

Computer Aided Synthesis and Optimisation of Electronic Logic Circuits

By
Ban A. AL-Jassani
B.Sc., M.Sc.

A thesis submitted in partial fulfilment of the
requirements of Edinburgh Napier University,
for the award of Doctor of Philosophy

October, 2011

Abstract

In this thesis, a variety of algorithms for synthesis and optimisation of combinational and sequential logic circuits are developed. These algorithms could be part of new commercial ECAD package for future VLSI digital designs. The results show that considerable saving in components can be achieved resulting in simpler designs that are smaller, cheaper, consume less power and easier to test.

The purpose of generating different sets of coefficients related to Reed Muller (RM) is that they contain different number of terms; therefore the minimum one can be selected to design the circuits with reduced gate count. To widen the search space and achieve better synthesis tools, representations of Mixed Polarity Reed Muller (MPRM), Mixed Polarity Dual Reed Muller (MPDRM), and Pseudo Kronecker Reed Muller (PKRO_RM) expansions are investigated. Efficient and fast combinatorial techniques and algorithms are developed for the following:

- Bidirectional conversion between MPRM/ MPDRM form and Fixed Polarity Reed Muller forms (FPRM)/Fixed Polarity Dual Reed Muller forms (FPDRM) form respectively. The main advantages for these techniques are their simplicity and suitability for single and multi output Boolean functions.
- Computing the coefficients of any polarity related to PKRO_RM class starting from FPRM coefficients or Canonical Sum of Products (CSOP).
- Computing the coefficients of any polarity related to MPRM/or MPDRM directly from standard form of CSOP/Canonical Product of sums (CPOS) Boolean functions, respectively. The proposed algorithms are efficient in terms of CPU time and can be used for large functions.

For optimisation of combinational circuits, new techniques and algorithms based on algebraic techniques are developed which can be used to generate reduced RM expressions to design circuits in RM/DRM domain starting from FPRM/FPDRM, respectively. The outcome for these techniques is expansion in Reed Muller domain with minimal terms. The search space is 3^{th} Exclusive OR Sum of Product (ESOP)/or Exclusive NOR Product of Sums (ENPOS) expansions.

Genetic Algorithms (GAs) are also developed to optimise combinational circuits to find optimal MPRM/MPDRM among 3^n different polarities without the need to do exhaustive search. These algorithms are developed for completely and incompletely specified Boolean functions. The experimental results show that GA can find optimum solutions in a short time compared with long time required running exhaustive search in all the benchmarks tested.

Multi Objective Genetic Algorithm (MOGA) is developed and implemented to determine the optimal state assignment which results in less area and power dissipation for completely and incompletely specified sequential circuits. The goal is to find the best assignments which reduce the component count and switching activity simultaneously. The experimental results show that saving in components and switching activity are achieved in most of the benchmarks tested compared with recently published research. All algorithms are implemented in C++.

Declaration

I hereby 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, October 2011

Ban A. AL- Jassani

Acknowledgements

All praise is due to ALLAH who helped me with his great mercy and enabled me to complete my thesis.

I would like deeply to express my gratitude and appreciation to research supervisor Prof. A.E.A. Almaini, School of Engineering and Built Environment, Edinburgh Napier University, for his help, encouragement, constant guidance, and valuable suggestions during weekly meetings.

I am really grateful for my second supervisor, Dr. Neil Urquhart, School of Computing, Edinburgh Napier University, for his great assistance during the research.

Many thanks for all staff of Edinburgh Napier University for their help and support.

Finally, I couldn't thank enough my husband, Ali AL-Araji, for his sacrificing, supporting, understanding and his patience throughout my studying. I would like also to thank my beloved children, and my parents for their endless support and encouragement.

This research work was funded by the Iraqi Government, Ministry of Higher Education. The support is gratefully acknowledged. I also extend my very sincere gratitude to the staff of the Iraqi embassy in London, for their assistance, and support during my stay in the United Kingdom.

Contents

Abstract	ii
Declaration	iii
Acknowledgements	iv
Contents.....	v
List of Abbreviations.....	ix
List of Symbols	xi
List of Figures	xiv
List of Tables.....	xvi

1. Introduction

1.1 Combinational Circuits	3
1.1.1 Fixed Polarity Reed Muller/Fixed Polarity Dual Reed Muller ...	7
1.1.2 Mixed Polarity Reed Muller/Mixed Polarity Dual Reed Muller .	9
1.1.3 Pseudo Kronecker Reed Muller	10
1.1.4 Exclusive or Sum of Products/Exclusive nor Product of Sums .	11
1.2 Sequential Circuits	12
1.3 Literature Survey.....	15
1.3.1 Combinational Circuits	14
1.3.2 Sequential Circuits	19
1.4 Aims and Objectives	22
1.5 Structure of this thesis.....	23

2. Extended Tabular techniques for Reed Muller forms Conversions

2.1 Introduction	25
2.2 Review of Tabular Technique.....	26
2.3 Boolean Operations of ExOR/ExNOR	28
2.4 Basic Definitions and Notations	30

2.4.1	Kronecker Product Operation	30
2.4.2	Pseudo-Kronecker Product Operation	32
2.4.3	Continues Sum Operation	33
2.5	Bidirectional Conversion Techniques.....	34
2.5.1	Between FPRM & MPRM for single output functions	34
2.5.2	Between FPRM & MPRM for multi-output functions	35
2.5.3	Demonstration of the Bidirectional Conversion for MPRM	38
2.5.4	Between FPDRM & MPDRM for single output functions	40
2.5.5	Between FPDRM & MPDRM for multi-output functions	41
2.5.6	Demonstration of the Bidirectional Conversion for MPDRM .	44
2.6	Experimental Results	46
2.7	Computing the Pseudo –Kronecker RM expansion	49
2.7.1	PKRO-RM expansion from FPRM expansion	49
2.7.2	PKRO-RM expansion from standard CSOP form.....	54
2.8	Summary	56
3.Minterm/Maxterm Separation Techniques for RM forms Conversion		
3.1	Introduction	57
3.2	Boolean matrix representation	57
3.3	Minterm Separation Method	63
3.3.1	Demonstration for the Minterm Separation technique	64
3.3.2	Minterm Separation Method from CSOP	65
3.3.3	Minterm Separation Method from FPRM	71
3.4	Maxterm Separation Method	74
3.4.1	Demonstration for the Maxterm Separation technique.....	75
3.4.2	Maxterm Separation Method from CPOS	77
3.4.3	Maxterm Separation Method from FPDRM	81
3.5	Experimental Results	84

3.6	Summary	87
4. Minimisation Techniques of RM/DRM expansions		
4.1	Introduction	89
4.2	Technique for Minimisation of RM Expansions	89
4.2.1	The proposed algorithm for single output Boolean functions ..	93
4.2.2	The proposed algorithm for multi-output Boolean functions ..	95
4.2.3	Experimental Results	97
4.3	Technique for Minimisation of DRM Expansions	100
4.3.1	The proposed algorithm for single output Boolean functions	103
4.3.2	The proposed algorithm for multi-output Boolean functions	104
4.3.3	Experimental Results	107
4.3	Summary	109
5. Genetic Algorithms for Optimisation of Combinational Circuits		
5.1	Introduction	110
5.2	Genetic Algorithm	111
5.3	Definitions.....	112
5.4	GA for single output Boolean functions	115
5.4.1	Population Initialisation	115
5.4.2	Evaluation and fitness score.....	117
5.4.3	Selection	117
5.4.4	Reproduction (Crossover and Mutation)	117
5.4.5	Replacement	118
5.4.6	Halting Criterion	119
5.4.7	Experimental Results for single output functions	119
5.5	GA for multi-output Boolean functions.....	125
5.5.1	GA using Extended_Tabular techniques	125
5.5.2	GA using Minterm/Maxterm Separation techniques.....	128

5.6	GA for Incompletely Specified Functions	133
5.6.1	Single stage GA for Incompletely specified functions	134
5.6.2	Scheme for Single stage GA	136
5.6.3	Multi stage GA for Incompletely specified sunctions	138
5.6.4	Experimental Results for single and Multi stage GA	144
5.7	Summary	147
6. Optimal State Assignment using Multi-Objective Genetic Algorithm		
6.1	Introduction.....	149
6.2	State Assignment for Sequential Circuits	149
6.3	Multi-Objective Genetic Algorithm.....	160
6.4	Proposed Algorithm	162
6.5	Experimental Results	171
6.6	Summary	176
7. Conclusions and Future Work		
7.1	Review of Algorithms and Techniques.....	177
7.2	Future Works.....	183
Publications		185
References		186
Disk Containing the Programs.....		199
Appendix An Example of Running of Each Program.....		203

List of Abbreviations

SR flip flop	Set_Reset flip flop
D flip flop	Delay flip flop
T flip flop	Toggle flip flop
ExOR	Exclusive-OR
ExNOR	Exclusive-NOR
IC's	Integrated Circuits
SSI	Small Scale Integration
VLSI	Very Large Scale Integration
CPU	Central Processing Unit
CSOP	Canonical Sum of Products
CPOS	Canonical Product of Sums
RM	Reed Muller
DRM	Dual Reed Muller
FPGA	Field Programmable Gate Array
LUTs	Look-Up tables
ESOP	Excusive –or-Sum-of-Products
PPRM	Positive Polarity Reed Muller
PPDRM	Positive Polarity Dual Reed Muller
FPRM	Fixed Polarity Reed Muller
FPDRM	Fixed Polarity Dual Reed Muller

List of Abbreviations

MPRM	Mixed Polarity Reed Muller
KRO	Kronecker expansion
MPDRM	Mixed Polarity Dual Reed Muller
PKRO-RM	Pseudo Kronecker Reed Muller
ENPOS	Exclusive-nor-Product-of-Sums
FSM	Finite State Machine
STG	State Transition Graph
S_TT	Serial Tabular technique
P_TT	Parallel Tabular technique
STT	State Transition Table
RKROs	Reduced Kronecker Expressions
GA	Genetic Algorithm
MOGA	Multi Objective Genetic Algorithm
AV.	Average
STD	Standard Deviation
DC	Don't Care state
BDD	Binary Decision Diagram
EHW	Evolvable Hardware
ISFSMs	Incompletely Specified Finite State Machines
HD	Hamming Distance
ECAD	Electronic Computer Aided Design

List of Symbols

n	Number of variables for combinational circuits
k	Number of states for sequential circuits
i	In Chapters (2-5), index for minterms $0 < i < 2^n - 1$ In Chapter 6, index for number of states k , $0 < i < k - 1$
j	In Chapters (2-5), index for number of variables, $0 < j < n - 1$ In Chapter 6, index for number of states k , $0 < j < k - 1$
a_i	CSOP coefficients
d_i	CPOS coefficients
m_i	CSOP minterms
M_i	CPOS maxterms
dc_i	don't-care coefficients
b_i	RM coefficients
π_i	RM Product terms
c_i	DRM coefficients
Q_i	DRM Sum terms
t	Number of terms
C_M	Coefficient matrix
B	Number of blocks within the coefficient matrix
S_α	Size of each block within the coefficient matrix
α	Symbol for each block within the coefficient matrix
$R(x_j)$	Partitioned matrix
μ	Number of don't care terms
T	Tournament size
o	Number of outputs
h	any arbitrary state

S	Finite Set of states
X	Finite input alphabet
Z	Finite output alphabet
β	State transition function
δ	Output transition function
q_0	Starting state
I	Inputs of sequential circuits
A	Total number of different possible state assignment of FSM
L	Total number of unique state assignment of FSM
$*$	Kronecker Product
\diamond	Pseudo Kronecker product
\star	Continuous Sum Operation
\oplus	ExOR operation
\odot	ExNOR Operation
$+$	OR Operation
$\langle p_j \rangle$	Polarity digits for RM expressions
\times	Multiplication
r	Total number of objectives
w	Weight
V_{dd}	Supply Voltage
C_L	Physical Capacitance
f_{clk}	Clock frequency
E_{sw}	Expected switching activity
$tp_{(i \leftrightarrow j)}$	Total state transition
P_i	Steady State Probability

List of Symbols

p_{ij}	Conditional state transition
$\sum_n N_{in}$	All transitions begin with state s_i
PT	Product terms
C	Cost as function of switching activity

List of Figures

1.1	Sequential logic circuits	2
1.2	Classes of Reed Muller Expansions	8
1.3	STG for Example 1.5	14
3.1	Pseudo code for Minterm separation technique	67
3.2	Pseudo code for Maxterm separation technique	78
4.1	Pseudo code for RM_Minimisation technique	94
4.2	Chart for saving in terms for all tested Benchmarks shown in Table 4.8	100
4.3	Chart for saving in terms for all tested Benchmarks shown in Table 4.16	108
5.1	Basic flowchart of GA	113
5.2	Chromosome representation Of GA	114
5.3	Pseudo code for the proposed GA	116
5.4	Chromosome representation for 6-variable Boolean function	116
5.5	Single point crossover used in the proposed GA	118
5.6	Uniform mutation used in the proposed GA	118
5.7	Average of GA results with respect to population size for the Benchmarks T481 & Ryy6	121
5.8	Average of GA results with respect to Tournament size for the Benchmarks T481 & Ryy6	122
5.9	Efficiency of GA , calculates as eq. (5.1)	123
5.10	Average of GA results with respect to evaluations for the Benchmarks T481& Ryy6	124
5.11	Time Comparison between GAs	130
5.12	Details of individuals for Example 5.1	135
5.13	Pseudo code for multi stage GA	140

List of Figures

6.1	State Transition Graph for Lion Benchmark	153
6.2	Conditional state Probabilities for Lion Benchmark	155
6.3	Pseudo code for the proposed MOGA	164
6.4	Uniform Crossover used in the proposed algorithm	165
6.5	Mutation used in the proposed algorithm	166
6.6	Terms and Switching Activity for different assignment of Example 6.3	168
6.7	Different ranks for Example 6.3	170
6.8	Number of MOGA Evaluations for different Benchmarks	176

List of Tables

1.1	FSM_STT representation of Example 1.5	15
2.1	Terms generated by variable x_2 for Example 2.1	26
2.2	Terms generated by variable x_1 for Example 2.1	27
2.3	Terms generated by variable x_2 for Example 2.2	28
2.4	Terms generated by variable x_1 for Example 2.2	28
2.5	Boolean Operations	29
2.6	FPRM terms for Example 2.3	36
2.7	Terms generated by variable x_1 for Example 2.3	36
2.8	Terms of Polarity 6 for Example 2.3	37
2.9	Terms generated by variable x_0 for Example 2.3	37
2.10	Terms of Polarity 7 for Example 2.3	37
2.11	FPDRM terms for Example 2.3	41
2.12	Terms generated by variable x_1 for Example 2.4	42
2.13	Terms of polarity 6 for Example 2.4	42
2.14	Terms generated by variable x_0 for Example 2.4	43
2.15	Terms of polarity 7 for Example 2.4	43
2.16	Benchmark results for MPRM	47
2.17	Benchmark results for MPDRM	48
2.18	Polarity changing of variable x_2 for Example 2.5	51
2.19	Polarity changing of variable x_1 for Example 2.5	51
2.20	Polarity changing of variable x_0 within true part of x_2 for Example 2.5.	52
2.21	Polarity changing of variable x_0 within complement part of x_2 and true part of x_1 for Example 2.5.	52
2.22	Polarity changing of variable x_0 within complement part of x_2 and x_1 for Example 2.5.	53

List of Tables

2.23	Resulted PKRO_RM terms – polarity 318 for Example 2.5.	53
2.24	Changing polarity of x_1 from mixed to true for Example 2.6.	54
2.25	Changing polarity of x_1 from true to complement for Example 2.6.	55
2.26	2.26 Changing polarity of x_0 from mixed to true in the true part of x_1 for Example 2.6.	55
2.27	PKRO_RM terms – polarity 11 for Example 2.6.	55
3.1	FPRM terms for Example 3.5.	72
3.2	FPDRM terms for Example 3.7.	82
3.3	Benchmark results for Minterm separation technique.	86
3.4	Benchmark results for Maxterm separation technique.	87
4.1	Truth table for Example 4.1.	91
4.2	Terms generated by variable x_0 for Example 4.1.	92
4.3	Terms generated by variable x_2 for Example 4.1.	92
4.4	Resulted terms for Example 4.1 .	92
4.5	PPRM terms for Example 4.2.	95
4.6	Terms generated by variable x_1 for Example 4.2.	96
4.7	Resulted terms for Example 4.2.	96
4.8	Benchmark results for RM_Minimisation technique.	98
4.9	Truth table for Example 4.3.	101
4.10	Terms generated by variable x_1 for Example 4.3.	102
4.11	Terms generated by variable x_2 for Example 4.3.	102
4.12	Resulted terms for Example 4.3.	103
4.13	PPDRM terms for Example 4.4.	105
4.14	Terms generated by variable x_1 for Example 4.4.	105
4.15	Resulted terms for Example 4.4.	106
4.16	Benchmark results for RM_Minimisation technique.	108
5.1	Total Number of MPRM/or MPDRM expressions.	110

List of Tables

5.2	Benchmark results of GA for single output Boolean functions.	120
5.3	Experimental results for the pdc Benchmark	125
5.4	Benchmark results of GA for multi-output Boolean functions	127
5.5	Benchmark results of GAs using Minterm/Maxterm separation Techniques	129
5.6	Time Comparison for large Benchmarks	131
5.7	Comparison between GAs and RM_Minimisation techniques	132
5.8	Comparison between the proposed GA and Reference [48]	133
5.9	Truth table for Example 5.2	141
5.10	Initialisation of the polarity population for Example 5.2	141
5.11	Initialisation of the “don’t care” population for Example 5.2	142
5.12	Truth vector for outputs of Example 5.2	143
5.13	Best individuals produced from the first stage of GA for Example 5.2	144
5.14	Benchmark results for multi stage GA	145
5.15	Comparison between single and multi stage GA	147
6.1	Total and unique number of possible state assignment	150
6.2	FSM_STT representation of Lion Benchmark	154
6.3	Random assignments for Example 6.1	156
6.4	Kiss2 format of Lion Benchmark	157
6.5	One possible state assignment for Example 6.2 .	163
6.6	FSM-STT representation of bbtas Benchmark .	167
6.7	Codes and ranks for different state assignments for Example 6.3.	169
6.8	Experimental results for the proposed MOGA	172

Chapter One

Introduction

Computer aided synthesis and optimisation techniques for electronic logic circuits is the use of computer software for the process of converting the high level description of digital circuits into an optimised design implementation in terms of logic gates.

Digital electronic circuits are used usually in digital television, mobile phones, digital computers and many other applications. These circuits are made of small electronic circuits known as logic gates such as AND, OR, and NOT which are implemented electronically using diodes or transistors and represented by different shape. Each logic gate performs logical operation, based on Boolean algebra. Many other complex circuits can be created by connected these gates in different ways such as NAND, NOR, Exclusive-OR (ExOR), Exclusive-NOR (ExNOR), adders, registers, multiplexer and microprocessor. The main logical operations are AND, OR, and NOT, while all other operations which are performed by NAND, NOR, ExOR and ExNOR can be derived from the main operations [1].

A logic gate has one or more inputs and produces a single output. Logic level can be represented by a voltage level. Each logic gate consumes power to be able to achieve the correct output voltage. Since the 1960s, gates have become available as standard integrated circuits (ICs) known as chips. Integrated circuits may contain only a few gates, which are called "Small-Scale Integration" (SSI), or up to hundreds of thousands, which are called "Very Large Scale Integration" (VLSI). Computer aided tools

provide an effective mean for designing VLSI digital circuits. Synthesis techniques which specify the gate level structure of the circuits are required to speed up the design cycle and to reduce the human efforts, while the optimisation techniques are crucial to enhance the quality of the design [2, 3].

Logic circuits can be classified into the following two main categories:

- a. Combinational Logic Circuits
- b. Sequential Logic Circuits

Combinational circuits use logic gates such as AND, OR, and NOT gates in which the outputs at any given time depends on the present input only at that time. In sequential circuits, the output at any given time is a function of both present and past inputs. Therefore, additional logic is necessary, which must be capable of storing the past inputs. Sequential circuits can be represented by a combinational circuit in conjunction with some form of memory elements such as flip flops as in Figure 1.1. There are many types of flip flops, which can be used such as SR, JK, T and D flip flop [1].

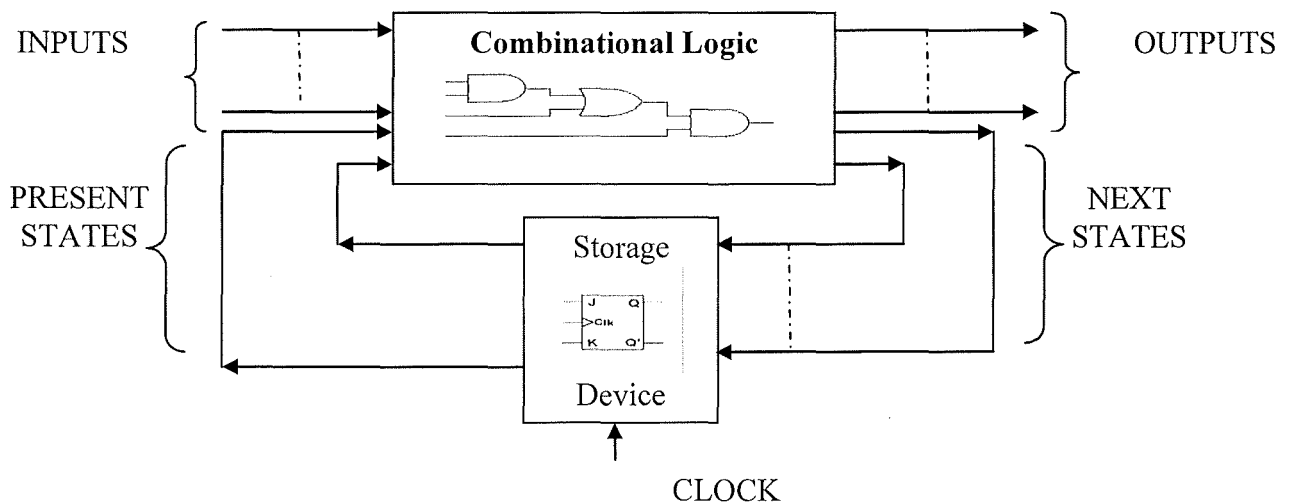


Figure 1.1: Sequential logic circuits

1.1 Combinational Circuits

Logic synthesis is the process of converting a digital circuit from the functional description into gate level representation. Optimisation means finding an equivalent representation for the specified digital circuit with less hardware. In fact, synthesis without optimisation may result in a non competitive circuit at all [2, 3]. Circuit area is one of the most important quantities and is measured by the sum of the areas of the components required to design the circuit. In this research, for combinational circuits, hardware required to design the logic circuits has been considered to reflect the quality for the optimisation process.

