3\\ \bar{x}_3\end{matrix}\ (x_3\ \text{XNOR}\ \bar{x}_3)$$
$$N=[0111\ \ 0000]$$

Since polarity is '0' for variable x_3, replace the un-complemented part of variable x_3 in **T** by the **N** vector results. The new truth vector is

$$\mathbf{T} = [0111 \;\; 0000 \;\; 1111 \;\; 0111].$$

Separate the new vector **T** around variable x_2 to obtain $T(x_2)$ matrix as follows

$$T(x_2) = \begin{bmatrix} 0111 & 1111 \\ 0000 & 0111 \end{bmatrix} \begin{matrix} x_2 \\ \bar{x}_2 \end{matrix} \quad (x_2 \text{ XNOR } \bar{x}_2)$$

$$N = [1000 \;\; 0111]$$

Since polarity is '1' for variable x_2 apply Algorithm 1.b, the new truth vector is

$$\mathbf{T} = [0000 \;\; 1000 \;\; 0111 \;\; 0111]$$

Similarly for variable x_1 and x_0

$$T(x_1) = \begin{bmatrix} 0010 & 0101 \\ 0000 & 1111 \end{bmatrix} \begin{matrix} x_1 \\ \bar{x}_1 \end{matrix} \quad (x_1 \text{ XNOR } \bar{x}_1)$$

$$N = [1101 \;\; 0101]$$

$$\mathbf{T} = [0011 \;\; 0001 \;\; 1101 \;\; 1101]$$

$$T(x_0) = \begin{bmatrix} 0100 & 1010 \\ 0101 & 1111 \end{bmatrix} \begin{matrix} x_0 \\ \bar{x}_0 \end{matrix} \quad (x_0 \text{ XNOR } \bar{x}_0)$$

$$N = [1110 \;\; 1010]$$

$$\mathbf{T} = [0111 \;\; 0110 \;\; 1110 \;\; 1110]$$

Therefore, the POS's coefficients are (0,4,7,11,15).

$$f(x_3,x_2,x_1,x_0) = \Pi \, M(0,4,7,11,15).$$

6.3 Optimization of the Fixed Polarity DRM forms

In the optimization of the FPDRM functions with different polarities are usually calculated directly from POS expressions [64, 65]. A new algorithm is presented in this chapter to find the optimal polarity directly from the truth vector of the zero polarity. This technique is aimed at large number of variables, where time is very crucial. It usually requires a long time to convert from POS to DRM for each polarity and then search for the best polarity among the 2^n polarities. The new algorithm introduced in this section will achieve maximum efficiency in respect of time for large number of variables and does not require a large memory. The time required to find a 'good' polarity, is almost equal to the time required for converting a single polarity as giving in algorithm 6.1.b. This algorithm doesn't search each polarity to convert from POS to FPDRM and it doesn't use matrix technique to convert from POS to DRM for each polarity. Thus the algorithm is fast with respect to time and efficient in terms of memory storage.

Algorithm 6.2

Step 1: Algorithm 6.1.a is used to obtain zero polarity, the coefficients are stored in vector **T**. Let $\mathbf{P}_{min}$ equals the polarity number zero for the zero polarity which is zero. Step 2: Count the number of zero terms in vector **T** and denote it by (TNZ).

Step 3: Construct T(x_i) matrix from vector **T** for each variable x_i. The first row of T(x_i) matrix contains the coefficients of the Maxterms for variable x_i in un-complemented form. The second row of T(x_i) contains the coefficients of the Maxterms with variable $\overline{x}_i$ in the complemented form.

Step 4: The elements in the first and second rows of matrix T(x_i) are grouped together using XNOR operation and the result is stored in vector **N**.

Step 5: Count the number of zeros of the un-complemented part from T(x_i) and **N** . Add the two numbers together and denote it by NZ(x_i).

Step 6: Repeat steps 4 and 5 for all the variables that have not been converted to polarity 1.

Step 7: To determine the variable (x_i) that has to be converted from '0' polarity to '1' polarity. Select the variable with the least number of zeros NZ(x_i) from step 6. This should be less than or equal to the total number of zeros from step 2, TNZ.

Step 8: Replace the contents of complemented part of T(x_i) by the contents of vector **N**. This will generate a new **T** vector. $\mathbf{P}_{min}$ is set to the new polarity number.

Step 9: Use the new **T** vector from step 8 and repeat the same procedure from step 2 for the variables that have not been converted.

Step 10: If the total number of zeros NZ(x_i) for each variable x_i from step 6 is greater than TNZ from step 2, then stop and there will be no more variables to convert from '0' polarity to '1' polarity.

Step 11: The zero elements stored in the last **T** vector are the coefficients for that particular polarity of the FPDRM.

Example 6.4

Find an optimal polarity for a 5-variable function

$$f(x_4,x_3,x_2,x_1,x_0) = \Pi\,(1,3,4,5,7,10,11,12).$$

Step 1: use Algorithm (6.1.a) to convert from POS to zero polarity DRM and set $\mathbf{P}_{min} = 0$.

The result is as follows:

$$\mathbf{T} = [1010\ \ 1100\ \ 0010\ \ 0110\ \ 1111\ \ 1111\ \ 1111\ \ 1111]$$

Hence the coefficients for PPDRM with polarity **p** = 0 are:

$$\mathbf{c} = \{1, 3, 6, 7, 8, 9, 11, 12, 15\}$$

Step 2: Count the number of zero terms in **T** vector and set TNZ = 9.