Combinational circuits can be represented using two different canonical standard forms [2]. These forms are known as Canonical Sum of Product (CSOP) and Canonical Product of Sum (CPOS).

The CSOP expansions are based on AND/OR for two level logic networks as shown in equation (1.1)

$$F(x_{n-1}, x_{n-2}, x_0) = \sum_{i=0}^{2^n-1} a_i m_i \quad (1.1)$$

Where m_i are the minterms; $a_i \in \{0,1\}$, and it indicates the absence or presence of minterms, respectively. $0 \leq i \leq 2^n - 1$.

$$m_i = \dot{x}_{n-1} \dot{x}_{n-2} \dots \dot{x}_0 \quad (1.2)$$

$$\dot{x}_j = \begin{cases} \bar{x}_j, & i_j=0 \\ x_j, & i_j=1 \end{cases} \quad (1.3)$$

Where; $0 \leq j \leq n - 1$.

The CPOS expansions are based on OR/AND as shown in equation (1.4).

$$F(x_{n-1}, x_{n-2}, x_0) = \prod_{i=0}^{2^n-1} d_i + M_i \quad (1.4)$$

Where M_i are the maxterms; $d_i \in \{0,1\}$, unlike a_i , it indicates the presence or absence of maxterms, respectively.

$$M_i = \dot{x}_{n-1} + \dot{x}_{n-2} + \dots \dot{x}_0 \quad (1.5)$$

$$\dot{x}_j = \begin{cases} x_j, & i_j=0 \\ \bar{x}_j, & i_j=1 \end{cases} \quad (1.6)$$

When the circuit doesn't use all the possible input combinations, it is known as an incompletely specified Boolean function and the unused combinations are called "don't-care" terms. An incompletely specified Boolean function is a function with one or more minterms/or maxterms with undefined values. These unspecified minterms/or maxterms are known as "don't care" terms/or sums respectively, and sometimes can help the process of minimisation.

Any n _variable incompletely specified Boolean function may be represented in CSOP form as:

$$F(x_{n-1}, x_{n-2}, x_0) = \sum_{i=0}^{2^n-1} a_i m_i + \sum_{i=0}^{2^n-1} dc_i m_i \quad (1.7)$$

Where $dc_i \in \{0,1\}$ are "don't care" coefficients which may be used as 0 or 1, and $dc_i = 1$ indicates the presence of don't care terms.

Boolean functions can be represented using either tabular form which is also known as a truth table, logic expressions or binary decision diagrams. The designing of the combinational circuit starts with deriving the truth table from a statement which specifies the logical behaviour of the circuit. Boolean equations are derived, simplified if possible, and implemented using suitable devices. The simplification is carried out to minimise the hardware components. Boolean expressions can be minimised by either

reducing the number of terms in the expressions and/or reducing the number of literals in the expressions [2].

Boolean algebra is commonly used to simplify the logic equations for small problems. In general, Karnaugh maps are used to simplify the functions with up to six variables. For functions with more than six variables, tabular techniques, Quine-McCluskey or Espresso [1, 4] are used to simplify the circuits.

Furthermore; combinational circuits can be described in terms of Reed Muller (RM) expansions [5]. RM expansions make use of modulo-2 logic operations where Galois fields over binary numbers $GF(2)$ are employed. Modulo-2 addition and multiplication are identical to ExOR & AND logic operations respectively. RM expression is another way to represent the standard CSOP Boolean function using AND/ExOR instead of AND/OR respectively. While Dual Reed Muller (DRM) expression is another way to represent the standard CPOS Boolean function using OR/ExNOR instead of OR/AND respectively [2, 4].

RM expressions have several advantages for functions that don't produce efficient solutions using CSOP/CPOS techniques. These advantages [4-9] are as follows:

- Logic functions which can't minimise well in neither CSOP nor CPOS form can often be implemented in either RM or DRM domain with fewer gates and interconnections which leads to reduced size.
- Circuits with ExOR / ExNOR gates are more amenable to efficient testing strategies.
- ExOR/ExNOR are classified as low speed and large area devices. But as the Field Programmable Gate Array (FPGA) made good progress in recent year, ExOR/ExNOR can easily be mapped to Look-up tables (LUTs), resulting in ExOR/ExNOR gates that are as fast as other gates.

- ExOR/ExNOR is linear and many techniques such as matrix and transform operation can be used with modulo-2 field.
- The Boolean function based on CSOP can have up to 3^{th} Exclusive-Or-Sum-Of-Products (ESOP) expressions for n variables and t terms. Due to this large number of different expressions for the Boolean circuits, there is better chance to find economical circuits in RM domain.

If all variables are present in every minterm in (1.1), then the OR gate can be replaced by ExOR gate results in ExOR-SOP expression as follows:

$$F(x_{n-1}, x_{n-2}, x_0) = \oplus \sum_{i=0}^{2^n-1} a_i m_i \quad (1.8)$$

Where \oplus denotes the ExOR operator. It can also be expressed as in (1.9) which is known as the RM form as follows:

$$F(x_{n-1}, x_{n-2}, x_0) = \oplus \sum_{i=0}^{2^n-1} b_i \pi_i \quad (1.9)$$

Where π_i are the product terms of the RM function, $b_i \in \{0,1\}$ and it indicates the absence or presence of coefficients, respectively.

Further, if all variables exist in every maxterm in (1.4), then the AND gate can be replaced by ExNOR gate which results in ExNOR-POS expressions as follows:

$$F(x_{n-1}, x_{n-2}, x_0) = \odot \prod_{i=0}^{2^n-1} d_i + M_i \quad (1.10)$$

Where \odot denotes the ExNOR operator. It can also be expressed as in (1.11) which is known as the DRM form as follows:

$$F(x_{n-1}, x_{n-2}, x_0) = \odot \prod_{i=0}^{2^n-1} c_i + Q_i \quad (1.11)$$

Where Q_i are the sum terms of the DRM, $c_i \in \{0,1\}$, unlike b_i , it indicates the presence or absence of coefficients, respectively.

Any arbitrary functions $f(x_{n-1}, x_{n-2}, x_0)$ can be expanded [10] with respect to any variable x_j using the Shannon theorem as follows:

$$f = \bar{x}_j f(x_{n-1}, \dots, x_{j+1}, 0, x_{j-1}, \dots, x_0) \oplus x_j f(x_{n-1}, \dots, x_{j+1}, 1, x_{j-1}, \dots, x_0)$$

Shannon expansion (1.12)

Substituting $\bar{x}_j = x_j \oplus 1$ in equation (1.12), gives:

$$f = f(x_{n-1}, \dots, x_{j+1}, 0, x_{j-1}, \dots, x_0) \oplus x_j [f(x_{n-1}, \dots, x_{j+1}, 0, x_{j-1}, \dots, x_0) \oplus f(x_{n-1}, \dots, x_{j+1}, 1, x_{j-1}, \dots, x_0)]$$

$$f = f_j^0 \oplus x_j [f_j^0 \oplus f_j^1] \quad \text{Positive Davio expansion (1.13)}$$

Substituting $x_j = \bar{x}_j \oplus 1$ in equation (1.12), gives:

$$f = f(x_{n-1}, \dots, x_{j+1}, 1, x_{j-1}, \dots, x_0) \oplus \bar{x}_j [f(x_{n-1}, \dots, x_{j+1}, 0, x_{j-1}, \dots, x_0) \oplus f(x_{n-1}, \dots, x_{j+1}, 1, x_{j-1}, \dots, x_0)]$$

$$f = f_j^1 \oplus \bar{x}_j [f_j^0 \oplus f_j^1] \quad \text{Negative Davio expansion (1.14)}$$

Seven classes of AND/ExOR expressions have been developed by many researchers [10, 11]. This research focuses on four classes for the RM/DRM expansions shown in Figure 1.2.

1.1.1 Fixed Polarity Reed Muller / Fixed Polarity Dual Reed Muller

Equations (1.9) & (1.11) are known as Positive Polarity Reed Muller (PPRM)/ Positive Polarity Dual Reed Muller (PPDRM) respectively, in case that all variables appear in their true form as in equation (1.13). If each

variable in equations (1.9) or (1.11) can appear in its true form as in equation (1.13) or complemented form as in equation (1.14) but not both, the expression is known as Fixed Polarity Reed Muller (FPRM) or Fixed Polarity Dual Reed Muller (FPDRM), respectively. Both of these types have 2^n different polarities or expansions. Each expression can be identified by a polarity number. The polarity of any FPRM/FPDRM expansion can be represented by replacing each variable by 0, or 1 depending on whether the variable is used in true or complement form, respectively. The polarity will be the decimal equivalent of the resulting binary number. A number of techniques have been developed to obtain any fixed polarity from the CSOP/CPOS [12 - 20].

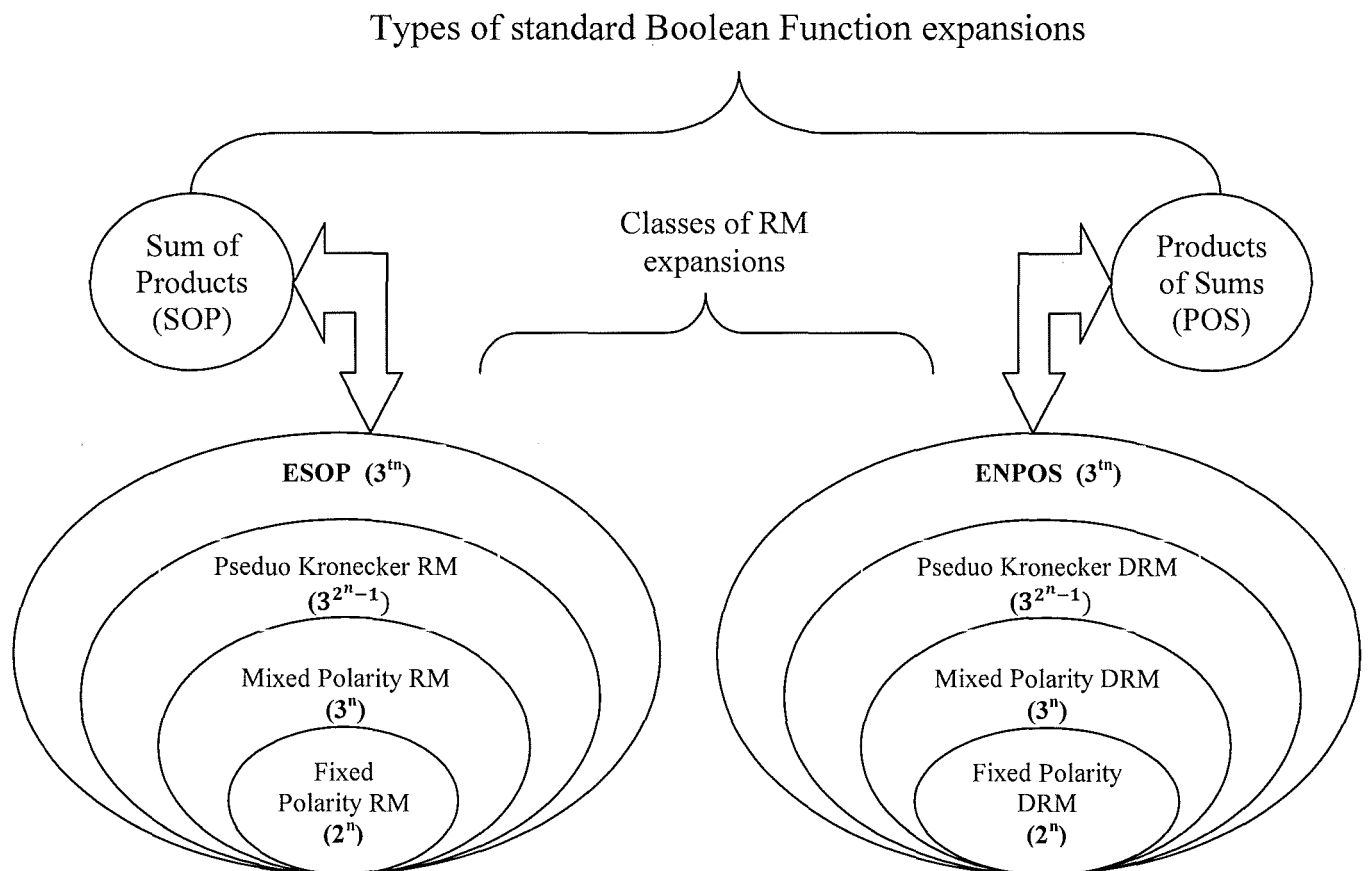


Figure 1.2: Classes of Reed Muller Expansions

In FPRM, π_i in (1.9) can be expressed as below:

$$\pi_i = \ddot{x}_{n-1} \ddot{x}_{n-2} \dots \ddot{x}_0 \quad (1.15)$$

$$\ddot{x}_j = \begin{cases} 1 & , i_j = 0 \\ \dot{x}_j & , i_j = 1 \end{cases} \quad (1.16)$$

$$\dot{x}_j = \begin{cases} \bar{x} & , \text{if polarity of } x_j = 1 \\ x & , \text{if polarity of } x_j = 0 \end{cases} \quad (1.17)$$

Further, In FPDRM , Q_i in (1.11) can be expressed as below :

$$Q_i = \ddot{x}_{n-1} + \ddot{x}_{n-2} + \dots \ddot{x}_0 \quad (1.18)$$

$$\ddot{x}_j = \begin{cases} 0 & , i_j = 1 \\ \dot{x}_j & , i_j = 0 \end{cases} \quad (1.19)$$

$$\dot{x}_j = \begin{cases} \bar{x} & , \text{if polarity of } x_j = 1 \\ x & , \text{if polarity of } x_j = 0 \end{cases} \quad (1.20)$$

Example 1.1: $f(x, y, z) = x\bar{y} \oplus \bar{y}z \oplus xz$

x & z appear in its true form, while y appears in its complement form.

Therefore this expression has fixed polarity $(010)_2 = (2)_{10}$

1.1.2 Mixed Polarity Reed Muller / Mixed Polarity Dual Reed Muller

Mixed Polarity Reed Muller (MPRM) is also called Kronecker expression (KRO). If each variable in equations (1.9), or (1.11) can appear in true, complement or both (mixed) form as in equation (1.12), then the expression is known as MPRM or Mixed Polarity Dual Reed Muller (MPDRM) respectively. Both of these two types have 3^n different polarities or expansions. Each expression may be identified by polarity number. The polarity of any mixed RM/DRM expansion can be represented by replacing each variable by 0, 1, or 2 depending on whether the variable is used in true, complement or mixed form, respectively. The polarity will be the decimal equivalent of the resulting ternary number. New techniques have

been presented to obtain any mixed polarity from the CSOP/CPOS [12, 21-26].

In MPRM, π_i in (1.9) can be expressed as in (1.21):

$$\pi_i = \ddot{x}_{n-1}\ddot{x}_{n-2} \dots \ddot{x}_0 \quad (1.21)$$

$$\ddot{x}_j = \begin{cases} \bar{x}, & \text{if polarity of } x_j = 1 \ \& \ i_j = 1 \\ x, & \text{if polarity of } x_j = 0 \ \& \ i_j = 1 \\ \check{x}, & \text{if polarity of } x_j = 2 \\ 1, & \text{if polarity of } x_j = 0 \ \text{or } 1 \ \& \ i_j = 0 \end{cases} \quad (1.22)$$

$$\check{x}_i = \begin{cases} x & \text{if } i_j = 1 \\ \bar{x} & \text{if } i_j = 0 \end{cases} \quad (1.23)$$

In MPDRM, Q_i in (1.11) can be expressed as below:

$$Q_i = \ddot{x}_{n-1} + \ddot{x}_{n-2} + \dots \ddot{x}_0 \quad (1.24)$$

$$\ddot{x}_j = \begin{cases} \bar{x}, & \text{if polarity of } x_j = 1 \ \& \ i_j = 0 \\ x, & \text{if polarity of } x_j = 0 \ \& \ i_j = 0 \\ \check{x}, & \text{if polarity of } x_j = 2 \\ 0, & \text{if polarity of } x_j = 0 \ \text{or } 1 \ \& \ i_j = 1 \end{cases} \quad (1.25)$$

$$\check{x}_i = \begin{cases} x & \text{if } i_j = 0 \\ \bar{x} & \text{if } i_j = 1 \end{cases} \quad (1.26)$$

Example 1.2: $f(x, y, z) = x\bar{y} \oplus \bar{y}z \oplus x\bar{y}\bar{z}$

x appears in true form, *y* appears in complement form, and *z* appears in mixed form. Therefore this expression has mixed polarity $(012)_3 = (21)_{10}$.

1.1.3 Pseudo Kronecker Reed Muller

Each function or sub function can be represented by one of the three types represented by eq's (1.13), (1.14) or (1.12) resulting in expressions known as Pseudo Kronecker Reed Muller (PKRO-RM) expressions. In PKRO-RM, same variables can appear in true and complement form within the expression. There are 3^{2^n-1} different pseudo Kronecker polarities within each of these classes including the 3^n KRO expansions.

In general, to allow the numbering of all the 3^{2^n-1} polarities for PKRO-RM forms, $(2^n - 1)$ digits are needed to represent the polarity number of each expression in PKRO class of RM.

The polarity of any PKRO-RM expansion can be represented using ternary numbers. The polarity will be the decimal equivalent of the resulting ternary number.

*Example 1.3: $f(x, y, z) = \bar{z} \oplus y\bar{z} \oplus \bar{x}y \oplus \bar{x}y\bar{z}$
This expression represents PKRO-RM expansion.*

1.1.4 Exclusive or Sum of Products / Exclusive nor Products of Sums

Different combinations of product terms combined by ExORs are called ESOP, while different combinations of sum terms combined by ExNORs are called Exclusive-Nor-Product-of-Sum (ENPOS). The ESOP and ENPOS are the most general classes. There are 3^m different expressions within each of these classes [11]. Each term within the expression can have its own polarity.

In ESOP, π_i in (1.9) can be expressed as below:

$$\pi_i = \ddot{x}_{n-1}\ddot{x}_{n-2} \dots \ddot{x}_0 \quad (1.27)$$

Where \ddot{x}_i in each products can be 1, x_i or \bar{x}_i independently of other terms

In ENPOS, Q_i in (1.11) can be expressed as in (1.28).

$$Q_i = \ddot{x}_{n-1} + \ddot{x}_{n-2} + \dots \ddot{x}_0 \quad (1.28)$$

Where \ddot{x}_i in each sums can be 0, x_i or \bar{x}_i independently of other sums.

Example 1.4: $f(x, y, z) = x \oplus y \oplus xyz \oplus \bar{x}\bar{y}$

ESOP expression where each term has its own polarity.

1.2 Sequential Circuits

Optimisation of sequential circuits corresponds to searching for the best design considering all objectives like power, area, and speed. In this research, for sequential circuits, area and power have been considered to express the quality for the optimisation process. Sequential circuits can be classified into two categories; synchronous (clocked) and asynchronous (unclocked). This research is concerned with synchronous sequential circuits where the transition between states is controlled by a clock pulse [27].

A Finite State Machine (FSM) is a mathematical model of the sequential circuit with discrete inputs, discrete outputs, and internal states.

State transition graph (STG) is another way to visualize FSM as directed graph where each node represents one state and the transitions between states represented by edges. Each edge in STG is labelled by the input causing this transition and the output asserted on the transition. Further, State Transition Table (STT) is a tabular form commonly used to represent FSM in which each row corresponds to an edge in STG.

An incompletely specified sequential circuit is one in which at least one state transition edge from some state is not specified. These states are called don't-care (DC) conditions [28] and represented using “-“ in the STT.

The designing of sequential circuits starts with deriving the STG and the STT from the word description of the desired behaviour of the circuit. The resulting state table may contain more states than necessary. Therefore; it is desirable to reduce the number of internal states in order to reduce the complexity of the sequential circuit. The state minimisation starts by the means of generating the implication table [29, 30] depending on the equivalencies and compatibilities between each pair of states. This

procedure is essential for completely and incompletely specified sequential circuits. State table reduction issue can be defined by finding state table with less number of states without changing the functionality.

Then synthesis tools are required to encode the internal symbolic states of FSM as binary code. The state assignment refers to the allocations of the binary codes to the states of the sequential circuits. The resulting logic of the sequential circuit depends on the codes assigned to the states. Different coding may lead to a different number of components giving the same functionality of the designed circuit. Therefore; it is vital to construct the circuit with less hardware through finding the optimal assignment of the designed circuit.

An FSM, M , is characterised by a six tuple vector $(S, X, Z, \beta, \delta, q_0)$, where S is a finite set of states, X is a finite input alphabet, β is the state transition function mapping $S \times X \rightarrow S^+$, Z is the output alphabet, δ is the output transition function and q_0 is the initial or starting state. There are two different types of FSM, depending on output transition function δ which are called the Moore and Mealy models. In the Moore model, the outputs depend on the states only while in the Mealy model, the outputs depend on the inputs as well as on the states. In the Moore FSM, output transition function δ is a mapping $S \rightarrow Z$, whereas in the Mealy FSM, δ is a mapping $S \times X \rightarrow Z$.

A FSM having k distinct states and x inputs, requires $s = \lceil \log_2 k \rceil$ state variables and $I = \lceil \log_2 x \rceil$ input variables for the complete assignments, where $\lceil g \rceil$ is defined as the smallest integer equal to or greater than g . The total number of the different possible encodings [28] is given by $L(k)$ as defined by equation (1.29).

$$L(k) = \frac{2^s!}{(2^s - k)!} \quad (1.29)$$

While the total number of unique state assignments [28] is given by $A(k)$ as defined by equation (1.30).

$$A(k) = \frac{(2^s - 1)!}{(2^s - k)! s!} \quad (1.30)$$

Example 1.5: Consider the benchmark Lion (in KISS2 format) which has 4 states as follows:

```
.i 2 // Primary inputs
.o 1 // Primary outputs
.p 11 // Number of rows
.s 4 // Number of states
```

-0 st0 st0 0	11 st1 st0 0	01 st2 st3 1
11 st0 st0 0	10 st1 st2 1	0- st3 st3 1
01 st0 st1 -	1- st2 st2 1	11 st3 st2 1
0- st1 st1 1	00 st2 st1 1	

The STG for this example is shown in Fig. 1.3.

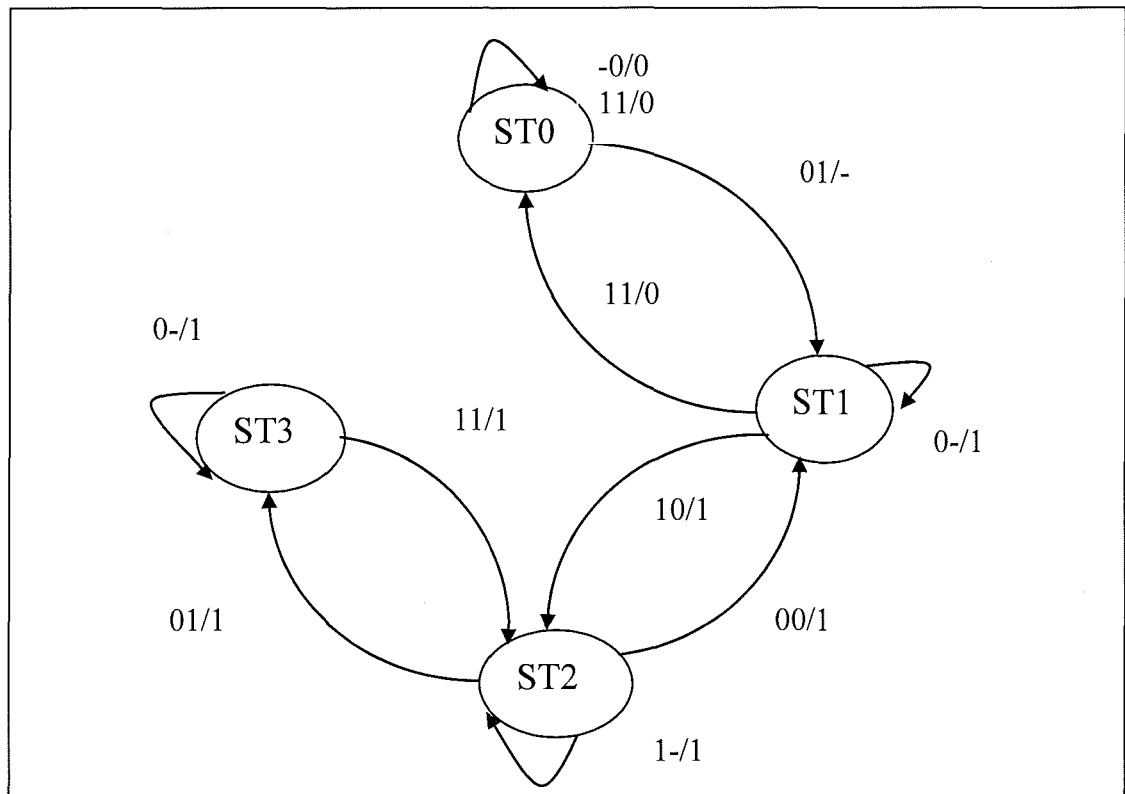


Figure 1.3: STG for Example 1.5

The FSM –STT for this example shown in Table 1.1 consists of four symbolically encoded states $ST0$, $ST1$, $ST2$, and $ST3$. These states can be assigned unique codes using two state variables y_1 and y_2 . The inputs can be represented by x_1 and x_2 and the single output is represented by Z . The next states are represented by y_1^+ and y_2^+ .

Table 1.1: FSM - STT representation of Example 1.5

Present states y_1, y_2	Next states (y_1^+, y_2^+) Inputs x_1x_2 / Output Z			
	00	01	11	10
ST0	ST0/0	ST1/-	ST0/0	ST0/0
ST1	ST1/1	ST1/1	ST0/0	ST2/1
ST2	ST1/1	ST3/1	ST2/1	ST2/1
ST3	ST3/1	ST3/1	ST2/1	-/-

1.3 Literature Survey

1.3.1 Combinational Circuits

There have been several techniques and algorithms developed for synthesis and optimisation of completely and incompletely specified Boolean function.

In [13], the authors developed fast Tabular techniques for Boolean to fixed polarity RM conversion. The motivation behind this technique is using the inherent parallel processing by generating new terms and updating the index table concurrently without the need to wait for all new terms to be generated,. The experimental results show that this technique achieves results consuming less CPU time compared with Tabular technique.

In [18] efficient techniques and algorithms for conversion between standard Boolean expressions and FPRM expansions are developed. The authors

also present new technique for optimisation of FPRM. These techniques are implemented using PASCAL language.

In [19], the authors proposed serial Tabular technique (S_TT) and parallel Tabular technique (P_TT) which may be used for conversion of Boolean function into FPDRM. S_TT deals with one variable at a time and can be used for functions with large number of terms. While in P_TT, generating new terms in parallel may suit large functions with large number of variables.

The authors of [20, 38] proposed a new technique for converting the standard CPOS Boolean functions into FPDRM form. They also proposed a non exhaustive technique to find the optimal polarity among 2^n FPDRM without generating all the polarity set. These techniques are based on separating the truth vector of CPOS and the use of sparse technique.

In [16, 24], efficient algorithms are proposed to generate the minimal Reed Muller expansions with fixed and mixed polarity for any arbitrary, completely and incompletely specified switching functions. These techniques are based on removing independent variables from the given function, generating the coefficients of PPRM expansion, and finally generating the minimal MPRM form. The authors also proposed new technique to generate all 2^n polarities for FPRM forms starting from the truth vector of standard CSOP Boolean form. This technique is based on minterm separation around the truth vector of CSOP form.

In [21, 22], new methods and algorithms are presented for deriving the KRO expansions either from disjoint reduced CSOP truth table or from the Boolean functions. This method is based on tabular technique and deals with completely and incompletely specified Boolean functions. They could be used to derive different KRO polarities from any other KRO polarity. These algorithms are implemented using PASCAL language.

In [31], the authors introduced the Tabular technique for bidirectional conversion between CSOP and Reed-Muller polynomials and for the derivation of all fixed polarities. This technique works with any number of variables. The proposed technique is simple, systematic, and suitable for hand as well as computer programming. Computer program was implemented using PASCAL language and tested with random functions up to 18 variables. The authors of [33] proposed new technique to generate minimal MPRM starting from an array of disjoint cube for multi-output incompletely specified Boolean functions. This technique is based on new operation called “xlinking” which is based on operations such as merger, exclusion and other logic operation. This technique is tested with examples up to 8 inputs, 8 outputs and 255 minterms.

The authors of [34], investigate the properties of the transform matrices for both SOP and RM expansions. They also present technique to obtain the best polarity of FPRM expression for large multiple-output Boolean functions using gray code based on the properties of the RM matrix.

In [35], decomposition technique has been presented to produce a simplified MPRM format from the conventional CSOP input based on top-down strategy and lead to Reed-Muller programmable logic array implementations for Boolean functions. The decomposition technique is implemented in the C language and tested with MCNC and IWLS'93 Benchmarks. The experimental results show that the decomposition method can produce much better results than Espresso for many test cases.

In [35] a non exhaustive technique is developed to determine the optimal FPRM for completely specified Boolean functions. They also proposed new technique to determine the allocation of don't care terms for incompletely specified Boolean functions resulting in reduced FPRM expansions. In [37], a new equation is derived which can be used to convert

any CPOS to any FPDRM polarity. A new algorithm is also proposed for finding the optimal FPDRM polarity for large functions.

Mapping Transform techniques has been investigated in [39, 40] between CPOS form and PPDRM expansion.

There are other interesting aspects of logic design such as multi-valued and multi-levels, especially the graphical approach based on Binary Decision Diagram (BDD) and RM_BDD as in [41-45]. These, however, are outside the scope of this research.

Recently Evolutionary or Genetic Algorithms (GA) are also used in many application such as optimisation of digital circuits, logic synthesis, computer aided design, test pattern, and FPGA placement due to the high complexity of modern VLSI circuits. There have been several interesting techniques and algorithms introduced during the past few years which were GAs are used for synthesis and optimisation of completely and incompletely specified Boolean functions.

In [46], the authors investigated a new class of 2-level RM expressions called Reduced Kronecker Expressions (RKROs) and a new method using Genetic Algorithm (GA) for their exact minimisation is developed for up to 20 variables. The experimental results show that this technique can produce expansions with fewer terms compared with FPRM and PKRO-RM expansions.

In [47], Hybrid GA is investigated to generate the minimal FPRM for large function. The authors combine GA with greedy heuristic to achieve better results.

In [48], Simulated Annealing and Genetic Algorithm are combined to minimise MPRM expressions. This technique incorporate annealing process into genetic operations to obtain better performance compared with simulated annealing and genetic algorithm alone. This algorithm is

implemented in C language and achieved better results compared with Espresso, Simulated Annealing, and Genetic Algorithm.

In [49], multilevel logic synthesis method has been developed which achieves 100% single stuck at fault testability. FPRM forms were used to build the initial design.

Extensive research has focused on developing techniques for representing and minimising FPRM/FPDRM expansions but very little research has focused on representing and minimising of MPRM forms. Further, to the best of my knowledge, there is no research published on the deriving and optimisation of MPDRM, PKRO_RM, and ESOP expansion. Therefore; the main objective of this thesis is to concentrate on developing techniques for synthesis and optimisation of MPRM, MPDRM, PKRO_RM, and ESOP expansions, which can be used to minimise the combinational circuits. The work done during this research is continuing and enhancing the previous work and should give the designer more flexibility for finding the ideal solution.

1.3.2 Sequential Circuits

State assignment is one of the most important problems which received a great deal of attention from researchers. One of the best known techniques which were used for state assignments issue is that of partitions and decomposition [27, 50], but not all state machines have useful closed partitions and may be minimised using these techniques.

In [51], a new approach is proposed utilising GA with Evolvable Hardware (EHW) to design synchronous sequential circuits with minimal hardware. Firstly GA is used for state assignment problem to produce FSM with minimal number of logic gates. Secondly design the circuit to achieve the functionality. Thirdly EHW is used to evaluate the designed circuits.

Finally the final circuit is assembled. The proposed algorithm tested with different benchmark circuits and achieved good results compared with other automated design tools.

In [52-55], the authors proposed the use of GA to generate state assignments which minimise the gate count and/or power dissipation.

A new comprehensive method consisting of an efficient state minimisation and an efficient synthesis technique for state assignment problem is presented in [56]. The aim for these techniques is to optimise power, area, and delay for next state logic network design. Less area, power and delay is achieved through testing these techniques with different examples.

The authors of [57] proposed a new approach to the synthesis problem for finite state machines with the reduction of power dissipation as a design objective. A finite state machine is decomposed into a number of coupled sub machines. Most of the time, only one of the sub machines will be activated which, consequently, could lead to savings in power consumption. Experimental results show that this approach achieved good results for functions with large number of states.

In [58], the authors present heuristic algorithms for state minimisation of FSM's for a large class of practical examples. The authors discuss two steps of the minimisation procedure, called state mapping and solution shrinking, which play a significant role in delivering an optimally implemented reduced machine.

In [59], the authors present a heuristic for state reduction of incompletely specified finite state machines (ISFSMs). The proposed heuristic is based on a branch-and-bound search technique and identification of sets of compatible states of a given ISFSM specification.

A new FSM partitioning approach for low power using GA is presented in [60] to search for an optimal partition. This technique is implemented using C language and achieved 80% less power dissipations.

In [61], a new (m-block) partitioning technique for the state assignment is proposed to improve the testability's and power consumption. This technique achieved good improvement in power consumptions as well as testabilities.

In [62], the usage of a stochastic search technique inspired by simulated annealing is explored to solve the state assignment problem.

In [63], upper and lower bounds for the switching activity are calculated on the state lines of the FSM which depends on the bounds of the average Hamming Distance (HD). The knowledge of these bounds is quite useful in the early stage of the FSM design to indicate the largest and smallest possible power consumption.

In the past, Genetic Algorithms have been applied to find optimal state assignments which minimise the gate count and/or power dissipation for sequential circuits. Genetic algorithm based approach are presented in [64] for finite state machine synthesis targeting to design the circuits with less power consumption. This technique is used to partition the states which result in a state encoding with less HD between them and that will reduce the power consumption of the designed circuit.

In the second part of this research, Multi-Objective GA is used for the state assignment problem using alternative scheme. One of the main reasons for trying to find an optimal state assignment for sequential circuits is to improve the result produced by the recent research which could result in less power consumption.

1.4 Aims and Objectives

The aim of this research is to develop a variety of efficient techniques, which can be used for synthesis and optimisation of combinational and sequential circuits.

The objectives for the first part of this research are as follows:

- Representations of the combinational circuits using MPRM and MPDRM, to widen the search space and achieve better synthesis tools. That's because, in some cases the circuit can be better simplified in DRM expansion using OR/ExNOR forms, whereas for other circuits the reverse will be the case.
- Determining efficient solutions amongst the 3^n polarities in MPRM/MPDRM domains without resorting to exhaustive search.
- Designing the combinational circuit in RM domain using minimal number of terms/or sums may result in smaller, faster or less power than those designed in standard forms of Boolean functions.

The objective for the second part of this research is as follows:

The objective here is regarding the state assignment issue for sequential circuits. There are a huge number of possible and unique state assignments as defined by (1.29) & (1.30) which can be used to design the sequential circuit. The problem is how to find an efficient state assignment among the very large number of different assignments without resorting to exhaustive search. It would require new technique to be developed to identify the good state assignments to design the circuit with fewer components and reduced switching activity simultaneously.

All these synthesis and optimisation techniques could be part of new commercial Electronic Computer Aided Design (ECAD) tool for the future.

1.5 Structure of this thesis

Chapter 2 presents new synthesis techniques for conversion between standard Boolean form and RM form which is categorised as follows:

- Technique for bidirectional conversion between FPRM form and MPRM form for single and multi-output functions. It also can be used to derive any mixed polarity expressions from another mixed polarity expression [25].
- Technique for bidirectional conversion between FPDRM form and MPDRM form for single and multi-output functions. It also can be used to derive any dual mixed polarity expressions from another dual mixed polarity expression [26].
- Technique to generate the coefficients of PKRO-RM starting from PPRM. It can also be used to generate PKRO expansions from standard CSOP Boolean functions [65].

These techniques are based on Tabular technique [31] which was used for bidirectional conversion between CSOP and FPRM and can be used manually or programmed on computers for any number of variables.

Chapter 3 develops new methods [26] to compute the coefficients of MPRM/MPDRM directly from truth vector of the CSOP/CPOS, respectively. Also it may be used to derive any mixed polarity from another MPRM/MPDRM expression. This method is based on separation around minterm/maxterm technique [16, 20] which was used to convert CSOP/CPOS expressions to FPRM/FPDRM.

Chapter 4 investigates new efficient and fast techniques [25, 26] to generate reduced mixed polarity expressions starting from PPRM/PPDRM

respectively for single and multi-output Boolean functions with any number of variables. These new techniques are based on Tabular techniques [31].

Chapter 5 develops an efficient GA based approach [25, 26, and 66] to find the optimal polarity (with fewer terms) among the 3^n expressions without the need to find all the 3^n polarities for the specified function. This technique is implemented for single and multi-output completely and incompletely specified Boolean functions.

Chapter 6 presents a new approach using a Multi Objective Genetic Algorithm (MOGA) [67] to determine the optimal state assignment with less area and power dissipations for completely and incompletely specified sequential circuits. The goal is to find the best assignments which reduce both the component count and switching activity.

Chapter 7 conclusions and future work.

Chapter Two

Extended Tabular techniques for Reed Muller forms Conversions

2.1 Introduction

The Tabular technique [31], which is derived from map folding technique, provides a means of converting standard canonical Sum of Product (CSOP) Boolean functions to any Fixed Polarity Reed Muller (FPRM) expansion and between different fixed polarities. It can also be used to generate CSOP minterms from the FPRM terms. In this Chapter, The Tabular technique is extended to provide new techniques for Mixed Polarity Reed Muller (MPRM) and Mixed Polarity Dual Reed Muller (MPDRM) conversions. The remainder of the Chapter is structured as follows: a review of Tabular technique is given in section 2.2. In section 2.3, the useful Boolean operations of ExOR/ExNOR logic gates are listed. Section 2.4 shows the basic definitions and notations of Kronecker product operation, Pseudo-Kronecker product operation, and Continuous sum operation. New techniques developed for bidirectional conversions between MPRM/MPDRM and FPRM/FPDRM expansions, respectively are explained in section 2.5. Experimental results for optimum polarity among MPRM and MPDRM by running an exhaustive search are given in section 2.6. Section 2.7 shows a new technique for computing the Pseudo-Kronecker (PKRO-RM) expansions starting either from PPRM form or from CSOP form.

2.2 Review of Tabular technique

The Tabular technique procedure for finding general PPRM expansion from Boolean function is summarised bellow [31]:

- *Step 1;* list all minterms in binary form.
- *Step 2;* select any variable x_j . For every term containing a zero in position j , \bar{x}_j , generate additional term with a one in position j for x_j .
- *Step 3;* compare newly generated terms with the ones that already exist and cancel any identical pair.
- *Step 4;* repeat steps 2 and 3 for all other variables. The resulting uncanceled terms are the RM terms.

Example 2.1: Given three-variable CSOP function:

$$f(x_2, x_1, x_0) = \sum m(7,3,1) = x_2x_1x_0 + \bar{x}_2x_1x_0 + \bar{x}_2\bar{x}_1x_0$$

In order to obtain the PPRM terms from CSOP minterms using Tabular technique, new terms are generated and duplicated terms are cancelled as shown in Tables 2.1 & 2.2 as follows:

Table 2.1: Terms generated by variable x_2 for Example 2.1

Minterms			Terms generated by x_2		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	1	1	0	1
0	1	1	1	1	1
1	1	1			

Table 2.2: Terms generated by variable x_1 for Example 2.1

Resulted terms			Terms generated by x_1		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	1	0	1	1
0	1	1			
1	0	1	1	1	1

Note that no terms are generated by variable x_0 . The PPRM expression is shown below:

$$f(x_2, x_1, x_0) = \pi_7 \oplus \pi_5 \oplus \pi_1 = x_2 x_1 x_0 \oplus x_2 x_0 \oplus x_0$$

The Tabular technique can also be used to convert the PPRM terms into CSOP minterms. Furthermore; it can be used to find any polarity as follows:

Step 1; list all terms in binary form.

Step 2; represent the desired polarity using n-bit binary number.

Step 3; select variable x_j whose j^{th} position in the polarity number is one.

For every term containing logic one in position j , x_j , generate additional term with a logic zero in position j for \bar{x}_j .

Step 4; compare newly generated terms with the ones that already exist and cancel any identical pair.

Step 5; repeat steps 3 and 4 for all other variables whose corresponding polarity digit is one. The resulting uncanceled terms are the RM terms for the desired polarity.

Example 2.2: Consider the same Example 2.1, given PPRM expansion as follows:

$$f(x_2, x_1, x_0) = x_2 x_1 x_0 \oplus x_2 x_0 \oplus x_0$$

Assume that $p \langle 6 \rangle_{10} = p \langle 110 \rangle_2$ is required. Variable x_2 and x_1 must be changed from true form to complement form as shown in Tables 2.3 & 2.4.

Table 2.3: Terms generated by variable x_2 for Example 2.2

PPRM terms			Terms generated by x_2		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	1			
1	0	1	0	0	1
1	1	1	0	1	1

Table 2.4: Terms generated by variable x_1 for Example 2.2

FPRM- polarity $\langle 100 \rangle_2$			Terms generated by x_1		
x_2	x_1	x_0	x_2	x_1	x_0
1	0	1			
0	1	1	0	0	1
1	1	1	1	0	1

The remaining terms, which represent polarity-6 for the function, are given as follows:

$$f(\bar{x}_2, \bar{x}_1, x_0) = \pi_7 \oplus \pi_3 \oplus \pi_1 = \bar{x}_2 \bar{x}_1 x_0 \oplus \bar{x}_1 x_0 \oplus x_0$$

2.3 Boolean Operations of ExOR/ExNOR

The RM/DRM expansions are based on ExOR/ExNOR binary operations respectively. These operations are displayed in Table 2.5.

Table 2.5: Boolean Operations

A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

(a) ExNOR

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

(b) ExOR

The ExOR/ExNOR operations have the following properties:

$$A \oplus 1 = \bar{A}$$

$$A \oplus 0 = A$$

$$A \oplus A = 0$$

$$A \oplus \bar{A} = 1$$

$$A \odot 1 = A$$

$$A \odot 0 = \bar{A}$$

$$A \odot A = 1$$

$$A \odot \bar{A} = 0$$

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

$$A \odot B = \overline{A \oplus B} = \bar{A}\bar{B} + AB$$

Associative:

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

$$(A \odot B) \odot C = A \odot (B \odot C) = A \odot B \odot C$$

Distributive:

$$A(B \oplus C) = AB \oplus AC$$

$$A + (B \odot C) = (A + B) \odot (A + C)$$

Commutative:

$$A \oplus B = B \oplus A$$

$$A \odot B = B \odot A$$

2.4 Basic Definitions and Notations

2.4.1 Kronecker Product Operation

The terms of the n-variable PPRM expansion expressed in equation (1.9) can also be written by Kronecker product symbolized by * on the basis vector $[1 \ x_j]$ for $0 \leq j \leq n - 1$, as in equation (2.1):

$$F(x_{n-1}, x_{n-2}, x_0) = \{[1 \ x_{n-1}] * [1 \ x_{n-2}] * \dots * [1 \ x_0]\} \cdot b_i \quad (2.1)$$

In FPRM expansion, the variable can appear in either true or complement form through the RM expansions. Therefore, there are 2^n different FPRM expansions each with different polarity numbers. Hence, any fixed polarity number is represented by (n) digits using a binary number. These different expansions can be expressed by Kronecker product on the following two basis vectors [68] in the form shown below:

$$[1 \ x_j] \quad \text{if } x_j \text{ is in true form} \quad (2.2)$$

$$[1 \ \bar{x}_j] \quad \text{if } x_j \text{ is in complement form} \quad (2.3)$$

The Kronecker product [68] of two vectors is defined as follows:

$$[a \ b] * [e \ f] = [ae \ af \ be \ bf]$$

For example, for $n=3$ and polarity $\langle 6 \rangle_{10}$, the terms of the FPRM expansion can be calculated by:

$$p \langle 6 \rangle_{10} = p \langle 110 \rangle_2 = [1 \ \bar{x}_2] * [1 \ \bar{x}_1] * [1 \ x_0]$$

$$p \langle 6 \rangle_{10} = p \langle 110 \rangle_2 = [1 \ \bar{x}_2] * [1 \ x_0 \ \bar{x}_1 \ \bar{x}_1 x_0]$$

$$p \langle 6 \rangle_{10} = [1 \ x_0 \ \bar{x}_1 \ \bar{x}_1 x_0 \ \bar{x}_2 \ \bar{x}_2 x_0 \ \bar{x}_2 \bar{x}_1 \ \bar{x}_2 \bar{x}_1 x_0]$$

Thus, polarity $\langle 110 \rangle_2$, denotes that variable x_0 appears in true form, while x_1 & x_2 appear in complement form.