Step 3: Separate **T** vector around variable x_4 into un-complemented and complemented parts, and store the result into vector **N** as follows:

$$\mathrm{T}(x_4) = \begin{bmatrix} 1010 & 1100 & 0010 & 0110 \\ 1111 & 1111 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_4 \\ \bar{x}_4 \end{matrix} (x_4 \text{ XNOR } \bar{x}_4)$$

$$\mathrm{N} = [1010\ \ 1100\ \ 0010\ \ 0110]$$

Step 4: Count the number of zeros of the un-complemented part from T(x_4) and **N** vector and add the two numbers together.

$$\mathrm{NZ}\ (x_4) = 18$$

By repeating steps 3 to 6, the following results are obtained for the following variables x_3, x_2, x_1 and x_0

$$\mathrm{T}(x_3) = \begin{bmatrix} 1010 & 1100 & 1111 & 1111 \\ 0010 & 0110 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_3 \\ \bar{x}_3 \end{matrix} (x_3 \text{ XNOR } \bar{x}_3)$$

$$\mathrm{N} = [0111\ \ 0101\ \ 1111\ \ 1111]$$

$$\mathrm{NZ}\ (x_3) = 7$$

$$T(x_2) = \begin{bmatrix} 1010 & 0010 & 1111 & 1111 \\ 1100 & 0110 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_2 \\ \bar{x}_2 \end{matrix} (x_2 \text{ XNOR } \bar{x}_2)$$

$$N = [1001 \quad 1011 \quad 1111 \quad 1111]$$

$$NZ\ (x_2) = 8$$

$$T(x_1) = \begin{bmatrix} 1011 & 0001 & 1111 & 1111 \\ 1000 & 1010 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_1 \\ \bar{x}_1 \end{matrix} (x_1 \text{ XNOR } \bar{x}_1)$$

$$N = [1100 \quad 0100 \quad 1111 \quad 1111]$$

$$NZ\ (x_1) = 9$$

$$T(x_0) = \begin{bmatrix} 1110 & 0101 & 1111 & 1111 \\ 0010 & 0010 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_0 \\ \bar{x}_0 \end{matrix} (x_0 \text{ XNOR } \bar{x}_0)$$

$$N = [0011 \quad 1000 \quad 1111 \quad 1111]$$

$$NZ\ (x_0) = 8$$

Since NZ (x_3) has the minimum number of zeros, replace the contents of complemented part of **T** matrix by the result of XNOR operation hence the new T vector is

$$\mathbf{T} = [1010 \ 1100 \ 0111 \ 0101 \ 1111 \ 1111 \ 1111 \ 1111].$$

The total number of zeros for the new vector TNZ = 7, and the polarity number for this vector is $\mathbf{P}_{min} = \{01000\} = 8$.

Similarly as in the previous part, separate vector **T** around each variable but not x_3 because it has been changed to $\bar{x}_3$.

$$T(x_4) = \begin{bmatrix} 1010 & 1100 & 0111 & 0101 \\ 1111 & 1111 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_4 \\ \bar{x}_4 \end{matrix} (x_4 \text{ XNOR } \bar{x}_4)$$

$$N = [1010 \quad 1100 \quad 0111 \quad 0101]$$

$$NZ\ (x_4) = 14$$

$$T(x_2) = \begin{bmatrix} 1010 & 0111 & 1111 & 1111 \\ 1100 & 0101 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_2 \\ \bar{x}_2 \end{matrix} (x_2 \text{ XNOR } \bar{x}_2)$$

$$N = [1001 \quad 1101 \quad 1111 \quad 1111]$$

$$NZ\ (x_2) = 6$$

$$T(x_1) = \begin{bmatrix} 1011 & 0101 & 1111 & 1111 \\ 1000 & 1101 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_1 \\ \bar{x}_1 \end{matrix} (x_1 \text{ XNOR } \bar{x}_1)$$

$$N = [1100 \quad 0111 \quad 1111 \quad 1111]$$

$$NZ\ (x_1) = 6$$

$$T(x_0) = \begin{bmatrix} 1110 & 0100 & 1111 & 1111 \\ 0010 & 1111 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_0 \\ \bar{x}_0 \end{matrix} (x_0 \text{ XNOR } \bar{x}_0)$$

$$N = [0011 \quad 0100 \quad 1111 \quad 1111]$$

$$NZ\ (x_0) = 9$$

Since the total number of zeros for variable x_2 is less than TNZ from the last operation, therefore x_2 will be converted to $\bar{x}_2$. Hence the new **T** vector is

$$\mathbf{T} = [1010 \ 1001 \ 0111 \ 1101 \ 1111 \ 1111 \ 1111 \ 1111].$$

The total number of zeros for the new vector TNZ = 6 and $P_{min} = \{01100\} = 12$.

Repeat the same procedure for x_4, x_1 and x_0.

$$T(x_4) = \begin{bmatrix} 1010 & 1001 & 0111 & 1101 \\ 1111 & 1111 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_4 \\ \bar{x}_4 \end{matrix} (x_4 \text{ XNOR } \bar{x}_4)$$

$$N = [1010 \quad 1001 \quad 0111 \quad 1101]$$

$$NZ\ (x_4) = 12$$

$$T(x_1) = \begin{bmatrix} 1010 & 0111 & 1111 & 1111 \\ 1001 & 1101 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_1 \\ \bar{x}_1 \end{matrix} (x_1 \text{ XNOR } \bar{x}_1)$$

$$N = [1100 \quad 0101 \quad 1111 \quad 1111]$$

$$NZ\ (x_1) = 7$$

$$T(x_0) = \begin{bmatrix} 1110 & 0110 & 1111 & 1111 \\ 0001 & 1111 & 1111 & 1111 \end{bmatrix} \begin{matrix} x_0 \\ \bar{x}_0 \end{matrix} (x_0 \text{ XNOR } \bar{x}_0)$$

$$N = [0000 \quad 0110 \quad 1111 \quad 1111]$$

$$NZ\ (x_0) = 9$$

Since the total number of zeros from each operation is greater than TNZ from the last operation, there are no more variables to convert to the complemented form and the process is terminated at this point.