In MPRM expansion, the variable can appear in either true, complement, or both (mixed) forms, through the RM expansions. Therefore; there are 3^n different MPRM expansions, each with different polarity numbers. Hence, an extra vector is needed to incorporate the mixed form as follows:

$$[\bar{x}_j \ x_j] \quad \text{if } x_j \text{ is in mixed form} \quad (2.4)$$

The terms of the n -variable MPRM expansion can also be expressed by Kronecker product [66] on the basis of the three basis vectors given by (2.2), (2.3), and (2.4). These MPRM expansions are identified by a polarity number $\langle p_{n-1}, \dots, p_0 \rangle$ and include the 2^n FPRM expansions. Hence, any mixed polarity number is represented by n digits using ternary numbers. For example, for $n=3$ and polarity $\langle 7 \rangle_{10}$, the terms of the MPRM expansion can be calculated by:

$$p \langle 7 \rangle_{10} = p \langle 021 \rangle_3 = [1 \ x_2] * [\bar{x}_1 \ x_1] * [1 \ \bar{x}_0]$$

$$p \langle 7 \rangle_{10} = p \langle 021 \rangle_3 = [1 \ x_2] * [\bar{x}_1 \ \bar{x}_1 \bar{x}_0 \ x_1 \ x_1 \bar{x}_0]$$

$$p \langle 7 \rangle_{10} = [\bar{x}_1 \ \bar{x}_1 \bar{x}_0 \ x_1 \ x_1 \bar{x}_0 \ x_2 \bar{x}_1 \ x_2 \bar{x}_1 \bar{x}_0 \ x_2 x_1 \ x_2 x_1 \bar{x}_0]$$

Thus, polarity $\langle 021 \rangle_3$, denotes that variable x_0 appears in complement form, x_1 appears in mixed form, and x_2 appears in true form.

2.4.2 Pseudo-Kronecker Product Operation

In PKRO-RM expansion, the variable can appear in either true, complement or mixed form through each function or sub function of the RM expansions. Therefore; there are 3^{2^n-1} different PKRO-RM expansions each with different polarity numbers. These PKRO-RM expansions are identified by a polarity number $\langle p_{2^n-2}, \dots, p_0 \rangle$ and include the 2^n FPRM expansions and the 3^n MPRM expansions. Hence, any pseudo polarity number represented by $(2^n - 1)$ digits uses a ternary number. The terms of the n-variable PKRO-RM expansion can be expressed by Pseudo-Kronecker product [67] symbolised by \diamond on the basis of the three vectors which are expressed in equations (2.2), (2.3), and (2.4) depending on the required polarity.

The Pseudo-Kronecker product [69] of two vectors is defined as follows:

$$[a \ b] \diamond \{[e \ f], [g \ h]\} = [a(g \ h), b(e \ f)] = [ag \ ah \ be \ bf]$$

For example, for n=2 and polarity $\langle 14 \rangle_{10}$, the terms of the PKRO-RM expansion can be calculated by:

$$\begin{aligned} p \langle 112 \rangle_3 &= [x_1 \ (\text{complement})] \diamond \{[x_0 \ (\text{complement})], [x_0 \ (\text{mixed})]\} \\ p \langle 14 \rangle_{10} &= p \langle 112 \rangle_3 = [1 \ \bar{x}_1] \diamond \{ [1 \ \bar{x}_0], [\bar{x}_0 \ x_0] \} \\ &= [\bar{x}_0 \ x_0 \ \bar{x}_1 \ \bar{x}_1 \bar{x}_0] \end{aligned}$$

Further, polarity $\langle 112 \rangle_3$, denotes that variable x_1 appears in complement form, while variable x_0 appears in complement form within the true part of variable x_1 , and in mixed form within the complement part of variable x_1 .

Thus, FPRM & MPRM terms for each polarity expansion can be produced by using Kronecker product while PKRO-RM terms for each polarity

expansion can be produced by using Pseudo-Kronecker product between three different vectors which are expressed in (2.2), (2.3), and (2.4) depending on the required polarity.

2.4.3 Continuous Sum Operation

The terms of the n-variable PPDRM expansion expressed in equation (1.11) can also be written by performing Continuous sum operation [20] symbolised \star on the basis vector $[x_j \ 0]$ as shown in equation (2.5).

$$F(x_{n-1}, x_{n-2}, x_0) = \{[x_{n-1} \ 0] \star [x_{n-2} \ 0] \star \dots \star [x_0 \ 0]\} + Q_i \quad (2.5)$$

Where Q_i are the sum terms of the PPDRM. There are 3^n different MPDRM expansions including the 2^n different FPDRM expansions, each with different polarity numbers. These different 3^n expansions can be expressed by Continuous sum operation on the basis of the following three vectors [69] in the form shown below:

$$[x_j \ 0] \quad \text{if } x_j \text{ is in true form} \quad (2.6)$$

$$[\bar{x}_j \ 0] \quad \text{if } x_j \text{ is in complement form} \quad (2.7)$$

$$[x_j \ \bar{x}_j] \quad \text{if } x_j \text{ is in mixed form} \quad (2.8)$$

The Continuous sum operation [20] of two vectors is defined as follows:

$$[a \ b] \star [e \ f] = [a + e \ a + f \ b + e \ b + f]$$

For example, for $n=3$ and polarity $\langle 7 \rangle_{10}$, the terms of the MPDRM expansion can be calculated by:

$$\begin{aligned} p \langle 7 \rangle_{10} &= p \langle 021 \rangle_3 = [x_2 \ 0] \star [x_1 \ \bar{x}_1] \star [\bar{x}_0 \ 0] \\ &= [x_2 \ 0] \star [x_1 + \bar{x}_0 \ x_1 \ \bar{x}_1 + \bar{x}_0 \ \bar{x}_1] \\ p \langle 7 \rangle_{10} &= [x_2 + x_1 + \bar{x}_0 \ x_2 + x_1 \ x_2 + \bar{x}_1 + \bar{x}_0 \ x_2 + \bar{x}_1 \\ &\quad x_1 + \bar{x}_0 \quad x_1 \quad \bar{x}_1 + \bar{x}_0 \quad \bar{x}_1] \end{aligned}$$

Thus, mixed Dual polarity $\langle 021 \rangle_3$, denotes that variable x_0 appears in complement form, x_1 appears in mixed form, and x_2 appears in true form.

2.5 Bidirectional Conversions Techniques

The Tabular technique [31] is extended here to generate MPRM/MPDRM expansions starting from PPRM/PPDRM expansion, respectively. The proposed techniques are suitable for manual computation as well as computer implementation and are applicable to single and multi-output Boolean functions. The polarity of any MPRM/MPDRM expansion can be represented by replacing each variable by 0, 1, or 2 depending on whether the variable is used in true, complement, or mixed form respectively. The polarity will be the decimal equivalent of the resulting ternary number.

2.5.1 Between FPRM & MPRM for single output functions

This method is explained as follows:

1. List all the product terms of the zero polarity RM in binary.
2. Represent mixed polarity by using ternary number

$\langle p_{n-1}, \dots, p_0 \rangle$ as follows:

$$\langle p_j \rangle = \begin{cases} 0, & \text{if } x_j \text{ is in true form} \\ 1, & \text{if } x_j \text{ is in complement form} \\ 2, & \text{if } x_j \text{ is in mixed form} \end{cases}$$

3. Two new extensions are added; the first one is to convert from true fixed to mixed and reverse, and the second is to convert from complement fixed to mixed and reverse depending on the required polarity. The procedure is as follows:

- a) Select a variable x_j to be changed from fixed true to mixed or reverse. For every term containing a zero in position j , \bar{x}_j , generate additional term with a one in position j , $(x_j)_{(new)}$.
 - b) Select a variable x_j to be changed from fixed true to complement or reverse (same as Tabular technique), for every term containing a one in position j , x_j , generate additional term with a zero in position j , (\bar{x}_j) .
 - c) Select a variable x_j to be changed from fixed complement to mixed. Use the methods explained above to convert from complement to true form and then to convert from true to mixed form_(new).
4. Compare the generated terms with original terms and cancel any identical pair of terms. Then add the uncanceled generated terms to the original terms.
 5. Repeat steps 3 and 4 for all other variables. The resulting uncanceled terms are the MPRM product terms.

2.5.2 Between FPRM & MPRM for multi-output functions

The Bidirectional conversion for multi-output Boolean functions from PPRM to MPRM is accomplished by adding an array which stores the outputs for each input. In order to calculate the outputs for the terms after generating new terms, step 4 in section 2.5.1 is changed as follows:

- In case that the new generated term duplicates one of the original terms, perform ExOR operation between the content of the array which contains the outputs for the original terms and the content of the array which contains the outputs for the generated terms. The output of the original terms must be replaced by the EXOR result.

Example 2.3: Consider multi-output PPRM functions as shown below:

$$\begin{aligned}
 f_1 &= 1 \oplus x_0 \oplus x_1 x_0 & f_2 &= x_1 \\
 f_3 &= 1 \oplus x_0 \oplus x_1 \oplus x_1 x_0 & f_4 &= x_0 \oplus x_1 \oplus x_1 x_0
 \end{aligned}$$

The terms for multi-output PPRM functions are shown in Table 2.6.

Table 2.6: PPRM terms for Example 2.3

Inputs		Outputs			
x_1	x_0	f_4	f_3	f_2	f_1
0	0	0	1	0	1
0	1	1	1	0	1
1	0	1	1	1	0
1	1	1	1	0	1

To find MPRM expansion with $p \langle 7 \rangle_{10} = p \langle 21 \rangle_3$, the polarity of variable x_1 must change from fixed to mixed form and the polarity of variable x_0 must change from fixed to complement form. The details of changing the polarity of variable x_1 from fixed true form to mixed form are shown in Table 2.7.

Table 2.7: Terms generated by x_1 for Example 2.3

PPRM						Terms generated by x_1					
Inputs		Outputs				New Terms		Outputs			
x_1	x_0	f_4	f_3	f_2	f_1	x_1	x_0	f_4	f_3	f_2	f_1
0	0	0	1	0	1	1	0	0	1	0	1
0	1	1	1	0	1	1	1	1	1	0	1
1	0	1	1	1	0						
1	1	1	1	0	1						

Performing ExOR operation for the outputs of each identical pair leads to $p \langle 20 \rangle_3 = p \langle 6 \rangle_{10}$ as shown in Table 2.8.

Table 2.8: Terms for Polarity 6 for Example 2.3

Inputs		Outputs			
x_1	x_0	f_4	f_3	f_2	f_1
0	0	0	1	0	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

→
(This term is cancelled)

The details of changing the polarity of variable x_0 from fixed true form to complement form are shown in Table 2.9.

Table 2.9: Terms generated by x_0 for Example 2.3

Polarity 6						Terms generated by x_0					
Inputs		Outputs				New Terms		Outputs			
x_1	x_0	f_4	f_3	f_2	f_1	x_1	x_0	f_4	f_3	f_2	f_1
0	0	0	1	0	1						
0	1	1	1	0	1	0	0	1	1	0	1
1	0	1	0	1	1						

Performing ExOR operation for the outputs of each identical pair leads to $p \langle 21 \rangle_3 = p \langle 7 \rangle_{10}$ as shown in Table 2.10.

Table 2.10: Terms of Polarity 7 for Example 2.3

Inputs		Outputs			
x_1	x_0	f_4	f_3	f_2	f_1
0	0	1	0	0	0
0	1	1	1	0	1
1	0	1	0	1	1

Hence, the MPRM form in $p \langle 7 \rangle_{10}$ are as follows:

$$\begin{aligned}
 f_1 &= \bar{x}_0 \bar{x}_1 \oplus x_1 & f_2 &= x_1, \\
 f_3 &= \bar{x}_1 \bar{x}_0 & f_4 &= \bar{x}_0 \bar{x}_1 \oplus x_1 \oplus \bar{x}_1
 \end{aligned}$$

As a result, the designing of the multi-output MPRM functions; Polarity 7 need three unique terms only.

To prove that all the polarities derived represent the same function:

1) In FPRM, $p < 0 >_{10}$

$$\begin{aligned}
 f_1 &= 1 \oplus x_0 \oplus x_1 x_0 \\
 &= 1 \oplus x_0 \bar{x}_1 \\
 &= 1 \oplus \bar{x}_1 \oplus \bar{x}_1 \bar{x}_0 \\
 &= x_1 \oplus \bar{x}_0 \bar{x}_1 \quad \text{which is the same expression for } f_1 \text{ in MPRM for } p < 21 >_3.
 \end{aligned}$$

2) Note that f_2 is same in FPRM & MPRM.

$$\begin{aligned}
 3) f_3 &= 1 \oplus x_0 \oplus x_1 \oplus x_1 x_0 \\
 &= \bar{x}_0 \oplus \bar{x}_0 x_1 \\
 &= \bar{x}_1 \bar{x}_0 \quad \text{which is the same as } f_3 \text{ in MPRM for } p < 21 >_3
 \end{aligned}$$

4) In FPRM, $p < 0 >_{10}$

$$f_4 = x_1 \oplus x_0 \oplus x_1 x_0$$

In MPRM, $p < 21 >_3$,

$$\begin{aligned}
 f_4 &= \bar{x}_0 \bar{x}_1 \oplus x_1 \oplus \bar{x}_1 \\
 &= \bar{x}_1 (x_0 \oplus 1) \oplus \bar{x}_1 \oplus x_1 \\
 &= \bar{x}_1 x_0 \oplus \bar{x}_1 \oplus \bar{x}_1 \oplus x_1 \\
 &= x_1 \oplus x_0 (x_1 \oplus 1) \\
 &= x_1 \oplus x_0 \oplus x_1 x_0 \quad \text{which is the same as } f_4 \text{ in FPRM for } p < 0 >_{10}
 \end{aligned}$$

2.5.3 Demonstration of the Bidirectional Conversion for MPRM

The standard CSOP Boolean function can be assembled into 3^n mixed polarities including the 2^n fixed polarities. Equations for the fixed and

mixed polarity RM expansions are listed below to compare terms between each of them. For a function with two variables the RM expansions will be as follows:

The terms of fixed true polarity are given in equation (2.9), the terms of fixed complement polarity are given in equation (2.10), while the terms of mixed polarity are given in equation (2.11)

$$[1 \ x_1] * [1 \ x_0] = [1 \ x_0 \ x_1 \ x_1x_0] \quad (2.9)$$

$$[1 \ x_1] * [1 \ \bar{x}_0] = [1 \ \bar{x}_0 \ x_1 \ x_1\bar{x}_0] \quad (2.10)$$

$$[\bar{x}_1 \ x_1] * [1 \ x_0] = [\bar{x}_1 \ \bar{x}_1x_0 \ x_1 \ x_1x_0] \quad (2.11)$$

It can be seen that 0/1 in fixed means absent/present while 0/1 in mixed means complement/true. By comparing equations (2.9) and (2.10), it is evident that variable x_1 in (2.11) has mixed polarity while it has fixed true in (2.9). The last two product terms for the two equations are the same while the first two terms are different.

Further,

$$\begin{array}{lclcl} \text{Term in (2.9)} & \oplus & \text{new term} & = & \text{Term in (2.11)} \\ 1 & \oplus & x_1 & = & \bar{x}_1 \\ x_0 & \oplus & x_1 x_0 & = & x_0 \bar{x}_1 \end{array}$$

It can be concluded that changing the polarity of variable x_1 from fixed true to mixed can be achieved by looking for any zeros related to variable x_1 to generate new terms with a one in the position of variable x_1 .

2.5.4 Between FPDRM & MPDRM for single output functions

This method is explained as follows:

1. List all the sum terms of the PPDRM in binary
2. Represent dual mixed polarity by using ternary number
 $\langle p_{n-1}, \dots, p_0 \rangle$ as shown in section 2.4.1.
3. Two new extensions are added to the Tabular technique; the first one is to convert from true dual fixed to dual mixed and reverse, and the second is to convert from dual complement fixed to dual mixed and reverse depending on the required polarity. The procedure is as follows:
 - a) Select a variable x_j to be changed from dual fixed true to dual complement or reverse (same as in Tabular Technique), for every term containing a zero in position j , x_j , generate additional term with a one in position j , \bar{x}_j .
 - b) Select a variable x_j to be changed from dual fixed true to dual mixed or reverse, for every term containing a one in position j , \bar{x}_j , generate additional term with a zero in position j , $x_{j \text{ (new)}}$.
 - c) Select a variable x_j to be changed from dual fixed complement to dual mixed. Use the methods explained in sections (a) and (b) to convert from dual complement to dual true and then to convert from dual true to dual mixed form_(new).
4. Compare the generated terms with original terms and cancel any identical pair of terms. Then add the uncanceled generated terms to the original terms.
5. Repeat steps 3 and 4 for all other variables. The resulting uncanceled terms are the MPDRM terms.

2.5.5 Between FPDRM & MPDRM for multi-output functions

Bidirectional conversion for multi-output Boolean functions from FPDRM to MPDRM is accomplished by adding an array which stores the outputs for each input. In order to calculate the outputs for the terms after generation new terms, step 4 in section 2.5.4 was changed as follows:

- In case that the new generated term duplicates one of the original terms, perform ExNOR operation between the content of the array which contains the outputs for the original terms and the content of the array which contains the outputs for the generated terms. The output of the original terms must be replaced by the EXOR result. Note that in CPOS, 0 means that this term is present.

Example 2.4: Consider multi-output PPDRM functions as shown below:

$$f_1 = 0, \quad f_2 = (x_1 + x_0) \odot x_0 \odot 0, \\ f_3 = (x_1 + x_0) \quad , \quad f_4 = (x_1 + x_0) \odot x_0$$

The terms for multi-output PPDRM functions are shown in Table 2.11.

Table 2.11: PPDRM Terms for Example 2.4

(This term doesn't exist) \longrightarrow

Inputs		Outputs			
x_1	x_0	f_4	f_3	f_2	f_1
0	0	0	0	0	1
0	1	1	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

To find its MPDRM expansion with $p \langle 7 \rangle_{10} = p \langle 21 \rangle_3$, the polarity of variable x_1 must change from dual fixed to dual mixed and the polarity of

variable x_0 must change from dual fixed to dual complement form. The details of changing the polarity of variable x_1 from dual fixed true form to dual mixed form are shown in Table 2.12.

Table 2.12: Terms generated by x_1 for Example 2.4

PPDRM						Terms generated by x_1					
Inputs		Outputs				New Terms		Outputs			
x_1	x_0	f_4	f_3	f_2	f_1	x_1	x_0	f_4	f_3	f_2	f_1
\emptyset	\emptyset	0	0	0	1						
1	0	0	1	0	1	\emptyset	\emptyset	0	1	0	1
1	1	1	1	0	0	0	1	1	1	0	0

Performing ExNOR operation for the outputs of each identical pair leads to $p \langle 6 \rangle_{10} = p \langle 20 \rangle_3$ as shown in Table 2.13.

Table 2.13: Terms for Polarity 6 for Example 2.4

Inputs		Outputs			
x_1	x_0	f_4	f_3	f_2	f_1
0	0	1	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	1	1	1	0	0

The details of changing the polarity of variable x_0 from dual fixed true form to dual form are shown in Table 2.14.

Table 2.14: Terms generated by x_0 for Example 2.4

Polarity 6						Terms generated by x_0					
Inputs		Outputs				New terms		Outputs			
x_1	x_0	f_4	f_3	f_2	f_1	x_1	x_0	f_4	f_3	f_2	f_1
0	0	1	0	1	1	0	1	1	0	1	1
0	1	1	1	0	0						
1	0	0	1	0	1	1	1	0	1	0	1
1	1	1	1	0	0						

Performing ExNOR operation for the outputs of each identical pair leads to $p \langle 7 \rangle_{10} = p \langle 21 \rangle_3$ as shown in Table 2.15.

Table 2.15: Terms for Polarity 7 for Example 2.4

Inputs		Outputs			
x_1	x_0	f_4	f_3	f_2	f_1
0	0	1	0	1	1
0	1	1	0	0	0
1	0	0	1	0	1
1	1	0	1	1	0

Hence, the MPDRM form in $p \langle 7 \rangle_{10}$ have the following expressions:

$$\begin{aligned}
 f_1 &= \bar{x}_1 \odot x_1, & f_2 &= x_1 \odot (\bar{x}_1 + \bar{x}_0) \\
 f_3 &= (x_1 + \bar{x}_0) \odot x_1 & f_4 &= (\bar{x}_1 + \bar{x}_0) \odot \bar{x}_1
 \end{aligned}$$

As a result, the designing of the multi-output MPDRM functions; Polarity 7 need three unique terms only.