Therefore, the final **c** vector is given as follows:

$$\mathbf{c} = [1010 \quad 1001 \quad 0111 \quad 1101 \quad 1111 \quad 1111 \quad 1111 \quad 1111]$$

The FPDRM terms that are needed for this form are {1,3,5,6,8,14}, and the best polarity is **P** = (01100) = 12.

A program has been developed based on the previous theory and sparse technique. A matrix is called sparse if most of its elements are non-zero [75, 76]. Considerable saving in memory and computation time can be achieved by using sparse formats that store only the zeros in this case. Since most of the elements in the truth vector **T** are non-zeros where the normal size of **T** is 2^n, a sparse format will be a suitable solution to store the zero elements to avoid wasting memory. The following example illustrates this method.

Example 6.5

Let vector **A** = [0,4,7] and vector **B** = [2,4,6] then **A** XNOR **B** is given as follows:

A = [0 y y y 4 y y 7]

B = [y y 2 y 4 y 6 y]

N = [0 y 2 y y y 6 y]

Where 'y' repents a non-zero element '1' in the truth vector.

6.4 Experimental Results

In this section, experimental results are presented for the proposed algorithms. The proposed algorithms are implemented in C language and the programs are compiled using Borland C++ compiler. It is tested on a personal computer with Pentium 4 processor of 2.4 GHz CPU and 512 MB of RAM under Window operating system. The algorithms where applied to several MCNC benchmarks. Table (6.1) shows the results obtained from converting POS coefficients into PPDRM coefficients. Where name denotes the name of circuit, n denotes the number of variables, init terms denotes the number of terms in POS form, P PDRM terms denotes the number of terms in PPDRM form and the CPU time is in seconds. For most of the circuits with n less 14 the CPU time is less than 0.6 seconds. The comparison time between the results which was obtained from Chapter 4 and Chapter 6 is nearly the same. Table (6.2) shows the results obtained from converting PPDRM coefficients into POSs coefficients. Finally Table (6.3) shows the results obtained to find the optimal polarity for some Boolean functions. The CPU time for most of the circuits is nearly zero, which reflects the efficiency of the algorithms and Improv. (%) represents the average saving in terms between polarity zero and the optimal polarity which is based on the following formula:

$$\text{Improv.Terms} = \frac{\text{Initi.Terms} - \text{DRM.Terms}}{\text{Initi.Terms}} \times 100\%$$

The CPU-time for chapter 6 is much less than the CPU-time for chapter 5, also for functions with variable number n greater than 14, algorithms in chapter 5 had failed to give results. Therefore, algorithms presented in this chapter is much more efficient than algorithms for chapter 5 in terms of CPU-time and large number of variables.

Table 6.1:

Conversion table from POS to PPDRM

Name	*n*	Init. terms in POS	PPDRM Terms	Chapter 4 PPDRM Terms	Chapter 4 CPU-Time	Chapter 6 CPU-Time
Con1	7	88	9	9	0.00	0.000
Rd84	8	136	37	37	0.05	0.000
Clip	9	480	92	92	0.00	0.000
Ex1010	10	142	480	480	0.01	0.000
F12†	12	1984	365	365	0.02	0.01
F13†	13	4152	127	127	0.10	0.04
F14†	14	16172	65	127	0.711	0.521
F15†	15	5792	3100	3100	0.54	3.695
spla	16	5348	517	517	0.931	1.843
Table5	17	28552	3359	3359	9.845	162.85

Table 6.2:

Conversion table from PPDRM to POS

Name	*n*	PPDRM Terms	POS Terms	Time (s)
Clip	9	92	480	0.000
Con1	7	9	88	0.000
Ex1010	10	480	142	0.010
Rd84	8	37	136	0.000
Table3	14	2528	1859	1.29
Table5	17	3359	28552	4.927

Table 6.3:

Optimal Polarity for DRM forms

Name	*n*	PPDRM Terms polarity 0	Optimal Polarity	Improv. (%)	CPU Time Ch- 5	Time for Ch-6	DRM Terms Ch- 6	DRM-Terms Ch- 5
Con1	7	9	23	11.1	0.000	0.000	8	8
Rd84	8	37	128	21.6	0.000	0.000	29	29
Clip	9	92	496	31.5	0.031	0.000	63	63
Ex1010	10	480	2	11.2	1.061	0.000	426	413
F12†	12	143	2868	9	3.455	0.01	130	127
F13†	13	127	4800	48	4.16	0.01	65	65
F14†	14	625	5587	89	260.07	0.031	66	66
Misex3	14	913	12505	89	-	0.030	95	-
F15†	15	3100	29412	88	-	0.241	365	-
spla	16	517	3584	35.4	-	0.040	334	-
Table5	17	3359	21760	55.5	-	0.531	1494	-

†Randomly generated Boolean functions

6.5 Summary

To find the best polarity, it is required in general to search among 2^n different polarities,
for large number of variables, this would be impractical, and it would require a long CPU time to search for the best polarity. Therefore, new algorithms have developed in this chapter which could handle a large number of variables to find the optimal polarity among the 2^n different polarities without searching for each polarity. The algorithms can be used to convert from POSs to FPDRM and find the optimal polarity for large number of variables. The algorithms are based on the truth vector of the POS function. The truth vector is separated around each variable (x_i) into two parts. The first part includes the coefficients of un-complemented x_i while the second part includes the complemented part of variable x_i. Following the procedure which is described in sections 6.2 and 6.3 respectively, the following can be obtained.

I. The coefficients of the PPDRM or the FPDRM forms can be computed directly from the truth table of POS.
II. The optimal polarity with the minimum number of sum terms for any n-variable function can be obtained directly.

Thus, time efficiency and computing speed are thus achieved in this technique.

Chapter 7

Conclusions and future work

The aim of the research is to develop a variety of algorithms for the synthesis and optimization for both types of Reed-Muller and Dual Reed-Muller logical expressions.

7.1 Conclusions

The main contribution in this thesis can be outlined as follows.

- In chapter 2, a novel, fast, computational technique for converting between Boolean functions in SOP form and FPRM expressions has been presented. This method is based on partitioning the Reed-Muller transformation matrix into vertical and horizontal layers. Sparse technique is used also to store the non-zero elements instead of storing the whole matrix. The algorithm is extended to convert between multi output SOP expressions and multi output FPRM expressions. The algorithm is implemented in C language. The program is tested on personal computers and the results for some benchmark functions of up to 20 inputs and 40 outputs are presented. The experimental results reflect the efficiency of the algorithm in terms of CPU

time and using less memory space to store the transformation matrix. The conversion time for functions with n equals 15 and outputs equals to 28 was less than 4.7 seconds. The other main advantage of the algorithm is its ability to handle large number of outputs, because the algorithm is not required to do any more calculations, except incrementing a counter for each new function.

- Chapter 3 presents the Dual Reed-Muller expressions, which are based on OR/XNOR operations. Any n-variable Boolean function in the Product of Sums form can be expressed by 2^n Fixed Polarity Dual Reed-Muller forms. Chapter three covers the basic theory and notations which are used in the Dual Reed-Muller form. It also introduced new operations which can be used to describe the Dual Reed-Muller form in order to convert between Product of Sums and Dual Reed-Muller form for single and multi output functions. An algorithm is developed based partitioning and Sparse methods. Algorithms were implemented using C language. The experimental results reflect the efficiency of the algorithms in terms of CPU time and memory size.

- In chapter 4, a new fast algorithm is introduced to convert between Product of Sums and Dual Reed-Muller forms without generating or using the Dual Reed-Muller transformation matrix. This facilitates efficient conversion between Product of Sums and Fixed Polarity Dual Reed-Muller (FPDRM) forms. New algorithm is presented for bidirectional conversion between the two forms. The algorithm starts by storing the off-set of the truth vector of the Product of Sums and uses two simple equations (4.15, 4.17) to calculate each coefficient of the Dual Reed-Muller expression. The algorithm is implemented in C language. Experimental results show that the algorithm is

very efficient in terms of space and CPU time. For Boolean functions with n equals 16, the CPU time was less than one second.

- The optimization of Fixed Polarity Dual Reed-Muller (FPDRM) expansions, require the search for the best function among the 2^n fixed polarities to find the FPDRM with the least number of Sums. The classical exhaustive method for finding the best polarity requires the computation of the 2^n FPDRM expansions from a Product of Sums expression using one of the conversion methods given previously. Then exhaustive search method is applied to find the best FPDRM with the least number of sums or with the least number of XNOR gates. In order to avoid all these disadvantages, a new, efficient algorithm is presented in chapter 5 to generate all the polarity sets from any polarity set **q** for a single output Boolean function. The algorithm is based on the dual property and the Gray code strategy. Time efficiency and computing speed were achieved in this technique because the information in finding FPDRM expansion of one polarity is utilized by others. Therefore, two-fixed polarities can be derived from each other without the need to go back to the original Boolean function in the Product of Sums form, if the two polarities are dual. The program is developed and implemented in C language. Test results for benchmark and some random examples of up to 13 inputs are given.

- To reduce the search time for finding the optimal Fixed Polarity Dual Reed-Muller for large input variables, two new, efficient, algorithms are presented in chapter 6. The first algorithm is used to convert between POSs and FPDRM, while the second algorithm is used to find the optimal polarity among the 2^n different polarities for large n-variable functions directly without involving exhaustive search. The algorithms are based on Boolean matrix representation and maxterm separation techniques. The second technique used in the algorithms is sparse technique, were only the zero

coefficients are stored (saved) in order to reduce the memory size. The new algorithms are implemented in C language. The programs are tested on personal computers and the results for some benchmark functions of up to 17 variables are given. The CPU time for finding the optimal polarity for n equals 17 is less than 0.55 seconds. Time efficiency and computing speed are thus achieved in this technique.

7.2 Future Work

The above work can be further generalized and improved along the following lines.

- The conversion algorithm for single output functions, presented in chapter four, has been successfully tested for n equals to 17 variables. This algorithm can be further improved and generalised for multiple output Boolean functions based on the same strategy.

- The exact polarity optimization method in chapter five can be utilized for incompletely specified Boolean functions.

- The conversion and the optimal polarity optimization algorithm for single output functions, presented in chapter 6, can be further generalized to incompletely specified Boolean functions. It can be extended also to minimize multiple outputs fixed and mixed polarity Dual Reed-Muller forms.

- The algorithms can form part of Electronic Computer Aided Design (ECAD) package for the synthesis and optimization of large logic functions and used as a front end in commercial tools.

Publications

The following papers are published, and submitted while at Napier University.

1 Cheng, J, Chen, X, **Faraj, K.M.**, and Almaini, A.E.A, 'Expansion of logical functions in the OR-coincidence system and the transform between it and maxterm', IEE Proc.-Computers and Digital Techniques, Vol. 150, No. 6, November 2003, pp 397-402.

2 **K. Faraj**, M. MacCallum, and A. E. A. Almaini, 'Polarity Conversion Using Sparse and Partitioning Techniques,' Proceedings of the 29th EUROMICRO Conference, Turky, 2003, pp.

3 **K. Faraj**, M. MacCallum, A.E.A. Almaini, 'Fast computation of Conjunctive Canonical Reed-Muller functions,' PREP Proceeding, University of Hertfordshire, 2004, pp. 144-145.