To prove that all the polarities derived represent the same function:

- 1) In MPDRM $p \langle 7 \rangle_{10}$, $f_1 = \bar{x}_1 \odot x_1 = 0$ same as f_1 for FPDRM

$$\begin{aligned}
 2) \text{ In FPDRM , } p < 0 >_{10}, f_2 &= (x_1 + x_0) \odot x_0 \odot 0 \\
 &= (x_1 + x_0) \odot \bar{x}_0 \\
 &= (x_1 \odot \bar{x}_0) + (x_0 \odot \bar{x}_0) \\
 &= x_1 \odot \bar{x}_0
 \end{aligned}$$

$$\begin{aligned}
 \text{In MPDRM , } p < 7 >_{10}, f_2 &= x_1 \odot (\bar{x}_1 + \bar{x}_0) \\
 &= (x_1 \odot \bar{x}_1) + (x_1 \odot \bar{x}_0) \\
 &= x_1 \odot \bar{x}_0
 \end{aligned}$$

$$3) \text{ In FPDRM , } p < 0 >_{10}, f_3 = (x_1 + x_0)$$

$$\begin{aligned}
 \text{In MPDRM , } p < 7 >_{10}, f_3 &= (x_1 + \bar{x}_0) \odot x_1 \\
 &= x_1 + x_0
 \end{aligned}$$

$$\begin{aligned}
 4) \text{ In FPDRM , } p < 0 >_{10}, f_4 &= (x_1 + x_0) \odot x_0 \\
 &= x_1 + \bar{x}_0
 \end{aligned}$$

$$\begin{aligned}
 \text{In MPDRM , } p < 7 >_{10}, f_4 &= (\bar{x}_1 + \bar{x}_0) \odot \bar{x}_1 \\
 &= \bar{x}_1(\bar{x}_1 + \bar{x}_0) + x_1 \overline{(\bar{x}_1 + \bar{x}_0)} \\
 &= \bar{x}_1 + \bar{x}_1 \bar{x}_0 + x_1 x_0 \\
 &= \bar{x}_0 + x_0 x_1 \\
 &= x_1 + \bar{x}_0
 \end{aligned}$$

2.5.6 Demonstration of the Bidirectional Conversion for MPDRM

The Boolean function can be assembled into 3^n mixed dual polarities.

These terms can be identified by performing Continuous sum operation \star on vectors of equations from (2.6) to (2.8).

For a function with two variables the PPDRM expansions will be as follows:

$$[x_1 \ 0] \star [x_0 \ 0] = [x_0 + x_1 \quad x_1 \quad x_0 \quad 0]$$

Dual Fixed Polarity (2.12)

If the variable x_1 appears in mixed and variable x_0 appears in true form, then the equation for MPDRM with Polarity $\langle 6 \rangle_{10} = \langle 20 \rangle_3$ will be expressed as in equation (2.13).

$$[x_1 \bar{x}_1] \star [x_0 \ 0] = [x_0 + x_1 \quad x_1 \quad x_0 + \bar{x}_1 \quad \bar{x}_1] \quad \text{Dual Mixed polarity} \quad (2.13)$$

It can be seen that 0/1 in FPDRM denotes present /absent, while 0/1 in MPDRM denotes true/complement. Comparing equations (2.12) and (2.13), it is obvious that variable x_1 in equation (2.13) has mixed polarity while it has fixed polarity in equation (2.12). The first two sum terms for the two equations are the same while the last two terms are different. Further,

$$\begin{aligned} \text{Term in (2.12)} \quad \odot \quad \text{New Term} &= \text{Term in (2.13)} \\ 0 \quad \odot \quad x_1 &= \bar{x}_1 \\ x_0 \quad \odot \quad (x_0 + x_1) &= x_0 + \bar{x}_1 \end{aligned}$$

It can be concluded that changing the polarity of variable x_1 from fixed to mixed form can be achieved by looking for any logic one related to variable x_1 to generate a new terms with a logic zero in the position of variable x_1 .

If both variables appear in mixed form, then the equation for MPDRM with $p \langle 8 \rangle_{10} = p \langle 22 \rangle_3$ will be as in equation (2.14).

$$[x_1 \bar{x}_1] \star [x_0 \bar{x}_0] = [x_0 + x_1 \quad \bar{x}_0 + x_1 \quad x_0 + \bar{x}_1 \quad \bar{x}_0 + \bar{x}_1] \quad (2.14)$$

Comparing equations (2.13) and (2.14), all the sum terms for the two equations are different. Therefore, for conversion between PPDRM as in equation (2.12) to MPDRM as in equation (2.14), can be done by looking for any logical one related to variable x_1 to generate a new term with

a logical zero in the position of variable x_1 and the same thing have to be done for variable x_0 .

2.6 Experimental Results

These algorithms for MPRM/MPDRM were applied to several MCNC benchmark functions [70-72]. The results are given in column five of Tables 2.16 and 2.17. These algorithms were tested on a personal computer with Intel CPU running at 2.4 GHz and 2 GB RAM under Windows XP, professional. They were implemented using C++ and compiled using Bloodshed Dev C++. An *I/O* denotes the number of inputs/outputs respectively of benchmark name.

PPRM/PPDRM Terms represent the number of fixed true/dual fixed true terms, respectively for the specified function. The number of terms for the optimum mixed/dual mixed polarity running exhaustive search are given in *Terms for optimum MPRM/MPDRM*, respectively.

The CPU time required to run exhaustive search for all the benchmark examples are given in *CPU Time*. *Saving MPRM/MPDRM* represents the saving in mixed/dual mixed terms, respectively, compared with Espresso terms based on equation (2.15)

$$Saving = \frac{Asolute(Espresso Terms - Optimal mixed terms)}{Largest(Espresso Terms, Optimal mixed terms)} \times 100\% \quad (2.15)$$

Largest of (Espresso Terms, Optimal mixed terms): choose either Espresso terms or optimal mixed terms depending on which one is bigger.

Table 2.16: Benchmark results for MPRM

Name	I/O	Espresso Terms	Terms for PPRM	Terms for Optimum MPRM	CPU Time	Saving for MPRM %
Dc1	4/7	15	16	10	0.031 sec.	33
Xor5	5/1	16	5	5	0.031 sec.	69
bw	5/28	87	32	22	0.031 sec.	75
Squar5	5/8	32	23	23	0.031 sec.	28
Con1	7/2	9	19	14	0.125 sec.	-35
Inc	7/9	34	91	34	0.14 sec.	0
newill	8/1	22	57	13	0.766 sec.	41
Misex1	8/7	32	60	13	0.766 sec.	59
Sqrt8	8/4	40	128	26	0.781 sec.	35
Rd84	8/4	256	107	107	0.831 sec.	50
Risc	8/31	74	89	30	0.922 sec.	59
Clip	9/5	167	217	182	5.68 sec.	-8
Apex4	9/19	438	445	444	7.04 sec.	-1
Sym10	10/1	837	266	266	38.985 sec.	68
Sao2	10/4	58	1022	76	38.703 sec.	-24
Ex1010	10/10	284	1023	810	43.594 sec.	-64
Dk17	10/11	31	996	30	39.42 sec.	6
Table3	14/14	162	5504	401	1 day & 2 h.	-59
Alu4	14/8	1028	4406	2438	1 day & 2 h.	-57
Misex3	14/14	1848	6028	1421	1 day & 3 h.	23
T481	16/1	481	41	13	14 days	97
B12	15/9	431	209	64	10 days & 4 h.	85
Table 5	15/17	157	74500	551	14 days & 8 h.	-72

Table 2.17: Benchmark results for MPDRM

Name	I/O	Espresso Terms	Terms for PPDRM	Terms for Optimum MPDRM	CPU Time	Saving for MPDRM %
Dc1	4/7	15	10	2	0.02 sec	86
root	5/8	57	136	83	0.75 sec.	-31
Con1	7/2	9	24	14	0.12 sec.	-52
Inc	7/9	34	56	34	0.12 sec.	0
Misex1	8/7	32	5	5	0.75 sec.	84
Sqrt8	8/4	40	11	11	0.75 sec.	73
Rd84	8/4	256	256	108	0.75 sec.	58
Risc	8/31	74	21	12	0.75 sec.	84
clip	9/5	167	305	174	5.36 sec.	-4
Apex4	9/19	438	512	407	5.34 sec.	7
Ex1010	10/10	284	1023	825	38.32 sec.	-65
Dk17	10/11	32	8	8	38.18 sec.	75
Table3	14/14	162	4553	421	1 day&6 h	-61
Alu4	14/8	1028	1091	496	1 day&4 h	52
Misex3	14/14	1848	6442	803	1 day&1 h	57
B12	15/9	431	10	10	12 days &2 h.	98
Table 5	15/17	157	4421	554	14 days &5 h.	-72

The experimental results for MCNC benchmark circuits, displayed in Tables 2.16 and 2.17 demonstrate that, in some cases, the circuit can be better minimised in DRM expansion using OR/ExNOR forms such as Dc1, Risc and Clip benchmarks, whereas for other circuits, the reverse is the case such as Table3 and Table5 benchmarks. Further, in other cases, the circuit can be better simplified in the standard CSOP Boolean function

using Espresso tools [73] such as Con1 benchmark. The reason for that is some circuits can be simplified in Boolean domain, while others can be simplified either in MPRM or MPDRM domain depending on the structure of the circuits.

2.7 Computing the Pseudo-Kronecker RM expansions

A new fast and efficient technique is proposed to generate the coefficients of PKRO-RM expansion from PPRM. It can also be used to generate PKRO-RM expansions from standard CSOP Boolean functions. This technique can be easily programmed on computers.

The purpose of generating different sets of coefficients related to RM is that they contain different number of terms. Therefore, the minimum one can be selected to design the circuits with reduced gate count. To widen the search space and achieve better synthesis tools, representations of PKRO-RM expansions are investigated.

2.7.1 PKRO-RM expansion from PPRM expansion

This new technique is based on the technique described in section 2.5. The explanations in detail (for $n=3$) are as described below:

1. List all the product terms of the zero polarity RM in binary
2. Represent pseudo polarity number using ternary codes, for 3-variables $f(x_2, x_1, x_0)$, $2^3-1 = 7$ digits to represent the polarity number as shown:
$$\langle p_3 2^{n-2} \rangle_{10} = \langle p_6, p_5, p_4, p_3, p_2, p_1, p_0 \rangle_3$$
3. If the required polarity for variable x_2 represented by (p_6) is complement or mixed, follow the Extended Tabular technique to change its polarity as required. If (p_6) is true, no action is required.
4. Change the polarity for variable x_1 which is represented by two polarity digits $(p_5 \ \& \ p_2)$, p_5 to represent the polarity of x_1 within the true part

of variable x_2 (i.e for all terms with $x_2 = 1$) , while p_2 to represent the polarity of x_1 within the complement part of variable x_2 (i.e for all terms with $x_2 = 0$). Therefore, depending on the requirements, follow the Extended_Tabular steps to change the polarity of variable x_1 in both parts.

5. Follow the same procedure with the polarity for variable x_0 which is represented by four digits (p_4, p_3, p_1 & p_0) , p_4 represents the polarity of x_0 within the true part of both variables x_2 & x_1 (i.e for all terms with $x_2 = x_1 = 1$) , while p_3 represent the polarity of x_1 within the true part of x_2 and complement part of variable x_1 . Similarly, p_1 represents the polarity of x_0 within the complement part of variable x_2 and true part of variable x_1 , while p_0 represents the polarity of variable x_0 within the complement part of both variables x_2 & x_1 . Therefore depending on the requirement, follow the Extended_Tabular steps to change the polarity of variable x_0 in both parts.

This technique could also be used to calculate the coefficients of PKRO-RM from CSOP terms. If all variables exist within the minterm, as in (1.8), follow the Extended_Tabular method [25] and this technique to change the polarity of all variables from mixed form to the required polarity.

Example 2.5: Given 3-variable PPRM function:

$$f(x_2, x_1, x_0) = \oplus \sum(7,2,1,0) = x_2 x_1 x_0 \oplus x_1 \oplus x_0 \oplus 1$$

To find its PKRO-RM expansion with polarity $318 = \langle 1211012 \rangle_3$, using the Extended_Tabular technique [25], the following steps are necessary:

- $\langle 1211012 \rangle$ represent the polarity digits as in $\langle p_6, p_5, p_4, p_3, p_2, p_1, p_0 \rangle$

- Since $p_6 = 1$, change the polarity of variable x_2 from true form to complement form by changing every terms with logic one related to variable x_2 to logic zero as explained by the Extended_Tabular technique [25] as shown in Table 2.18.

Table 2.18: Polarity changing of variable x_2 for Example 2.5

PPRM terms			Generated terms		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	0			
0	0	1			
0	1	0			
1	1	1	0	1	1

- Since $p_5 = 2$, the polarity of variable x_1 within the true part of variable x_2 has to be changed to mixed polarity. Therefore, change the polarity of variable x_1 from true form to mixed form for all terms with $x_2 = 1$, by changing every terms with logic zero related to variable x_1 to logic one. Since $p_2 = 0$ which is related to variable x_1 in the complement part of variable x_2 , no action is required as shown in Table 2.19.

Table 2.19: Polarity changing of variable x_1 for Example 2.5

Polarity <1200000>			Generated terms		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	1	1			

Complement part of x_2 (bracketed rows 1-4)

True part of x_2 (arrow pointing to row 5)

- Since $p_4 \& p_3 = 1$, change the polarity for variable x_0 of both true and complement parts of variable x_1 within the true part of variable x_2 from true to complement form as shown in Table 2.20.

Table 2.20: Polarity changing of x_0 within the true part of x_2 for Example 2.5

Polarity <1200000>			Generated terms		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	1	1			

Complement parts of x_2 & x_1 {

True parts of x_2 & x_1 →

- Since $p_1 = 1$, change the polarity for variable x_0 within the true part of variable x_1 and complement part of variable x_2 , from true form to complement form as shown in Table 2.21.
- Since $p_0 = 2$, change the polarity for variable x_0 in the complement parts of variables x_1 and x_2 from true form to mixed form as shown in Table 2.22.
- The resulting PKRO-RM terms are shown in Table 2.23

Table 2.21: Polarity changing of x_0 within the complement part of x_2 & true part of x_1 for Example 2.5

Polarity <1211000>			Generated terms		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	0			
0	0	1			
0	1	0			
0	1	1	0	1	0
1	1	1			
1	1	0			

Table 2.22: Polarity changing of x_0 within the complement part of x_2 & x_1 for Example 2.5

Polarity <1211010>			Generated terms		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	0	0	0	1
0	0	1			
0	1	1			
1	1	1			
1	1	0			

Table 2.23: Resulting PKRO-RM terms, polarity 318 for Example 2.5

PKRO terms – Polarity <1211012> ₃		
x_2	x_1	x_0
0	0	0
0	1	1
1	1	1
1	1	0

The terms of PKRO-RM expansion-polarity <318>₁₀ can be obtained using Pseudo-Kronecker product as follows:

$$\begin{aligned}
 & [1 \ \bar{x}_2] \diamond ([\bar{x}_1 \ x_1] \diamond \{[1 \ \bar{x}_0], [1 \ \bar{x}_0]\}, [1 \ x_1] \diamond \{[1 \ \bar{x}_0], [\bar{x}_0 \ x_0]\}) \\
 &= [1 \ \bar{x}_2] \diamond ([\bar{x}_1 \ \bar{x}_1\bar{x}_0 \ x_1 \ x_1\bar{x}_0], [\bar{x}_0 \ x_0 \ x_1 \ x_1\bar{x}_0]) \\
 &= [\bar{x}_0 \ x_0 \ x_1 \ x_1\bar{x}_0 \ \bar{x}_2\bar{x}_1 \ \bar{x}_2\bar{x}_1\bar{x}_0 \ \bar{x}_2x_1 \ \bar{x}_2x_1\bar{x}_0]
 \end{aligned}$$

$$\begin{aligned}
 P \langle 1211012 \rangle_3 \rightarrow f(x_2, x_1, x_0) &= \pi_0 \oplus \pi_3 \oplus \pi_7 \oplus \pi_6 \\
 &= \bar{x}_0 \oplus x_1\bar{x}_0 \oplus \bar{x}_2x_1 \oplus \bar{x}_2x_1\bar{x}_0
 \end{aligned}$$

$$\begin{aligned}
 &= x_0 \oplus 1 \oplus x_1x_0 \oplus x_1 \oplus x_2x_1 \oplus x_1 \oplus x_2x_1x_0 \oplus x_1x_0 \oplus x_2x_1 \oplus x_1 \\
 &= x_2x_1x_0 \oplus x_1 \oplus x_0 \oplus 1 = \text{Polarity 0}
 \end{aligned}$$

2.7.2 PKRO-RM expansion from standard CSOP form

This technique could also be used to calculate the coefficients of PKRO-RM from CSOP terms. Therefore, it is possible to find any polarity related to PKRO-RM form directly from the CSOP form.

Example 2.6: Given 2-variable, CSOP Boolean function:

$$f(x_1, x_0) = \sum m(3,1,0) = x_1x_0 + \bar{x}_1x_0 + \bar{x}_1\bar{x}_0$$

If all variables exist within minterm, OR gate (+) can be replaced by ExOR gate (\oplus), then the function can be rewritten as shown below:

$$f(x_1, x_0) = x_1x_0 \oplus \bar{x}_1x_0 \oplus \bar{x}_1\bar{x}_0$$

This expression represents ExOR CSOP with $p \langle 8 \rangle_{10} = p \langle 22 \rangle_3$, with both variables in mixed form. To find its PKRO-RM expansion with polarity $\langle 11 \rangle_{10} = \langle 102 \rangle_3$, using the Extended_Tabular technique [25], the following steps are followed:

- The polarity of variable x_1 must change from mixed form to complement form using the Extended_Tabular technique described in section 2.4, changing the polarity of variable x_1 from mixed to fixed true as shown in Table 2.24, and then change it from fixed true form to complement form as shown in Table 2.25.

Table 2.24: Changing Polarity of x_1 from mixed to true for Example 2.6

MPRM terms-polarity $\langle 22 \rangle_3$		Generated terms	
x_1	x_0	x_1	x_0
0	0	1	0
0	1	1	1
1	1		

Table 2.25: Changing Polarity of x_1 from true to complement for Example 2.6

MPRM terms-polarity $\langle 02 \rangle_3$		Generated terms	
x_1	x_0	x_1	x_0
0	0		
0	1		
1	0	0	0

- The polarity of variable x_0 is already mixed in complement and true part of variable x_1 , therefore no action is required to change its polarity in the complement part of variable x_1 , while the polarity of variable x_0 in the true part of variable x_1 should be changed to true as shown in Table 2.26.

Table 2.26: Changing Polarity of x_0 from mixed to true in the true part of x_1 for Example 2.6

MPRM terms-polarity $\langle 12 \rangle_3$		Generated terms	
x_1	x_0	x_1	x_0
0	1		
1	0	1	1

- Finally the resulting PKRO-RM terms are shown in Table 2.27.

Table 2.27: PKRO-RM terms, Polarity 11 for Example 2.6

PKRO-RM , Polarity $\langle 102 \rangle_3$	
x_1	x_0
0	1
1	0
1	1

Using Pseudo-Kronecker product:

$$[1 \ \bar{x}_1] \diamond \{ [1 \ x_0], [\bar{x}_0 \ x_0] \} = [\bar{x}_0 \ x_0 \ \bar{x}_1 \ \bar{x}_1 x_0]$$

$$\begin{aligned} p < 102 >_3 = p < 11 >_{10} \rightarrow f(x_1, x_0) &= \pi_1 \oplus \pi_2 \oplus \pi_3 \\ &= x_0 \oplus \bar{x}_1 \oplus \bar{x}_1 x_0 \end{aligned}$$

$$\begin{aligned} p < 8 >_{10} \rightarrow f(x_1, x_0) &= \oplus \sum m(3,1,0) = x_1 x_0 \oplus \bar{x}_1 x_0 \oplus \bar{x}_1 \bar{x}_0 \\ &= \bar{x}_1 x_0 \oplus \bar{x}_1 \oplus \bar{x}_1 x_0 \oplus \bar{x}_1 x_0 \oplus x_0 \\ &= x_0 \oplus \bar{x}_1 \oplus \bar{x}_1 x_0 = p < 11 >_{10} \end{aligned}$$

2.8 Summary

The definitions and notations for the Kronecker product [68], Pseudo-Kronecker product [69] and Continuous sum operation [20] are given through this Chapter. These operations can be used to find the terms related to any polarity for MPRM, PKRO-RM, and MPDRM expansions, respectively.

This Chapter presents new techniques and algorithms for bidirectional conversion between FPRM/FPDRM logic functions and MPRM/MPDRM, respectively [25, 26]. It can also be used to derive any mixed polarity from another mixed polarity for any number of variables for single and multi output functions. These techniques are implemented using C++ language and fully tested using standard benchmark examples.

Further, a new and efficient technique is presented to generate the coefficients of PKRO-RM starting from PPRM [65]. It can also be used to generate PKRO-RM expansions from standard CSOP Boolean functions.

All these techniques are based on Tabular technique [31] and can be used manually or programmed on computers.

Chapter Three

Minterm/Maxterm Separation Techniques for RM forms Conversion

3.1 Introduction

Techniques presented in [16, 20] were developed to convert CSOP/CPOS expressions to FPRM/FPDRM forms respectively. These are extended in this Chapter to Minterm/Maxterm separation techniques which are straightforward techniques to compute the coefficients of MPRM/MPDRM directly from truth vector of the CSOP/CPOS form respectively. Also it can be used to derive any mixed polarity from another MPRM/MPDRM expression.

The significant advantages for this approach are its suitability for computer implementation and its ability to deal for any number of variables.

The rest of the Chapter is organised as follows. Boolean matrix representation is given in section 3.2. In section 3.3, Minterm separation method is explained in details. Section 3.4 demonstrates the Maxterm separation method. Experimental Results for optimum polarity using Minterm/Maxterm separation technique through doing exhaustive search for benchmarks with up to 16 inputs and 28 outputs are given in section 3.5.

3.2 Boolean matrix representation

The proposed synthesis techniques are based on the Boolean matrix representation. The following principles and derivations are essential to be used for computing the coefficients of the MPRM/MPDRM from the standard form of CSOP/CPOS Boolean function, respectively.