4 **K. Faraj**, M. Al-Asadi, A.E.A. Almaini, 'A New Technique for Converting Sum of Products (SOP) into Fixed Polarity Reed-Muller (FPRM) and Vice Versa,' PREP 2004 Proceeding, University of Hertfordshire, pp. 178-179.

5 **K. Faraj**, and A.E.A. Almaini, 'Near optimal Polarity for Conjunctive Canonical Reed-Muller expansions,' submitted to International Journal of Electronics, 10/2004.

6 **K. Faraj**, Y. Xia, and A.E.A. Almaini, 'Exact minimization of the Dual Reed-Muller expansions,' submitted to International Journal of Electronics, 4/2005.

Appendix A

This section describes the input files that were used in the programmes and some information on how to run the programmes.

The input file should be given in the following form:

Table A.1: File-1

```
.i 4
.o 1
.ilb f b c d a h g
.ob f0 f1
.p
0000 1
0001 0
0010 0
0011 1
0100 0
0101 1
0110 1
0111 0
1000 0
1001 1
1010 1
1011 0
1100 1
1101 0
1110 0
1111 1
.e
```

The minimum required set of keywords is **.i** , **.o** and **.e (.end)** for binary-valued functions using PLA description.

Where

.i [d] specifies the number of input variables.

.o [d] specifies the number of output functions.

.e (.end) specifies the end of the PLA description.

Running the programmes

The following commands are necessary to run the programmes in this disk

InputFile - name of the file with the input data,

OutputFile - name of the output file to store the results of the programme.

For example: using the first programme from chapter 6 (separation1) to convert from product of sums to positive polarity Dual Reed-Muller form using zero polarity using table A.1.

The user needs to specify the input file name (File-1), polarity number (for example zero) and the output file name (c:\test.txt).

The out put of this programme is given as follows:

Number of variables = 4
Number of Dual Reed-Muller coefficients = 4
7
11
13
14

Therefore, the Dual Reed-Muller coefficients are (7, 11, 13, 14) and Dual Reed-Muller function is:

$$f(x_3, x_2, x_1, x_0) = \odot \prod(14,13,11,7)$$

$$f(x_3, x_2, x_1, x_0) = x_0 \odot x_1 \odot x_2 \odot x_3$$

Decimal-value	$x_3x_2x_1x_0$	f	Fast-Algm P = 0	Exact-Algm	Optimal-Algm
0	0000	1			
1	0001	0			
2	0010	0			
3	0011	1			
4	0100	0			
5	0101	1			
6	0110	1			
7	0111	0	7	7	7
8	1000	0			
9	1001	1			
10	1010	1			
11	1011	0	11	11	11
12	1100	1			
13	1101	0	13	13	13
14	1110	0	14	14	14
15	1111	1			

References and Bibliography

[1] Giovanni, D. M., 'Synthesis and Optimization of Digital circuits,' McGraw-Hill, Inc., NJ, 1994.

[2] Stefan Sjoholm and Lennart Lindh, 'VHDL for Designers,' Prentice Hall Europe, 1997.

[3] Andrew Rushton, 'VHDL for Logic Synthesis,' Chichester, England: John Wiley & Sons, 1998.

[4] Frederick J. Hill, Gerald R. Peterson, 'Computer Aided Logical Design with Emphasis on VLSI,' John Wiley & Sons. Inc., New York, 1993.

[5] John P. Uyemura, 'Introduction to VLSI Circuits and Systems,' New York, NY: John Wiley & Sons, 2002.

[6] Hong, S. J, Cain, R. G. and Ostapko, D. L., 'MINI: Aheuristic approach for logic minimization,' IBM J. of Res. And Dev., Vol. 18, 443-458, 1974.

[7] Quine, W. V., 'The Problem of simplifying Truth Functions,' American Mathematics Monthly, Vol.59, no.8, pp. 521-531, 1952.

[8] McCluskey, E., 'Minimization of Boolean functions,' Bell System Technical Journal, Vol.35, No.5, pp. 1417-1444. 1956.

[9] Brayton, R. K., Hatchtel, G. D, McMullen, C. T. and Sangiovanni-Vincentelli, A. L., 'Logic Minimization Algorithms for VLSI Synthesis,' Boston, Kluwer Academic Publishers, 1984.

[10] Zhegalkin, I. I., 'The technique of calculation of statements in symbolic logic,' (in Russian) Mathe. Sbornik, Vol. 34, pp. 9-23, 1927.

[11] Reed, I. S., 'Class of multiple error correcting codes and their decoding scheme,' Institute of Radio Engineers Transaction on Information Theory, PGIT-4: pp. 38- 49, 1954.

[12] Muller, D. E., 'Application of Boolean algebra to switching circuit design and to error detection,' Institute of Radio Engineers Transaction on Electronic Computers, EC-3: pp. 6-12, September 1954.

[13] Guan, Z., and Almaini, A. E. A., 'One-bit adder design based on reed-Muller expansions,' INT. J. Electronics, Vol. 79, No. 5, pp 519-529, 1995.

[14] Xu, L, and Almaini, A. E. A., 'Full-custom design of Reed-Muller universal logic modules,' INT. J. Electronics, Vol. 74, No. 4, pp 605-613, 1993.

[15] Sasao, T and Besslich, Ph, 'On the complexity of mod-2 sum PLAs,' IEEE Trans. On Comp., 39: pp. 262-266, 1990.

[16] Saul, J., 'Logic synthesis for arithmetic circuits using the Reed-Muller representation,' In European Conf. On Design Automation, pp. 109-113, 1992.