For n-variable Boolean function , CSOP equation given in equation (1.1) & CPOS equation given in equation (1.4) can be re-written in expanded form using ExOR/ExNOR logic gates as in equation (3.1) & (3.2) as shown:

$$F(x_{n-1}, x_{n-2}, x_0) = a_0 \bar{x}_{n-1} \dots \bar{x}_1 \bar{x}_0 \oplus a_1 \bar{x}_{n-1} \bar{x}_{n-2} \dots \bar{x}_1 x_0 \oplus \dots \oplus a_{(2^n-1)} x_{n-1} \dots x_1 x_0 \quad (3.1)$$

$$F(x_{n-1}, x_{n-2}, x_0) = (d_0 + x_{n-1} + \dots x_1 + x_0) \odot (d_1 + x_{n-1} + \dots x_1 + \bar{x}_0) \odot \dots \odot (d_{(2^n-1)} + \bar{x}_{n-1} + \dots + \bar{x}_1 + \bar{x}_0) \quad (3.2)$$

Definition 3.1: The Boolean functions of equations (3.1) and (3.2) can be represented in terms of a coefficient matrix, using the Boolean matrix representation [74] by one column 2^n row matrix containing the coefficient of product/or sum terms . The coefficient matrix is the same as the truth vector of the Boolean function. Accordingly, the coefficients matrix for an n _variable Boolean function is represented as:

- In CSOP, the coefficient matrix can be represented as

$$C_M = [a_0 \quad a_1 \quad a_2 \quad a_3 \dots \dots \quad a_{(2^n-1)}] \quad (3.3)$$

- In CPOS, the coefficient matrix can be represented as

$$C_M = [d_0 \quad d_1 \quad d_2 \quad d_3 \dots \dots \quad d_{(2^n-1)}] \quad (3.4)$$

The elements of matrix $[C_M]$ are placed in the order of decimal equivalent to the binary coding of the product/or sum terms.

Definition 3.2: In equations (3.1) and (3.2), half of the product or sum terms contains the variables $x_j, j = 0,1,2..n - 1$, in complement form, while the other half contains the same variables x_j in un-complemented (true) form. The complemented and the true forms of each variable in equations (3.1) and (3.2) are organised in blocks. These blocks are classified as odd and even blocks [16, 20].

- In case of CSOP form, the odd blocks contain the coefficients of the product terms related to complement part of the variable and the even blocks contain the coefficients of the product terms related to true part of the same variable.
- In case of CPOS form, the odd blocks contain the coefficients of the sum terms related to true part of the variable and the even blocks contain the coefficients of the sum terms related to complement part of the selected variable.

Definition 3.3: Each variable x_j , divides the coefficients matrix $[C_M]$ into B blocks. The number of blocks within the coefficient matrix $[C_M]$ can be computed as $B = 2^{(n-j)}$, denoted by $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_B$ and the size of each block $S_\alpha = 2^n/B$ [16, 20].

Definition 3.4: For any Boolean function in equations (3.1) and (3.2), the coefficients matrix $[C_M]$ can be separated into two rows for each variable x_j and the result stored in a partitioned matrix $R(x_j)$. The first row of the partitioned matrix $R(x_j)$ will contain the functional values of the minterm/maxterm that include the odd blocks of variable (x_j) while the second row of this matrix will contain the functional values of the minterm/maxterm for the even blocks of the same variable [16, 20].

$$R(x_j) = \begin{bmatrix} \alpha_1 & \alpha_3 & \alpha_5 & \dots & \alpha_q \\ \alpha_2 & \alpha_4 & \alpha_6 & \dots & \alpha_r \end{bmatrix} \quad (3.5)$$

Where $q=r-1, r=2^{(n-j)}$

Definition 3.5:

- In case of CSOP form, perform ExOR operation between the elements in the first and second rows of R (x_j) matrix and store the result vector in T(x_j). This operation is known as “Minterm separation around variable x_j ”. This operation is illustrated by Example 3.1.
- In case of CPOS form, perform ExNOR operation between the elements in the first and second rows of R (x_j) matrix and store the resulting vector in T(x_j). This operation is known as “Maxterm separation around variable x_j ”. This operation is illustrated by Example 3.2.

Example 3.1: For 3-variables, the coefficient matrix $[C_M]$ for $f(x_2, x_1, x_0) = \sum(7,6,2,0)$ has $= 2^3$ elements = 8 elements. Each product term corresponds to ‘1’ in the coefficient matrix. Hence, $[C_M]$ can be represented by

$$C_M = [a_0 \ a_1 \ a_2 \ a_3 \ a_4 \ a_5 \ a_6 \ a_7]$$

$$= [1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1]$$

Each variable x_j divides the C_M into α Blocks, Number of Blocks $B = 2^{(n-j)}$ as explained in Definition 3.3. Accordingly,

- For variable x_2 , Number of Blocks $B = 2^{(3-2)} = 2$, and size of each block $S_\alpha = 2^3/2 = 4$ elements in length. Therefore; there are 2 blocks, each with 4 elements, $C_M = [1010 \ 0011]$

- For variable x_1 , Number of Blocks $B= 2^{(3-1)} = 4$, and size of each block $S_\alpha= 2^3/4 = 2$ elements in length. Therefore; there are 4 blocks , each with 2 elements , $C_M=[10 \ 10 \ 00 \ 11]$
- For variable x_0 , Number of Blocks $B= 2^{(3-0)} = 8$, and size of each block $S_\alpha= 2^3/8 = 1$ element in length . Therefore; there are 8 blocks , each with one element , $C_M=[1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1]$

Each variable x_j appears complement in half of the terms and divides the $[C_M]$ into Blocks $B= 2^{(n-j)}$ as explained in Definition 3.3. Accordingly,

- For variable x_2 , The partitioned matrix $R(x_2)$ can be represented by

$$R(x_2) = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}$$

$$R(x_2) = \begin{bmatrix} 1010 \\ 0011 \end{bmatrix}$$

α_1 represents the complement part (odd block) for variable x_2 , while α_2 represents the un-complemented part (even block) for variable x_2 .

- For variable x_1 , The partitioned matrix $R(x_1)$ can be represented by

$$R(x_1) = \begin{bmatrix} \alpha_1 & \alpha_3 \\ \alpha_2 & \alpha_4 \end{bmatrix}$$

$$R(x_1) = \begin{bmatrix} 10 & 00 \\ 10 & 11 \end{bmatrix}$$

α_1 & α_3 represent the complement parts (odd blocks) for variable x_1 , while α_2 & α_4 represent the un-complemented parts (even blocks) for variable x_1 .

- For variable x_0 , The partitioned matrix $R(x_0)$ can be represented by

$$R(x_0) = \begin{bmatrix} \alpha_1 & \alpha_3 & \alpha_5 & \alpha_7 \\ \alpha_2 & \alpha_4 & \alpha_6 & \alpha_8 \end{bmatrix}$$

$$R(x_0) = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\alpha_1, \alpha_3, \alpha_5$ & α_7 represent the complement parts (odd blocks) for variable x_0 , while $\alpha_2, \alpha_4, \alpha_6$ & α_8 represent the un-complemented parts (even blocks) for variable x_0 .

Example 3.2: for 4-variables CPOS Boolean function

$$f(x_3, x_2, x_1, x_0) = \prod (15, 11, 7, 6, 5, 0)$$

The coefficient matrix $[C_M]$ has $2^4 = 16$ elements. Each sum term corresponds to '0' in the coefficient matrix. Hence, $[C_M]$ can be represented by:

$$\begin{aligned} C_M &= [d_0 \ d_1 \ d_2 \ d_3 \ d_4 \ d_5 \ d_6 \ d_7 \ d_8 \ d_9 \ d_{10} \ d_{11} \ d_{12} \ d_{13} \ d_{14} \ d_{15}] \\ &= [0111 \ 1000 \ 1110 \ 1110] \end{aligned}$$

Each variable x_j divides the $[C_M]$ into Blocks $B = 2^{(n-j)}$ as explained in definition 3.3. Accordingly,

- For variable x_3 , Number of Blocks $B = 2^{(4-3)} = 2$, and size of each block $S_\alpha = 2^4/2 = 8$ elements in length.
- For variable x_2 , Number of Blocks $B = 2^{(4-2)} = 4$, and size of each block $S_\alpha = 2^4/4 = 4$ elements in length.
- For variable x_1 , Number of Blocks $B = 2^{(4-1)} = 8$, and size of each block $S_\alpha = 2^4/8 = 2$ elements in length.
- For variable x_0 , Number of Blocks $B = 2^{(4-0)} = 16$, and size of each block $S_\alpha = 2^4/16 = 1$ element in length.

Each variable x_j appears complement in half of the terms and divides the C_M into blocks $B = 2^{(n-j)}$ as explained in Definition 3.3. Accordingly,

- For variable x_3 , The partitioned matrix $R(x_3)$ can be represented by

$$R(x_3) = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}$$

α_1 represents the un-complemented part (odd block) for variable x_3 which is = [0111 1000], while α_2 represents the complemented part (even block) for variable x_3 which is = [1110 1110].

- For variable x_2 , The partitioned matrix $R(x_2)$ can be represented by

$$R(x_2) = \begin{bmatrix} \alpha_1 & \alpha_3 \\ \alpha_2 & \alpha_4 \end{bmatrix}$$

$$R(x_2) = \begin{bmatrix} 0111 & 1110 \\ 1000 & 1110 \end{bmatrix}$$

α_1 & α_3 represent the true parts (odd blocks) for variable x_2 , while α_2 & α_4 represent the complement parts (even blocks) for variable x_2 .

- For variable x_1 , The partitioned matrix $R(x_1)$ can be represented by

$$R(x_1) = \begin{bmatrix} \alpha_1 & \alpha_3 & \alpha_5 & \alpha_7 \\ \alpha_2 & \alpha_4 & \alpha_6 & \alpha_8 \end{bmatrix}$$

$$R(x_1) = \begin{bmatrix} 01 & 10 & 11 & 11 \\ 11 & 00 & 10 & 10 \end{bmatrix}$$

$\alpha_1, \alpha_3, \alpha_5$ & α_7 represent the un-complemented parts (odd blocks) for variable x_1 , while $\alpha_2, \alpha_4, \alpha_6$ & α_8 represent the complement parts (even blocks) for variable x_1 .

- For variable x_0 , The partitioned matrix $R(x_0)$ can be represented by

$$R(x_0) = \begin{bmatrix} \alpha_1 & \alpha_3 & \alpha_5 & \alpha_7 & \alpha_9 & \alpha_{11} & \alpha_{13} & \alpha_{15} \\ \alpha_2 & \alpha_4 & \alpha_6 & \alpha_8 & \alpha_{10} & \alpha_{12} & \alpha_{14} & \alpha_{16} \end{bmatrix}$$

$$R(x_0) = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$\alpha_1, \alpha_3, \alpha_5 \dots$ & α_{15} represent the un-complemented parts (odd blocks) for variable x_0 , while $\alpha_2, \alpha_4, \alpha_6 \dots$ & α_{16} represent the complement parts (even blocks) for variable x_0 .

3.3 Minterm Separation Method

In this section, new method is presented which can be used to compute the coefficients of MPRM expansions directly from truth vector of the standard

CSOP Boolean function or from FPRM form. Also it can be used to derive any mixed polarity from another. Further; these methods could be used to calculate the coefficients of standard CSOP Boolean function from any mixed polarity.

3.3.1 Demonstration for the Minterm Separation technique

In this section, a fast, and efficient, method has been invented based on separation of minterms within CSOP expression to compute the b_i coefficients (equation (1.9)) of MPRM expansion from a_i coefficients (equation (1.1)) of CSOP expression, where $i = 0,1,2 \dots (2^n - 1)$. Each MPRM expansion has 2^n coefficients set of b_i , which is associated with product terms of RM expansion as given in equation (1.9). If all variables are present in every term in CSOP function as in (1.1), then the OR can be replaced by ExOR giving equation (3.1). This expression represents MPRM with all variables in mixed form.

For n-variable functions (equation (3.1)), the minterm separation around the variable x_j is shown in equation (3.5) as partitioned matrix $R(x_j)$. The odd blocks which include the terms where the variable x_j appears in complement form only are allocated in the upper row, while the even blocks which include the terms where the variable x_j appears in true form only are allocated in the lower row.

In order to change the variable from standard CSOP form to true form, the complement part has to be eliminated from equation (3.1). To eliminate the complement part for any variable \bar{x}_j , the following formulas can be applied:

$$a \bar{x}_j \oplus b x_j = a (1 \oplus x_j) \oplus b x_j = a \oplus (a \oplus b)x_j \quad (3.6)$$

where $a \& b \in \{0,1\}$. It is clear that the coefficient associated with the complement part will stay unchanged while the coefficient associated with true part has to be ExORed with the coefficient of the complement part. This can be achieved by having the ExOR operation between the upper and lower rows of the partitioned matrix $R(x_j)$, and the elimination of the complement part can be done by replacing the coefficients of the corresponding even blocks of the $[C_M]$ by the ExOR results.

In order to change the variable from standard CSOP form to complement form, the true part has to be eliminated from equation (3.1) as shown below:.

$$a \bar{x}_j \oplus bx_j = a \bar{x}_j \oplus b(1 \oplus \bar{x}_j) = b \oplus (a \oplus b)\bar{x}_j \quad (3.7)$$

It is obvious that in the elimination of the true part of variable x_j , the coefficients associated with the true part will stay unchanged while the coefficients associated with complement part have to be ExORed with the coefficients of the true part.

This can be achieved by also having the ExOR operation between the upper and lower rows of the partitioned matrix $R(x_j)$, and the elimination of the true part can be done by replacing the coefficients of the corresponding odd blocks of the $[C_M]$ by the ExOR results.

The variable x_j in the standard CSOP represents also the mixed form for the given variable. Therefore; in order to change the variable from standard CSOP form to mixed form, no action is required.

3.3.2 Minterm Separation Method from CSOP

Step 1: Store the coefficients of the CSOP in the truth vector $[C_M]$.

Step 2: Create partitioned matrix $R(x_j)$ from $[C_M]$ vector for each variable x_j as defined in definition 3.4.

- Step 3:* For each variable x_j , perform ExOR operation between the elements in the first and second rows of partitioned matrix R (x_j) matrix and store the result vector in $T(x_j)$.
- Step 4:* If the required polarity for the x_j variable is true, replace the content of each true variable x_j (even blocks) in the vector $[C_M]$ by the content of vector $T(x_j)$.
- Step 5:* If the required polarity for the x_j variable is complement form, replace the content of each complement variable x_j (odd block) in the truth vector $[C_M]$ by the content of vector $T(x_j)$.
- Step 6:* If the required polarity for the x_j variable is mixed form, no action is required.
- Step 7:* Repeat the previous steps for the rest of the variables by using the new vector from steps 2,3,4,5, or 6 depending on the polarity.
- Step 8:* The one element stored in the coefficient matrix $[C_M]$ represent the number of coefficients for the particular polarity for the MPRM expansion.
- Step 9:* To derive the equations for the designed circuit, re-arrange the coefficients of C_M matrix. The re-arrangement can be done by replacing all polarity digits with mixed form ($p_j=2$) by true form ($p_j=0$). Then perform ExOR operation of the extension for each coefficient stored in the resulting matrix $[C_M]$ with the polarity number calculated after doing replacement. This step is explained in details in Example 3.3. Figure 3.1 shows the pseudo code for this algorithm.

```

Procedure Minterm Separation ( )
{
  Generate the truth Vector CM for the given Boolean Function
  (as in Definition 3.1).
  Represent mixed polarity by using ternary numbers <pj>=<pn-1 pn-2 ...p0>
  For each variable xj
  {
    Generate partitioned matrix R(xj) (as in Definition 3.4)
    Generate T (xj) = Upper row of R (xj) ⊕ Lower row of R(xj)

    If ( pj = 0 (True polarity) )
      MPRM_TRUE( )

    If ( pj = 1 (Complement polarity) )
      MRRM_COMPLEMENT( )

    If (pj = 2 (Mixed polarity) )
      Leave the CM vector without any change
  } End for

  Re-arrange the coefficients of CM matrix, replace any 2 ( if exist) in the
  polarity number calculated by 0 and perform ExOR operation of the
  extension for each coefficient with the polarity number calculated.
} End Minterm Separation ( )

Procedure MPRM_TRUE( )
Replace the even blocks of CM matrix (true form of variable (xj)) by the T
(xj), odd blocks of CM remain unchanged.
End MPRM_TRUE( )

Procedure MPRM_COMPLMENT( )
Replace the odd blocks of CM matrix (complement form of variable (xj)) by
the T (xj), even blocks of CM remain unchanged
End MPRM_COMPLEMENT( )

```

Figure 3.1: Pseudo Code for the Minterm separation technique

Example 3.3: Convert 3-variables Boolean function from CSOP form to MPRM-Polarity $\langle 7 \rangle_{10} = \langle 021 \rangle_3$

$$f(x_2, x_1, x_0) = \sum m(7,6,2,0)$$

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

- 1) Store the coefficient of the product terms in the truth vector

$$C_M = [1010 \ 0011]$$

- 2) Separate C_M matrix around variable (x_2). The number of blocks for the variable (x_2) = $2^{(3-2)} = 2$ and the size of each block = $2^3/2 = 4$, this means that C_M is divided into two blocks and each block contains 4 elements. (note that the upper row of $R(x_2)$ contains the bold letters of C_M matrix)

$$C_M = [\overbrace{\mathbf{1010}}^{\text{Upper Row}} \quad \overbrace{0011}^{\text{Lower Row}}]$$

$$R(x_2) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \oplus$$

$$T(x_2) = \begin{array}{c} \text{-----} \\ (1 \ 0 \ 0 \ 1) \end{array}$$

- 3) Since the required polarity is '0' for variable (x_2), replace the true part of (x_2) in C_M vector by $T(x_2)$

$$C_M = [1010 \ \mathbf{0011}] \implies C_M = [1010 \ \mathbf{1001}]$$

- 4) Since the polarity is '2' for variable (x_1), leave the vector C_M without any change.

$$C_M = [1010 \ 1001]$$

- 5) Separate C_M matrix around variable (x_0). The number of blocks for the variable (x_0) = $2^{(3-0)} = 8$ and the size of each block = $2^3/8 = 1$, this means that C_M is divided into eight blocks and each block contains 1 element.

$$C_M = [\mathbf{1010} \ \mathbf{1001}]$$

$$R(x_0) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \oplus$$

$$T(x_0) = \begin{array}{c} \text{-----} \\ (1 \ 1 \ 1 \ 1) \end{array}$$

6) Since the polarity is '1' for variable (x_0), replace the complement part of (x_0) in C_M vector by T (z).

$$C_M = [\mathbf{1010 \ 1001}] \implies C_M = [\mathbf{1010 \ 1011}]$$

7) Re-arrange the coefficients of C_M . First, replace any 2 in the polarity number by 0 (i.e Polarity will be changed from (021) to (001)) and perform EXOR operation of all the coefficients with the (001) as follows:

$$\begin{aligned} b_{000} \oplus b_{001} &= b_{001} & , & \quad b_{001} \oplus b_{001} = b_{000} \\ b_{010} \oplus b_{001} &= b_{011} & , & \quad b_{011} \oplus b_{001} = b_{010} \\ b_{100} \oplus b_{001} &= b_{101} & , & \quad b_{101} \oplus b_{001} = b_{100} \\ b_{110} \oplus b_{001} &= b_{111} & , & \quad b_{111} \oplus b_{001} = b_{110} \end{aligned}$$

This means that the resulting coefficients of C_M are arranged as

$[b_1 \ b_0 \ b_3 \ b_2 \ b_5 \ b_4 \ b_7 \ b_6]$. These should be re arranged as follow:

$[b_0 \ b_1 \ b_2 \ b_3 \ b_4 \ b_5 \ b_6 \ b_7]$.

Therefore ;

$$C_M = [1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1] \implies C_M = [0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1]$$

$$\begin{aligned} C_{M(021)} &= [1 \ x_2] * [\bar{x}_1 \ x_1] * [1 \ \bar{x}_0] \\ &= [\bar{x}_1 \ \bar{x}_1\bar{x}_0 \ x_1 \ x_1\bar{x}_0 \ x_2\bar{x}_1 \ x_2\bar{x}_1\bar{x}_0 \ x_2x_1 \ x_2x_1\bar{x}_0] \end{aligned}$$

$$C_{M(021)} = [0 \ \mathbf{1} \ 0 \ \mathbf{1} \ 0 \ \mathbf{1} \ \mathbf{1} \ \mathbf{1}]$$

It is obvious from the resulting coefficient matrix $C_{M(021)}$ that the second, fourth, sixth, seventh and eighth product terms exist. Therefore;

$$f(x_2, x_1, x_0) = \bar{x}_1\bar{x}_0 \oplus x_1\bar{x}_0 \oplus x_2\bar{x}_1\bar{x}_0 \oplus x_2x_1 \oplus x_2x_1\bar{x}_0$$

To prove that the polarity derived represents the same function, Boolean algebra can be used as follows:

$$\begin{aligned} f(x_2, x_1, x_0) &= \bar{x}_2\bar{x}_1\bar{x}_0 + \bar{x}_2x_1\bar{x}_0 + x_2x_1\bar{x}_0 + x_2x_1x_0 \\ &= \bar{x}_2\bar{x}_0 + x_2x_1 \end{aligned}$$

$$\begin{aligned} f(x_2, x_1, x_0) &= \bar{x}_1\bar{x}_0 \oplus x_1\bar{x}_0 \oplus x_2\bar{x}_1\bar{x}_0 \oplus x_2x_1 \oplus x_2x_1\bar{x}_0 \\ &= \bar{x}_0(\bar{x}_1 \oplus x_1) \oplus x_2\bar{x}_1\bar{x}_0 \oplus x_2x_1 \oplus x_2\bar{x}_1\bar{x}_0 \oplus x_2\bar{x}_0 \\ &= \bar{x}_0 \oplus x_2x_1 \oplus x_2\bar{x}_0 \\ &= \bar{x}_2\bar{x}_0 \oplus x_2x_1 \\ &= (\bar{x}_2\bar{x}_0)(x_2x_1) + \overline{(\bar{x}_2\bar{x}_0)}(x_2x_1) \\ &= (\bar{x}_2\bar{x}_0)(\bar{x}_2 + x_1) + (x_2 + x_0)(x_2x_1) \\ &= \bar{x}_2\bar{x}_0 + \bar{x}_2\bar{x}_1\bar{x}_0 + x_2x_1 + x_2x_1x_0 \\ &= \bar{x}_2\bar{x}_0(\bar{x}_1 + 1) + x_2x_1(x_0 + 1) \\ &= \bar{x}_2\bar{x}_0 + x_2x_1, \end{aligned}$$

which represent the same function in CSOP form.