[17] McKenzie, L., Almaini, A. E. A., and Miller, J. F., 'Optimization of Reed-Muller logic functions' INT. J. Electronics, Vol. 75, No. 3, pp 451-466, 1993.

[18] Sasao, T., 'Easily Testable Realizations for Generalized Reed-Muller Expressions,' IEEE Transaction on computers, Vol. 46, No. 6, June 1997.

[19] Reddy, S. M., 'Easily Testable realization for Logic functions,' Technical Report no. 54, Univ. of Iowa, May 1972.

[20] Mitrajit, Chatterjee and Dhiraj, K. Pradhan, 'Logic Optimization with testability-New Transformations for logic Synthesis,' IEEE Transactions on Computer-Aided of integrated Circuits and systems, Vol. 17, No. 5, May 1998.

[21] Takashi, H., Kazuyuki, N., Yasuaki, N. and kensuke, S., 'Double fixed-Polarity Reed-Muller Expressions: A new Class of AND-EXOR Expressions for compact and testable Realization,' IPSJ Journal, Vol. 42, No. 4, April. 2001.

[22] Wang, L., and Almaini. A.E.A., 'Optimisation of reed-Muller PLA implementations,' IEE Proc.-Circuits Devices Syst., Vol. 149. No. 2, April 2002.

[23] Younes, A., and Miller, J. F., 'Representation of Boolean circuits as Reed-Muller expansions,' INT. J. ELECTRONICS, Vol. 91, No. 7, pp. 431-444, July 2004.

[24] Aborhey, A., ' Reed-Muller tree-based minimisation of fixed polarity Reed-Muller expansions,' IEE Proc.-Compu. Digit. Tech., Vol. 148. No. 2, pp. 63-70, March 2001.

[25] Bui, H. T., Al-Sheraidah, A. K., and Wang, Y., 'New 4-transistor XOR and XNOR designs,' Proceeding of the Second IEEE Asia Pacific ASIC design, pp. 25-28, 2000.

[26] Jeong, B. K., Sung, J.H., and Jong, K., 'New circuit for XOR and XNOR functions,' INT. J. ELECTRONICS, Vol. 82, No. 2, pp. 131-143, 1997.

[27] Wang, J.M., Fang, S. C., and Feng, W.S., 'New efficient design for XOR and XNOR functions on the transistor level,' IEEE Journal of solid-state Circuits, 29, pp. 780-786, 1994.

[28] Bui, H. T., Wang, Y., and Jiang, Y., 'Design and analysis of low-power 10-transistor full adders using novel XOR-XNOR gates,', IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 49 Issue: 1 , Jan. 2002.

[29] Walker, R. A., and Thomas. D.E., 'Behavioral Transformation for Algorithmic Level IC Design, ', IEEE Transaction on Computer-Aided Design of Integratred Circuits, 8(10), pp. 1115-1128, October 1989.

[30] http://www.hwswworld.com/downloads/f3/06A_3.PDF.

[31] Almaini, A.E.A., 'Elecrtonic logic systems,' 3 rd ed. (Prentice Hall, 1994).

[32] Green, D.H, 'Modern logic design,' (Addison-Wesley , 1986).

[33] Ercegovac, M., Lang, T., and Moreno, J. H., 'Introduction to Digital Systems,' (John Wiley & Sons, INC, 1999).

[34] Faraj, K., MacCallum, M., and Almaini, A. E. A., 'Polarity Conversion Using Sparse and Partitioning Techniques,' Proceedings of the 29th EUROMICRO Conference, Turky, 2003.

[35] Faraj, K., Al-Asadi, M., and Almaini, A. E. A., 'A New Technique for Converting Sum of Products (SOP) into Fixed Polarity Reed-Muller (FPRM) and Vice Versa,' PREP Proceeding, University of Hertfordshire, pp. 178-179, 2004.

[36] Wang, L., Almaini. A.E.A., 'Exact minimisation of large multiple output FPRM functions', IEE Proc.-Comput. Digit. Tech., Vol. 149, No. 5, pp. 203-212, September 2002.

[37] Sasao, T., and Fujita, M., 'Representations of Discrete Functions,' Kluwer Academic Publishers, 1996.

[38] Almaini, A. E. A., and McKenzie, L., 'Tabular techniques for generating kronecker expansions,' Proceddings of the Institution of Electrical Engineers, Computers and Digital Techniques, 143, pp. 205-212, 1996.

[39] Wang, L., Almaini, A. E. A., 'Fast Conversion Algorithm for Very Large Boolean Functions,' Electronics Letters, Vol.36, No.16, pp. 1370-1371, 2000.

[40] Wang, L., Almaini, A. E. A., and Bystrov, A., 'Efficient Polarity Conversion for Large Boolean Functions,' IEE Proceedings Computers and Digital Techniques, Vol.146, No.4, pp. 197-204, 1999.

[41] Xu, L., Almaini, A. E. A., Miller, J. K, and McKenzie, L., 'Reed-Muller universal logic module networks,' IEE Proceeding-E, Vol. 140, No. 2, pp.105108, March 1993.

[42] Aborthy, S., 'Reed-Muller tree-based minimisation of fixed polarity Reed-Muller expansions,' IEE Proceedings Computers and Digital Techniques, Vol.148, No. 2, pp. 63-70, March 2001.

[43] Bapiraju, V., and Bapeswara. R. V. V., 'Generation of all Reed-Muller Expansions of a Switching Function,' IEEE Transactions on Computers, Vol. 43, No. 1, January 1994.

[44] Saluja, K. K and Ong, E. H, 'Minimization of Reed-Muller canonic expansion, 'IEEE Transactions on Computers, C-28(7): pp. 535-537, July 1979.