Example 3.4: Re-convert the previous example from MPRM-Polarity $\langle 7 \rangle_{10}$ to CSOP form.

- 1) Take the C_M vector before re-arrangement (or do step 7 of previous Example to get C_M before rearrangement) as shown below:

$$C_{M(021)} = \left[\begin{array}{cc} \text{Upper Row} & \text{Lower Row} \\ \underbrace{\quad\quad\quad} & \underbrace{\quad\quad\quad} \\ \mathbf{1010} & 1011 \end{array} \right]$$

- 2) Create $R(x_2)$ as follows:

$$R(x_2) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \oplus$$

$$\text{-----}$$

$$T(x_2) = (0 \ 0 \ 0 \ 1)$$

- 3) For the requirement of changing the polarity from true form to CSOP form for variable x_2 , replace the true part of x_2 in C_M vector by $T(x_2)$

$$C_M = [1010 \mathbf{1011}] \implies C_M = [1010 \mathbf{0001}]$$

- 4) For the requirement of changing the polarity from mixed form to CSOP form for variable x_1 , leave the vector C_M without any change.

$$C_M = [1010 \ 0001]$$

- 5) Create $R(x_0)$ as follows:

$$R(x_0) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \oplus$$

$$T(x_0) = (1 \ 1 \ 0 \ 1)$$

- 6) For the requirement of changing the polarity from complement form to CSOP form for variable x_0 , replace the complement part (odd blocks) of x_0 in C_M vector by $T(x_0)$.

$$C_M = [\mathbf{1010} \ \mathbf{0001}] \implies C_M = [\mathbf{1010} \ \mathbf{0011}]$$

- 7) The resulted C_M vector = $[1010 \ 0011]$ which is same vector related to the original function $f(x_2, x_1, x_0) = \sum m(7,6,2,0)$.

3.3.3 Minterm Separation method from FPRM

It is possible to use Minterm separation method to derive any polarity directly from other FPRM avoiding the time-consuming CSOP to FPRM conversion for each polarity. Time efficiency is achieved in this technique because the information utilised in finding mixed polarity expansion of one polarity is utilised by others.

- To change polarity from mixed form to true form or opposite, follow the procedures $MPRM_TRUE()$ as detailed in Fig. 3.1.

- To change polarity from mixed form to complement form, follow the procedures MPRM_TRUE(), then MPRM_COMPLEMENT(), as detailed in Fig. 3.1.
- To change polarity from complement form to mixed form, follow the procedures MPRM_COMPLEMENT(), then MPRM_TRUE(), as detailed in Fig. 3.1
- To change polarity from true form to complement form or opposite, follow the procedure MPRM_COMPLEMENT(), as detailed in Fig. 3.1

Example 3.5: Convert a 3-variable FPRM- Polarity $\langle 3 \rangle_{10} = \langle 010 \rangle_3$ to MPRM - Polarity $\langle 25 \rangle_{10} = \langle 221 \rangle_3$

$$f(x_2, \bar{x}_1, x_0) = \bar{x}_1 \oplus \bar{x}_1 x_0 \oplus x_2 \bar{x}_1$$

- 1) The truth table for the giving function is as shown in Table 3.1
- 2) Each product term corresponds to '1' in the coefficient matrix. Hence, $[C_M]$ can be represented by :

$$C_{M(010)} = [0011 \ 0010]$$

Table 3.1: FPRM Terms for Example 3.5

FPRM-Polarity 3		
x_2	x_1	x_0
0	1	0
0	1	1
1	1	0

- 3) Create $R(x_2)$ as follows:

$$R(x_2) = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \oplus$$

$$T(x_2) = (0 \ 0 \ 0 \ 1)$$

- 4) Changing the polarity from $p\langle 010 \rangle_3$ to $p\langle 221 \rangle_3$, can be done by changing the polarity for variable x_2 from true form to mixed form. This can be done by replacing the true part of x_2 in C_M vector by $T(x_2)$.

$$C_{M(010)} = [0011 \ 0010] \implies C_{M(210)} = [0011 \ 0001]$$

- 5) Changing the polarity from $p\langle 210 \rangle_3$ to $p\langle 221 \rangle_3$, can be done by changing the polarity for variable x_1 from complement form to mixed form. This can be done by 2 steps :

- Replacing the complement part of x_1 in C_M vector by $T(x_1)$.

Create $R(x_1)$ as follows:

$$R(x_1) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \oplus$$

$$T(x_1) = (1 \ 1 \ 0 \ 1)$$

$$C_{M(210)} = [0011 \ 0001] \implies C_{M(200)} = [1111 \ 0101]$$

- Replacing the true part of x_1 in C_M vector by $T(x_1)$.

Create $R(x_1)$ as follows:

$$R(x_1) = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \oplus$$

$$T(x_1) = (0 \ 0 \ 0 \ 0)$$

$$C_{M(200)} = [1111 \ 0101] \implies C_{M(220)} = [1100 \ 0100]$$

- 6) Changing the polarity from $p\langle 220 \rangle_3$ to $p\langle 221 \rangle_3$, can be done by changing the polarity for variable x_0 from true form to complement form. This can be done by creating $R(x_0)$ and replacing the complement part of x_0 in C_M vector by $T(x_0)$ as follows:

$$R(x_0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \oplus$$

$$T(x_0) = (0 \ 0 \ 1 \ 0)$$

$$C_{M(220)} = [1100 \ 0100] \implies C_{M(221)} = [01 \ 00 \ 11 \ 00]$$

7) The resulted $C_{M(221)}$ vector = [01 00 11 00] which has four coefficients for polarity $\langle 25 \rangle_{10}$.

$$C_{M(221)} = [\bar{x}_2 \ x_2] * [\bar{x}_1 \ x_1] * [1 \ \bar{x}_0]$$

$$= [\bar{x}_2\bar{x}_1 \ \bar{x}_2\bar{x}_1\bar{x}_0 \ \bar{x}_2x_1 \ \bar{x}_2x_1\bar{x}_0 \ x_2\bar{x}_1 \ x_2\bar{x}_1\bar{x}_0 \ x_2x_1 \ x_2x_1\bar{x}_0]$$

$$= [0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0]$$

$$p \langle 221 \rangle_3, f(x_2, x_1, x_0) = \bar{x}_2\bar{x}_1\bar{x}_0 \oplus x_2\bar{x}_1 \oplus x_2\bar{x}_1\bar{x}_0$$

To prove that all polarities derived represent the same function:

$$p \langle 010 \rangle_3 f(x_2, \bar{x}_1, x_0) = \bar{x}_1 \oplus \bar{x}_1x_0 \oplus x_2\bar{x}_1$$

$$\begin{aligned} p \langle 221 \rangle_3, f(x_2, x_1, \bar{x}_0) &= \bar{x}_2\bar{x}_1\bar{x}_0 \oplus x_2\bar{x}_1 \oplus x_2\bar{x}_1\bar{x}_0 \\ &= \bar{x}_1\bar{x}_0(x_2 \oplus \bar{x}_2) \oplus x_2\bar{x}_1 \\ &= \bar{x}_1(x_0 \oplus 1) \oplus x_2\bar{x}_1 \\ &= \bar{x}_1 \oplus \bar{x}_1x_0 \oplus x_2\bar{x}_1 \end{aligned}$$

3.4 Maxterm Separation Method

In this section, new fast method is presented which can be used to compute the coefficients of MPDRM expansions directly from truth vector of the standard CPOS Boolean function or from FPDRM form. Also it can be used to derive any mixed dual polarity from another. Further; these methods could be used to calculate the coefficients of standard CPOS Boolean function from any mixed dual polarity.

3.4.1 Demonstration for the Maxterm Separation technique

In this section, fast, efficient, and straightforward method has been invented based on separation of maxterms within CPOS expression to compute the d_i coefficients (equation (1.11)) of MPDRM expansion from

c_i coefficients (equation (1.4)) of CPOS expression, where $i = 0, 1, 2, \dots, (2^n - 1)$. Each MPDRM expansion has 3^n coefficients set of d_i , which is associated with sum terms of DRM expansion as given in equation (1.11). If all variables are present in every term in CPOS function as in (1.4), then the AND can be replaced by ExNOR giving equation (3.2). This expression represents MPDRM with all variables in mixed form. For n-variable functions (equation (3.2)), the Maxterm separation around the variable x_j is shown in equation (3.5) as partitioned matrix $R(x_j)$. Unlike Minterm separation, the odd blocks which include the terms where the variable x_j appears in true form only are allocated in the upper row, while the even blocks which include the terms where the variable x_j appears in complement form only are allocated in the lower row.

In order to change the variable from standard CPOS form to dual true form, the complement part has to be eliminated from equation (3.2). To eliminate the complement part for any variable \bar{x}_j , the following formulas can be applied:

$$\begin{aligned}\bar{x}_j &= x_j \odot 0 \\ (a + \bar{x}_j) \odot (b + x_j) &= [a + (0 + x_j)] \odot (b + x_j) \\ &= [(a + 0) \odot (a + x_j)] \odot (b + x_j) \\ &= a \odot [(a + x_j) \odot (b + x_j)]\end{aligned}$$

However

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

Therefore;

$$(a + \bar{x}_j) \odot (b + x_j) = a \odot [(a \odot b) + x_j] \quad (3.8)$$

As a result, it is clear that the coefficients associated with the complement part will stay unchanged while the coefficients associated with true part have to be ExNORed with the coefficients of the complement part.

This can be achieved by having the ExNOR operation between the upper and lower rows of the partitioned matrix $R(x_j)$, and the elimination of the complement part can be done by replacing the coefficients of the corresponding true part (odd blocks) of the $[C_M]$ by the ExNOR results.

In order to change the variable from standard CPOS form to complement form, the true part has to be eliminated from equation (3.2) as shown below:

$$\begin{aligned} (a + \bar{x}_j) \odot (b + x_j) &= (a + \bar{x}_j) \odot [b + (0 \odot \bar{x}_j)] \\ &= (a + \bar{x}_j) \odot [(b + 0) \odot (b + \bar{x}_j)] \\ &= [(a + \bar{x}_j) \odot (b + \bar{x}_j)] \odot b \end{aligned}$$

Taking the complement of the following equation:

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

Taking the complement of the last equation gives the following equation:

$$\overline{x_j(\bar{a} \oplus \bar{b})} = \bar{x}_j + (a \odot b)$$

Therefore;

$$(a + \bar{x}_j) \odot (b + x_j) = b \odot [(a \odot b) + \bar{x}_j] \quad (3.9)$$

According to equation (3.9), the coefficients associated with true part stays as it is while the coefficients associated with complement part is replaced by the coefficients of complement part ExNORed with the coefficients of the true part.

The variable x_j in the standard CPOS also represents the mixed form for the given variable. Therefore; in order to change the variable from standard CPOS form to mixed form, no action is required.

3.4.2 Maxterm Separation Method from CPOS

Step 1: Store the coefficients of the CPOS in the truth vector C_M .

Step 2: Create $R(x_j)$ matrix from C_M vector for each variable x_j as defined in Definition 3.5.

Step 3: For each variable x_j , perform ExNOR operation between the elements in the first and second rows of $R(x_j)$ matrix and store the result vector in $T(x_j)$.

Step 4: If the required polarity for the x_j variable is true, or if you want to change polarity for the given variable from true to CPOS form, then replace the content of each true variable x_j (odd blocks) in the truth vector C_M by the content of vector $T(x_j)$.

Step 5: If the required polarity for the x_j variable is complement form, or if you want to change polarity for the given variable from complement form to CPOS form, then swap between the complement part and the true part of the C_M matrix for the selected x_j , then replace the odd blocks of C_M matrix, by the $T(x_j)$.

Step 6: If the required polarity for the x_j variable is mixed form, no action is required.

Step 7: Repeat the previous steps for the rest of the variables by using the new vectors from steps 2,3,4,5, and 6 depending on the polarity.

Step 8: The logic zero elements stored in the coefficient matrix C_M represent the number of coefficients for the particular polarity for the MPDRM. Figure 3.2 shows the pseudo code for this algorithm.

```

Procedure Maxterm Separation( )
{
Generate the truth Vector CM for the given Boolean Function
(as in Definition 3.1).
Represent mixed polarity by using ternary numbers <pj>=<pn-1 pn-1 ...p0>

For each variable xj
{
Generate partitioned matrix R(xj) (as in Definition 3.5)
Generate T (xj) = Upper row of R (xj) ⊙ Lower row of R(xj)

If ( pj = 0 (True polarity) )
MPDRM_TRUE( )

If ( pj = 1 (Complement polarity) )
MRDRM_COMPLEMENT( )

If (pj = 2 (Mixed polarity) )
Leave the CM vector without any change
} End for
}
End Maxterm Separation ( )

Procedure MPDRM_TRUE( )
Replace the odd blocks of CM matrix (true form of variable (xj)) by the T
(xj), even blocks of CM remain unchanged.
End MPDRM_TRUE ( )

Procedure MPDRM_COMPLMENT( )
Swap between the complement part and the true part of the CM matrix for
the selected xj , then replace the odd blocks of CM matrix by the T (xj).
End MPDRM_COMPLMENT ( )

```

Figure 3.2: Pseudo Code for the Maxterm separation technique

Example 3.6: Convert a 3-variable function from CPOS form to MPDRM with polarity $\langle 7 \rangle_{10} = p\langle 021 \rangle_3$.

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

1) Store the coefficient of the maxterms in the truth vector C_M :

$$C_M = [1010 \ 0011]$$

- 2) Separate the C_M matrix around variable (x_2). The number of blocks for the variable (x_2) = $2^{3-2} = 2$ and the size of each block = $2^3/2 = 4$, this means that C_M is divided into two blocks and each block contains 4 elements.

$$C_M = \begin{array}{cc} \text{Upper Row} & \text{Lower Row} \\ \hline [1010] & [0011] \end{array}$$

$$R(x_2) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \odot$$

$$T(x_2) = (0 \ 1 \ 1 \ 0)$$

- 3) Since the polarity is '0' for variable x_2 , replace the true part of x_2 in C_M by $T(x_2)$

$$C_M = [1010 \ 0011] \implies C_M = [0110 \ 0011]$$

- 4) Since the polarity is '2' for variable x_1 , leave the Matrix C_M without any change.

$$C_M = [0110 \ 0011]$$

- 5) Separate C_M matrix around variable (x_0). The number of blocks for the variable (x_0) = $2^{(3-0)} = 8$ and the size of each block = $2^3/8 = 1$, this means that C_M is divided into eight blocks and each block contains 1 element.

$$C_M = [0110 \ 0011]$$

$$R(x_0) = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \odot$$

$$T(x_0) = (0 \ 0 \ 1 \ 1)$$

- 6) Since the polarity is '1' for variable x_0 , replace the content of each true part related to the variable by the content of the complement part for the same variable in the matrix C_M .

$$C_M = [0110 \ 0011] \implies C_M = [1001 \ 0011]$$

Then replace the true part of x_0 in C_M by $T(x_0)$

$$C_M = [1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1] \implies C_M = [0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1]$$

The resulted C_M indicates that the $p<021>_3$ has four sum terms, that's mean:

$$\begin{aligned} C_{M(021)} &= [x_2 \ 0] \star [x_1 \ \bar{x}_1] \star [\bar{x}_0 \ 0] \\ &= [x_2 + x_1 + \bar{x}_0 \quad x_2 + x_1 \quad x_2 + \bar{x}_1 + \bar{x}_0 \quad x_2 + \bar{x}_1 \\ &\quad x_1 + \bar{x}_0 \quad x_1 \quad x_1 + \bar{x}_0 \quad \bar{x}_1] \end{aligned}$$

$$C_{M(021)} = [0001 \ 1011]$$

The matrix $C_{M(021)}$ shows that the first, second, third and sixth sum terms exist. Therefore;

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

To prove that the polarity derived represents the same function using Boolean algebra:

In CPOS:

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

this represents the same function in MPRM-Polarity $<021>_3$ form.

3.4.3 Maxterm Separation method from FPDRM

It is also possible to use Maxterm separation method to derive any dual polarity directly from other FPDRM avoiding the time-consuming CPOS to FPDRM conversion for each polarity.

- To change polarity from dual mixed form to dual true form or opposite, follow the procedure MPDRM_TRUE() as detailed in Fig. 3.2.
- To change polarity from dual true form to dual complement form, follow the procedures MPDRM_TRUE(), then MPDRM_COMPLEMENT(), as detailed in Fig. 3.2.
- To change polarity from dual complement form to dual true form, follow the procedures MPDRM_COMPLEMENT (), then MPDRM_TRUE(), as detailed in Fig. 3.2.
- To change polarity from dual mixed form to dual complement form or opposite, follow this procedure:

MPDRM_MIX_COMP():

Swap between the complement part and the true part of the C_M matrix for the selected x_j , then replace the odd blocks of C_M matrix (complement form of variable (x_j)), by the $T(x_j)$

End MPDRM_MIX_COMP

Example 3.7: Convert a 3-variable FPDRM- Polarity $\langle 3 \rangle_{10} = \langle 010 \rangle_3$ to MPDRM - Polarity $\langle 25 \rangle_{10} = \langle 221 \rangle_3$

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

The truth table for the giving function is as shown in Table 3.2

Table 3.2: FPDRM Terms for Example 3.7

FPDRM-Polarity 3		
x_2	x_1	x_0
1	0	1
1	0	0
0	0	1

1) Each sum term corresponds to '0' in the coefficient matrix. Hence, $[C_M]$ can be represented by

$$C_{M(010)} = [10 \ 11 \ 00 \ 11]$$

2) Changing the polarity from $p\langle 010 \rangle_3$ to $p\langle 221 \rangle_3$, can be done by changing the polarity for variable x_2 from dual true form to dual mixed form for. This can be done by creating $R(x_2)$ and then replacing the true part (odd blocks) of x_2 in C_M vector by $T(x_2)$ as shown below:

$$R(x_2) = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \odot$$

$$T(x_2) = (0 \ 1 \ 1 \ 1)$$

$$C_{M(010)} = [1011 \ 0011] \implies C_{M(210)} = [0111 \ 0011]$$

3) Changing the polarity from $p\langle 210 \rangle_3$ to $p\langle 221 \rangle_3$, can be done by changing the polarity for variable x_1 from dual complement form to dual mixed form. This can be done by creating $R(x_1)$ and then swap between the complement part and the true part of the C_M matrix for the selected variable, and finally replace the even blocks of C_M matrix (complement form of variable (x_1)), by the $T(x_1)$ as shown below:

$$R(x_1) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \odot$$

$$T(x_1) = (0 \ 1 \ 0 \ 0)$$

$$C_{M(210)} = [0111 \ 0011] \xrightarrow{\text{Swap}} [1101 \ 1100]$$

$$C_{M(210)} = [1101 \ 1100] \implies C_{M(220)} = [1101 \ 1100]$$

4) Changing the polarity from $p\langle 220 \rangle_3$ to $p\langle 221 \rangle_3$, can be done by changing the polarity for variable x_0 from dual true form to dual complement form. This can be done by 2 steps :

- Replacing the true part of x_0 in C_M vector by $T(x_0)$.

$$R(x_0) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix} \odot$$

$$\text{-----}$$

$$T(x_0) = (1 \ 0 \ 1 \ 1)$$

$$C_{M(220)} = [1101 \ 1100] \implies [1101 \ 1110]$$

- Swap between the true part and complement part and replace the true part (odd blocks) of x_0 in C_M vector by $T(x_0)$.

$$R(x_0) = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \odot$$

$$\text{-----}$$

$$T(x_0) = (1 \ 0 \ 1 \ 0)$$

$$C_{M(220)} = [1101 \ 1110] \xrightarrow{\text{Swap}} [1110 \ 1101]$$

$$C_{M(220)} = [1110 \ 1101] \implies C_{M(221)} = [1100 \ 1101]$$

5) The resulted $C_{M(221)}$ vector = [11 00 11 01] which has three coefficients for polarity $\langle 25 \rangle_{10}$.

$$C_{M(221)} = [x_2 \ \bar{x}_2] \star [x_1 \ \bar{x}_1] \star [\bar{x}_0 \ 0]$$

$$= [x_2 + x_1 + \bar{x}_0 \quad x_2 + x_1 \quad x_2 + \bar{x}_1 + \bar{x}_0 \quad x_2 + \bar{x}_1$$

$$\quad \bar{x}_2 + x_1 + \bar{x}_0 \quad \bar{x}_2 + x_1 \quad \bar{x}_2 + \bar{x}_1 + \bar{x}_0 \quad \bar{x}_2 + \bar{x}_1]$$

$$C_{M(221)} = [1100 \ 1101]$$

It is obvious from the resulting coefficient matrix $C_{M(221)}$ that coefficients (2,3,& 6) are exist. Therefore;

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

To prove that all polarities derived represent the same function:

$$p < 010 >_3, f(x_2, \bar{x}_1, x_0) = \bar{x}_1 \odot (\bar{x}_1 + x_0) \odot (x_2 + \bar{x}_1)$$

$$\begin{aligned} p < 221 >_3, f(x_2, x_1, \bar{x}_0) &= (x_2 + \bar{x}_1 + \bar{x}_0) \odot (x_2 + \bar{x}_1) \odot (\bar{x}_2 + \bar{x}_1 + \bar{x}_0) \\ &= (\bar{x}_0 + \bar{x}_1 + (\bar{x}_2 \odot x_2)) \odot (x_2 + \bar{x}_1) \\ &= (\bar{x}_1 + (x_0 \odot 0)) \odot (x_2 + \bar{x}_1) \\ &= \bar{x}_1 \odot (\bar{x}_1 + x_0) \odot (x_2 + \bar{x}_1) \end{aligned}$$

3.5 Experimental Results

Minterm/Maxterm separation techniques are implemented and applied to several MCNC benchmark functions [70 -72]. The results are given in Tables 3.3 & 3.4. These algorithms are tested on a personal computer with Intel CPU running at 2.4 GHz and 2 GB RAM under Window XP, professional. They are implemented using C++ and compiled using Bloodshed Dev C++. An *I/O* denotes the number of inputs/outputs respectively of benchmark name. The number of terms for the optimum mixed/dual mixed polarity doing exhaustive search are given in *Terms for optimum MPRM/MPDRM* respectively.