[45] Wu, X. , Chen, X., and Hurst, S. L., 'Mapping of Reed-Muller coefficients and the minimisation of exclusive-OR switching functions,' IEE Proceedings Part E, Computers and Digital Techniques, 129(1): pp. 15-20, Jan 1982.

[46] Zhang, Y. Z., and Rayner, P.J.W., 'Minimization of Reed-Muller polynomials with fixed polarity,' IEE Proceedings Part E, Computers and Digital Techniques, 131(5): pp. 177-186, Sept 1984.

[47] McKenzie, L., and Almaini, A. E. A., 'Generating Kronecker expansions from reduced Boolean forms using tabular methods' INT. J. Electronics, Vol. 82, No. 4, pp 313-325, 1997.

[48] Sasao, Tsutomu, 'Logic Synthesis and Optimization' Kluwer Academic Publishers, 1993.

[49] Haomin, Wu., Perkowski, M. A., Xiaoqiang, Z., and Nan, Z., 'Generalized Partially-Mixed-Polarity Reed-Muller Expansion and its Fast Computation,' IEEE Transactions on Computers, Vol. 45, No. 9, September 1996.

[50] http://robotics.me.jhu.edu/~llw/courses/me530647/kron_1.pdf.

[51] Lee, C.Y., 'Representation of switching circuits by binary decision diagrams,' Bell System Technical Jour., 38: pp. 985-999, 1959.

[52] Akers, S. B., 'Binary decision diagrams,' IEEE TRANS. ON Comp., 27: pp. 509-516, 1978.

[53] Bryant, R.E., 'Graph – based algorithms for Boolean function manipulation,' IEEE Tran. On Comp., 35(8): pp. 677-691, 1986.

[54] Almaini, A. E. A., Zhuang, N., and Bourst, F., 'Minimisation of multi output Reed-Muller binary decision diagrams using hybrid genetic algorithm,' Electronic Letters, Vol. 31, No. 20, pp. 1722-1723, September 1995.

[55] Aborhey, S., 'Binary decision graph reduction,' IEE Proc. Part-E, Vol. 136, No.4, pp. 277-283, 1989.

[56] Brace, K.S., Rudell, R. L. and Bryant, R. E., 'Efficient Implemtation of a BDD package,' 27th ACM/IEEE Design Automation Conference, pp. 40-45, 1990.

[57] Ishiura, N., 'Synthesis of Multilevel Logic Circuits from Binary Decision Diagrams,' IEICE Trans. Inf. And Syst., Vol. E-76D, No. 9, pp. 1085-1092, 1993.

[58] Friedman, S. J., and Supowit, K., 'Finding the optimal variable Ordering for binary Decision Diagrams,' IEEE Trans. Comput., Vol. C-39, no. 5, pp. 710-713, 1990.

[59] Fujita, M., and Matsunaga, Y., 'Variable Ordering of Binary Decision Diagrams for Multilevel Logic Minimization,' Fujitsu Scientific and Technical journal, Vol. 29, pp. 137-145, 1993.

[60] Lind-Nielsen, J.B., 'Verification of Large State/Event Systems,' PhD Thesis April 2000.

[61] Drechsler, R., Theobald, M., and Becker, B., 'Brief Contributions Fast OFDD-Based Minimizations,' IEEE Trans. Comput., Vol. 45, No. 11, pp. 1294-1299, 1996.

[62] Lindgren, Per., 'Applications of Decision Diagrams in Digital Circuit Design,' PH. D. Thesis, Lulea University of Technology, December 1999.

[63] Wang, L, 'Automated Synthesis and optimization of Multilevel Logic Circuits,' PH. D. Thesis, Napier University, U.K., 2000.

[64] Green, D.H., 'Dual forms of Reed-Muller expansions', IEE Proc.-Comput. Digit. Tech., Vol. 141, No. 3, pp.184-192, May 1994.

[65] Cheng, J., Chen, X., Faraj, K.M., and Almaini, A.E.A., 'Expansion of logical functions in the OR-coincidence system and the transform between it and maxterm', IEE Proc.-comput. Digit. Tech., Vol. 150, No. 6, pp. 397-402, November 2003.

[66] Faraj, K., MacCallum, M., and Almaini, A.E.A., 'Fast computation of Conjunctive Canonical Reed-Muller functions,' PREP Proceeding, University of Hertfordshire, pp. 144-145, 2004.

[67] Cheng, Jie., 'The research on modern digital theory and method', PH. D. Thesis, Zhejiang University, Hangzhou, 2001.

[68] Panda R., and Najm F. 'Technology decomposition for low-power synthesis,' IEEE Custom Integrated Circuits Conference, Santa Clara, Ca, pp. 627-630, May 1995.

[69] Almaini, A. E. A., and Thomson, P., and Hanson, D., 'Tabular techniques for Reed-Muller logic,' INT. J. Electronics, 70, pp. 23-34, 1991.

[70] Tan, E. C., and Yang, H., 'Fast tabular technique for fixed –polarity Reed-Muller logic with inherent parallel processes,' INT. J. Electronics, Vol. 85, No. 4, pp. 511-520, 1998.

[71] Habib, M. K., ' Boolean matrix representation for the conversion of minterms to Reed-Muller coefficients and the minimization of exclusive-OR switching functions,' INT. J. Electronics, Vol. 68, No. 4, pp 493-506, 1990.

[72] Purwar, S., 'An efficient method of computing generalized Reed-Muller expansions from binary decision diagram,' IEEE Trans. Compu., Vol. 40, No. 11, pp. 1298-1301, 1991.

[73] Lui, P. K., and Muzio, J. C., 'Boolean matrix transforms for the minimization of modulo-2 canonical expansions,' IEEE Trans. Compu., Vol. 41, No. 3, pp. 342-347, 1992.

[74] Khan, Md. M. h. A., and Alam, Md. S., 'Mapping of fixed polarity Reed-Muller coefficients from minterms and the minimisation of fixed polarity Reed-Muller expressions,' Int. J. Electron., Vol. 83, No. 2, pp. 235-247, 1997.

[75] Toledo, S., 'Improving the memory-system performance of sparse-matrix vector multiplication,' IBM J. RES. Develop, Vol. 41, No. 6, pp. 711-725, November 1997.

[76] Duff, I. S., Heroux, M. A., and Pozo, R., 'The Sparse BLAS, Technical Report, CERFACS tr/pa/01/24,' September 2001.

[77] http://stsdas.stsci.edu/bps/linked_list.html.

[78] Almaini, A. E. A., 'A semicustom IC for generating optimum generalized Reed-Muller expansions,' Microelectronics Journal 28, pp. 129-142, 1997.

[79] Mozammel, H. K., and Shamsul, A., 'Mapping of on-set fixed polarity Reed-Muller coefficients from on-set canonical sum of products coefficients and the minimization of pseudo Reed-Muller expressions,' INT. J. Electronics, Vol. 86, No. 3, pp 255-268, 1999.

[80] Horowitz, E, Sahni, S., and Anderson-Freed. S., 'Fundamentals of Data Structures in C,' Computer Science Press, pp.135-185, 1993.

[81] Stankovic, R. S., and Falkowski, B. J., 'Spectral interpretation of the fast tabular technique for fixed-polarity reed-Muller expressions,' INT. J. Electronics, Vol. 87, No. 6, pp 641-648, 2000.

[82] Tran, A., 'Graphical method for the conversion of minterms to Reed-Muller coefficients and the minimisation of EX-OR switching functions,' Proceeding of the IEEE, Computers and Digital Techniques, 134, pp. 93-99, 1989.

[84] McKenzie, L, 'Logic synthesis and optimisation using Reed-Muller expansions,' PH. D. Thesis, Napier University, U.K., 1995.

[85] Khan, M. H. A., 'Development of algorithms for synthesis and minimization of EXOR-based logic,' PH. D. Thesis, Bangladesh University of Engineering and Technology, Bangladesh, 1998.

[86] Purwar, S., 'An efficient method of computing generalized Reed-Muller expansion from the binary decision diagram,' IEEE Trans. Comput., Vol. 40, pp. 1298-1301, 1991.

[87] Sulistyo, J. B., and Ha, D. S., 'HYPIPE: A new approach for high speed circuit design,' International ASIC/SOC Conference, Rochester, New York, pp. 203-207, September 2002.

[88] Fernandez, G. E., and Sridhar, R., 'Dual Rail static CMOS architecture for wav pipelining,' 9th International conference on VLSI Design: VLSI im mobile communication, Bangalore, India, pp. 335-336, January 1996.

[89] Sulistyo, J. B., and Ha, D. S., '5GHZ pipelined multiplier and MAC in 0.18 *u*M complementary static CMOS,' International Symposium on Circuits and Systems, Bangkok, Thailand, pp. 117-120, May 2003.

[90] Elgamel, M, Goel, S., and Bayoumi, M., ' Noise tolerant low voltage XOR-XNOR for fast arithmetic,' Proceedings of the 13th ACM Great Lakes Symposium on VLSI 2003, Washington, DC, USA, pp. 125-130, 2003.

[91] http://www.sigda.org/Archives/ProceedingArchives/Glsvlsi/Glsvlsi2003/papers/2003/glsvlsi03/pdffiles/p2_15.pdf.

[92] Sasao, T., Switching theory for logic synthesis, Kluwer Academic Publications, Boston, 1999.

[93] Stankovic, R. S., and Sassao, T., ' A discussion on the history of research in arithmetic and Reed-Muller expressions,' IEEE Transaction on computer-aided design of integrated Circuits and systems, Vol. 20, No. 9, September 2001.

[94] Moraga, C, Sassao, T., and Stankovic, R., 'A unifying approach to edge-valued and arithmetic transform decision diagrams,' Automation and remote control, Vol. 63, No. 1, pp. 125-138, 2002.

[95] Tan, E.C., and Yang, H., 'Optimization of fixed-polarity Reed-Muller circuits using dual-polarity property,' Circuits System Signal Process, Vol. 19. No. 6, pp. 535-548, 2000.

[96] Jankovic, D., Stankovic, R. and Moraga, C., 'Dual polarity in optimization of polynomial representation of switching functions,' International Workshop on Modern Functional Analysis, Operator Theory, Summability and Applications, Niska Banja, pp. 47-58, September 2003.

[97] Habib, M. K., A new approach to generate fixed-polarity Reed-Muller expansions for completely specified functions. International Journal of Electronics, 89, pp. 845-876, 2002.

Disk containing the programs

The following programs are developed in this thesis and written in C language.

1 Polarity conversion program (Chapter-2): This program reads the information from a benchmark (PLA-file), by storing the coefficients of the SOP in a vector. Then converts SOP coefficients to FPRM form. More details on this programme can be found in chapter 2.

2 Polarity conversion between POS and FPDRM (Chapter-3). This program converts between POS and FPDRM forms using sparse and partitioning techniques. More details on this programme can be found in chapter 3.

3 Polarity conversion between POS and FPDRM (Chapter-4) using fast transformation method. More details on can this programme be found in chapter 4.

4 Exact minimization of the FPDRM (Chapter-5) forms. This program finds the best polarity for FPDRM forms, which leads to FPDRM expression with the minimum number of sums. More details on this programme can be found in chapter 5.

5 Polarity conversion between POS and FPDRM (Chapter-6) forms. The program converts and finds the optimal polarity among the 2^n different polarities for large n-variable functions, without generating all of the polarity sets. More details on this programme can be found in chapter 6.