The CPU time required to do exhaustive search using Minterm/Maxterm separation methods are given in *CPU Time*.

From the results, it is obvious that the same optimum polarities are obtained as in Tables 2.16 & 2.17 and that's because both of Minterm/Maxterm separation techniques and Extended_Tabular algorithms are doing exhaustive search. Experimental results show that the difference is in the time consumed to find these results. It was found that the time required using Minterm/Maxterm methods are much less than the time

required using Extended_Tabular techniques especially for the large functions. That's because Extended_tabular techniques convert the Boolean functions to MPRM starting from PPRM. Hence, the Extended_Tabular techniques implemented in Chapter 2 convert the function from Boolean domain to PPRM using Tabular technique [31] and then convert the function from PPRM domain to MPRM domain using the proposed techniques explained in Chapter 2. While the Minterm/Maxterm, separation techniques convert the Boolean functions to MPRM starting directly from CSOP form, thus saving the time required to change the function from Boolean domain to PPRM. The other thing, time required to produce results using Extended tabular techniques increase with number of terms, while Minterm/Maxterm separation techniques do not depend on number of terms, because these techniques start from the truth vector of Boolean functions.

Table 3.3: Benchmark results of Minterm separation technique
(doing exhaustive search)

<i>Name</i>	<i>I/O</i>	PPRM Terms	Terms for Optimum MPRM	CPU Time
Dc1	4/7	16	10	0.015 sec.
Xor5	5/1	5	5	0.031 sec.
bw	5/28	32	22	0.328 sec.
Squar5	5/8	23	23	0.016 sec.
Con1	7/2	19	14	0.031 sec.
Inc	7/9	91	34	0.062 sec.
newill	8/1	57	13	0.062 sec
Misex1	8/7	60	13	0.219 sec
Sqrt8	8/4	28	26	0.141 sec
Rd84	8/4	107	107	0.156 sec.
clip	9/5	217	182	0.98 sec.
risc	8/31	89	30	0.75 sec.
Apex4	9/19	445	444	3.625 sec.
Sym10	10/1	266	266	1.922 sec.
Sao2	10/4	1022	76	4.078 sec.
Ex1010	10/10	1023	810	12.716 sec.
Dk17	10/11	996	30	9.25 sec.
Table3	14/14	5504	401	4 h. & 14 min.
Alu4	14/8	4406	2438	2 h. & 35 min.
Misex3	14/14	6028	1421	4 h. & 20 min.
T481	16/1	41	13	15 h. & 12 min.
b12	15/9	209	64	15 h. & 32 min.
Table5	15/17	74500	551	5 days & 4 h.

Table 3.4: Benchmark results of Maxterm separation technique
 (doing exhaustive search)

<i>Name</i>	<i>I/O</i>	PPDRM Terms	Terms for Optimum MPDRM	CPU Time
Dc1	4/7	3	2	0.031 sec.
root	5/8	136	83	0.188 sec.
Con1	7/2	24	14	0.047 sec.
Inc	7/9	56	34	0.078 sec.
Misex1	8/7	5	5	0.203 sec.
Sqrt8	8/4	11	11	0.14 sec.
Rd84	8/4	256	108	0.156 sec.
Risc	8/31	21	12	0.75 sec.
clip	9/5	305	174	1.016 sec.
Apex4	9/19	512	407	3.656 sec.
Ex1010	10/10	1023	825	12.453 sec.
Dk17	10/11	8	8	9.28 sec.
Table3	14/14	4553	421	6 h.
Alu4	14/8	1091	496	3 h. & 12 min.
Misex3	14/14	6442	803	5 h. & 32 min.
b12	15/9	10	10	20 h. & 25 min.
Table5	15/17	4421	554	6 days & 8 h.

3.6 Summary

In this Chapter, new fast and straightforward techniques and algorithms are presented which can be used to compute the coefficients of MPRM/MPDRM expansions directly from truth vector of the standard CSOP/CPOS Boolean functions respectively. Also they can be used to

derive any mixed polarity from another MPRM/MPDRM expression without any constraints on the number of variables for the given function. These new techniques [26] are called Minterm/Maxterm separation techniques and implemented using C++ language and fully tested using standard benchmark examples. They also can be used to convert any mixed polarity to standard CSOP/CPOS Boolean function.

Furthermore; they can also be used to generate MPRM/MPDRM expansions from FPRM/FPDRM expansions, respectively. The proposed techniques are based on separating the truth vector of CSOP/CPOS. They are also based on methods proposed in [16, 20] which were used to convert CSOP/CPOS expressions to FPRM/FPDRM expansions respectively. These new techniques can be used manually or programmed on computers.

Chapter Four

Minimisation Techniques of RM/DRM expansions

4.1 Introduction

RM expressions have several advantages for functions that don't produce efficient solutions using CSOP, or CPOS techniques. Logic functions which can't minimise well in SOP form can often be implemented in the RM domain with fewer product terms which leads to reduced size and power consumption, and improved testability [6-8]. Additionally, circuits with ExOR/ExNOR gates are more amenable to efficient testing strategies [9]. The rest of this Chapter is organised as follows: New minimisation technique and algorithm for RM expansion for single and multi-output Boolean functions is given in section 4.2. In section 4.3, new minimisation technique and algorithm for DRM for single and multi-output Boolean function are given. All algorithms are implemented in C++ and fully tested using standard benchmark examples.

4.2 Technique for Minimisation of RM Expansions

In this section, new technique and algorithm is presented to generate minimal RM expansions from PPRM for any number of variables for completely specified single and multi-output Boolean functions. This technique is based on Tabular technique [31], which was developed to generate FPRM expression from CSOP Boolean function and described in details in Chapter two. The aim of this technique is to minimise the number

of terms. This is achieved by deriving the polarity for each term instead of each variable as in Tabular technique.

The main advantage of this algorithm is providing an optimal expansion with a minimal number of terms among ESOP expressions. The ESOP class [10] as in equation (1.27) is the most general class of AND-ExOR expressions and has 3^m different expressions where t/n is the number of products/ number of inputs respectively. In ESOP class of RM expansions, each term may has its own polarity.

Other advantage of this technique is to avoid the time-consuming exhaustive search for large functions, although it can't be guaranteed that all circuits can be minimised well using this technique.

This technique can be summarised by; initially creating two dimensional ($m \times n$) table which is called Polarity table to store the polarity for each variable within all terms, where m is the number of minterm for the given function. After that, starting from PPRM, the algorithm will change the polarity of a variable if and only if it results in a term which is identical to another term. The algorithm will generate new term and cancel the identical pair and set/reset the bit related to the variable of the specified term in the Polarity table to indicate that the polarity of this variable is changed from true to complement or vice versa. The algorithm will test all the variables in all the terms.

It is obvious that each term will have a different polarity. In other words the polarity of the term will be changed if this change will be useful to cancel another term. This technique is best illustrated by means of an example.

Consider Example 4.1.

Example 4.1 Given 3_variable PPRM Function as follows:

$$f(x_2, x_1, x_0) = \oplus \sum \pi(7,5,4,3,1,0) = x_2 x_1 x_0 \oplus x_2 x_0 \oplus x_2 \oplus x_1 x_0 \oplus x_0 \oplus 1$$

1. List all terms in a binary form and create two dimension (3×6) Polarity Table which will store the polarity for each variable within each term. Initially, reset Polarity table

Table 4.1: Truth table for Example 4.1

PPRM terms			Polarity Table		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	0	0	0	0
0	0	1	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	1	0	0	0

2. For variable x_0 within all terms, generate new term if and only if changing the polarity of this variable from true form to complement form can generate identical pair. Then cancel the identical pair and toggle (set or reset) the bit that denotes to the variable of the specified term within the Polarity table to indicate that the polarity of this variable is changed from true to complement or vice versa. The terms generated by variable x_0 are shown in Table 4.2.
3. For variable x_1 , no terms are generated by this variable. This is because the new terms will not be identical to any other term.
4. For variable x_2 , the terms generated by variable x_2 are shown in Table 4.3 and the final resulting terms after minimisation are shown in Table 4.4.

Table 4.2: Terms generated by variable x_0 for Example 4.1

PPRM terms			Terms generated by x_0			Polarity table		
x_2	x_1	x_0	x_2	x_1	x_0	x_2	x_1	x_0
0	0	0				0	0	0
0	0	1	0	0	0	0	0	1
0	1	1				0	0	0
1	0	0				0	0	0
1	0	1	1	0	0	0	0	1
1	1	1				0	0	0

Table 4.3: Terms generated by variable x_2 for Example 4.1

Truth table			Terms generated by x_2			Polarity table		
x_2	x_1	x_0	x_2	x_1	x_0	x_2	x_1	x_0
0	0	1				0	0	1
0	1	1				0	0	0
1	0	1	0	0	1	1	0	1
1	1	1	0	1	1	1	0	0

Table 4.4: Resulted terms for Example 4.1

Final Uncancelled Terms			Polarity table		
x_2	x_1	x_0	x_2	x_1	x_0
1	0	1	1	0	1
1	1	1	1	0	0

$$f(x_2, x_1, x_0) = \bar{x}_2 x_1 x_0 \oplus \bar{x}_2 \bar{x}_0$$

Note

- In order to cancel any identical terms in the truth table; they should have the same polarity in the Polarity table.
- Changing logic 1 to logic 0 for any variable means changing the polarity for this variable within the selected term from true form to complement form.
- 1/0 in the Polarity table means that the polarity of the variable is complement/true respectively.

To prove that the resulted expression represent the same function

$$p < 0 >, f(x_2, x_1, x_0) = x_2x_1x_0 \oplus x_2x_0 \oplus x_2 \oplus x_1x_0 \oplus x_0 \oplus 1$$

After minimisation,

$$\begin{aligned} f(x_2, x_1, x_0) &= \bar{x}_2x_1x_0 \oplus \bar{x}_2\bar{x}_0 \\ &= \bar{x}_2x_1x_0 \oplus \bar{x}_2x_0 \oplus \bar{x}_2 \\ &= \bar{x}_2x_0 (x_1 \oplus 1) \oplus (\bar{x}_2 \oplus 1) \\ &= x_2x_1x_0 \oplus x_2x_0 \oplus x_2 \oplus x_1x_0 \oplus x_0 \oplus 1 \end{aligned}$$

It is clear in this technique; different polarity for the each variable within each term was derived depending on the usefulness for this action to the minimisation process for the circuit. The algorithm described in the following section may be employed to derive an expansion with minimal number of terms among 3^{th} ESOP different expansions.

4.2.1 The proposed algorithm for single output Boolean functions

The pseudo code for the proposed algorithm which is called RM_Minimisation () is shown in Figure 4.1, and can be summarised as follows:

1. List all the product terms in binary

2. Generate Polarity table to indicate the polarity of each variable for all terms.
3. Select a variable x_j , for every term containing a one in position j , test if the generation of new term with a zero in position j , \bar{x}_j will result in a duplicate term, delete duplicate pairs and toggle position j of the Polarity table to indicate the changed polarity of the variable for this product from true to complement or vice versa.
4. Repeat step 3 for all other products and for all variables.
5. The resulting uncanceled products will be the product terms of the resulted RM expressions. Further; (1/0) in the Polarity table indicates the polarity (complement/true) of each variable in each term.

```
Procedure RM_Minimisation( )
{
    Read file and store the minterm/or maxterms in array
    Convert the function from Boolean domain into FPRM/or FPDRM

    Loop for (j=0 to j=number of variables-1)
    {
        Loop for (i= 0 to i= 2n-1)
        {
            Flag = check_bit (j,i) // Flag = 1 if changing the jth bit of a
            // term generates duplicate term.

            If (Flag = 1)
            {
                Delete_duplicated_term (j,i)
                Toggle_Polarity_table (j,i)
            }
        } end Loop i
    } end Loop j

    Output Results ( )
} End RM_Minimisation
```

Figure 4.1: Pseudo Code for RM_Minimisation technique

4.2.2 The proposed algorithm for multi output Boolean functions

The algorithm in section 4.2.1 can be extended to multi-outputs by adding an array to store the outputs. In order to decide whether the generation of new terms is useful for the minimisation process or not, step 3 in section 4.2.1 will be replaced by the following:

- Generate a new term if and only if {number of one's in the output for the generated term < number of one's in the output for the original term}

Example 4.2: Consider multi-output PPRM functions shown in Table 4.5.

Table 4.5: PPRM terms for Example 4.2

Inputs		Outputs		
x_1	x_0	f_3	f_2	f_1
0	0	1	1	0
1	0	1	1	0
1	1	0	1	1

$$f_1 = x_1 x_0, f_2 = 1 \oplus x_1 \oplus x_1 x_0, f_3 = 1 \oplus x_1$$

For optimisation of multi-output CSOP Boolean function, the proposed algorithm starts by generating a new term for each variable. Therefore; the generated terms will have the same outputs as the original terms. For each generated term, the algorithm checks whether this action is useful for minimisation or not. These steps for minimisation of this example are as follows:

1. Find the terms generated by variable x_1 as shown in Table 4.6.

2. Two terms can be generated by changing the polarity of variable x_1 from true form to complement form. These terms are as follows :

- Term (0 1), this generation is not useful because it doesn't result in duplicate term.
- Term (0 0), this generation is useful because it results in identical terms, Therefore; term (0 0) can be cancelled because the result of the $(110 \oplus 110=000)$ which means that this term doesn't exist. The resulting terms are shown in Table 4.7.

Table 4.6: Terms generated by variable x_1 for Example 4.2

PPRM terms					Terms generated by x_1					Polarity table	
Inputs		Outputs			New Term		Outputs				
x_1	x_0	f_3	f_2	f_1	x_1	x_0	f_3	f_2	f_1	x_1	x_0
0	0	1	1	0						0	0
1	0	1	1	0	0	0	1	1	0	1	0
1	1	0	1	1						0	0

Table 4.7: Resulted Terms for Example 4.2

Inputs		Outputs			Polarity table	
x_1	x_0	f_3	f_2	f_1	x_1	x_0
1	0	1	1	0	1	0
1	1	0	1	1	0	0

3. No terms are generated by variable x_0 . This is because the new terms will not be able to cancel identical pair and the reason for that is the identical pair doesn't have identical polarity as in Polarity table.

Therefore; the cancellation will not be acceptable in this occasion. Table 4.7 shows the resulting terms.

4. The final equations after minimisation are as shown below:

$$f_1 = x_1 x_0, f_2 = \bar{x}_1 \oplus x_1 x_0, f_3 = \bar{x}_1$$

To prove that, it is clear that the resulting equations represent the same functions before minimisation as explained below:

f₁ is same before and after minimisation

$$\begin{aligned} f_2 &= \bar{x}_1 \oplus x_1 x_0 \\ &= 1 \oplus x_1 \oplus x_1 x_0, \end{aligned}$$

$$\begin{aligned} f_3 &= \bar{x}_1 \\ &= 1 \oplus x_1 \end{aligned}$$

4.2.3 Experimental Results

This algorithm was applied to several MCNC benchmark functions [70-72]. The results are given in column six of Table 4.8. *Terms for Optimum MPRM among 3ⁿ expressions* denotes the optimum MPRM terms doing exhaustive search using one of the new techniques described in Chapters 2 & 3. The minimal number of terms results using this algorithm are given in *minimal Terms*. The algorithm is tested on a personal computer with Intel CPU running at 2.4 GHz and 2 GB RAM under Window XP, professional. It is implemented using C++. The CPU times for all the examples were less than one second. The experimental results obtained within very short time compared with the size of the tested circuits reflecting the efficiency of the algorithm.

The RM_Minimisation technique as in Fig. 4.1 produced good results in very short time searching a field of 3th ESOP expressions.

Table 4.8: Benchmark results for RM_Minimisation technique

Benchmark Number as in Fig. (4.2)	Name	I/O	CSOP/PPRM Terms	Terms for Optimum MPRM	Minimal Terms	Saving %
1	Dc1	4/7	10/16	10	16	-37.5
2	Xor5	5/1	16/5	5	5	0
3	Bw	5/28	22/32	22	31	-29.3
4	Squar5	5/8	29/23	23	23	0
5	Con1	7/2	118/19	14	11	21.48
6	Inc	7/9	104/91	34	69	-50.7
7	newill	8/1	142/57	13	12	7.6
8	Misex1	8/7	128/60	13	37	-64.8
9	Sqrt8	8/4	255/128	26	47	-44.6
10	Rd84	8/4	256/107	107	107	0
11	9sym	9/1	420/210	173	154	10.98
12	Risc	8/31	256/89	30	85	-64.7
13	clip	9/5	496/217	182	155	14.8
14	Apex4	9/19	512/445	444	444	0
15	Sym10	10/1	837/266	266	255	4.1
16	Sao2	10/4	511/1022	76	65	14.4
17	Ex1010	10/10	1024/1023	810	1007	19.5
18	Table3	14/14	3176/5504	401	1067	-62.4
19	Alu4	14/8	16384/4406	2438	1790	26.5
20	Misex3	14/14	12281/6028	1421	1233	13.23
21	T481	16/1	42016/41	13	13	0
22	B12	15/9	32768/209	64	54	15.6
23	Table5	15/17	24572/74500	551	2956	-81.3

Table 4.8 shows that for 12 out of 18 tested benchmark circuit, the results produced are better or same as results doing exhaustive search among 3^n MPRM expressions consuming less than one second to provide these results. Saving in terms is calculated using equation (4.1) is shown in Fig. 4.2.

The proposed technique provides expression with minimal terms among ESOP expressions in which each term has its own polarity, while in the exhaustive search of chapters 2 & 3 , the proposed techniques provide optimum polarity among MPRM/or MPDRM with minimum terms. Moreover; the search space for the proposed technique is 3^{2n} which is larger than the search space 3^n for the exhaustive search for MPRM/MPDRM.

Therefore; there is a chance to find solutions with less terms than optimum terms in previous chapters and this is the reason why in some cases the results produced using this technique are better than optimum MPRM as shown in Table 4.8.

Furthermore; note that zero saving means that using RM_Minimising technique, the result is as good as optimum terms among 3^n expression consuming less or equal to one second for all cases. While positive saving means that the result for the proposed technique is better than optimum polarity among 3^n polarities. Negative saving means the results is worse than the optimum MPRM but that doesn't means worse than CSOP terms.

$$Saving = \frac{Absolute(Optimum\ MPRM - Minimal\ terms)}{Largest\ of(Optimum\ MPRM, Minimal\ terms)} \times 100\% \quad (4.1)$$

Largest of(Optimum MPRM, Minimal terms): choose either Optimum MPRM or Minimal terms depending on which one is bigger.

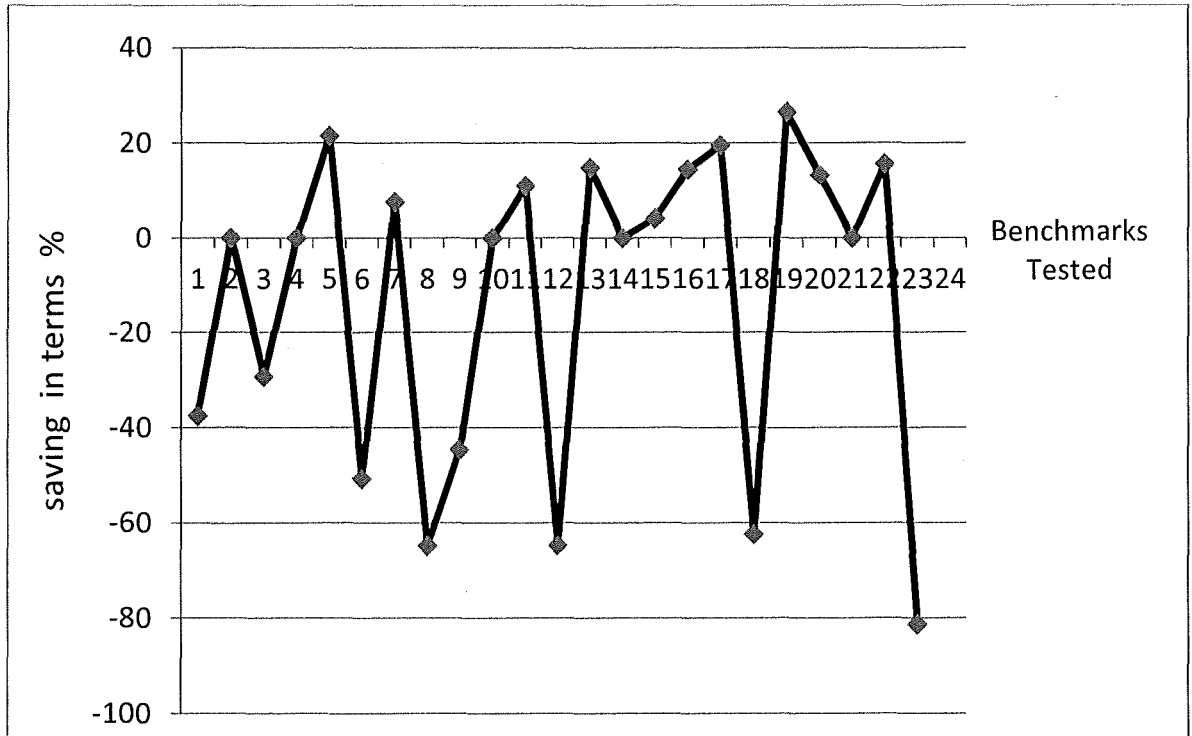


Figure 4.2: Chart for saving of terms for all tested Benchmarks Shown in Table 4.8

4.3 Technique for Minimisation of DRM Expansions

Any n-variable Boolean function can be represented using either CSOP or CPOS standard form of Boolean functions. Boolean expansions can also be represented using either RM or DRM form. DRM is introduced by [75] and based on using of ExNOR/OR gates instead of AND/OR gates in CPOS form. To achieve better minimisation, new technique and algorithm for DRM form as well as RM form in previous section are investigated. It is known that in some cases, the circuits can be better simplified in dual forms of RM expansions, whereas for other circuits the reverse will be the case.

In this section, an efficient technique based on Tabular technique is developed to generate minimal DRM expansion starting from PPDRM for any number of variables of Boolean functions. The aim here is to minimize

the number of sum terms. This is achieved by deriving the polarity for each term instead of each variable as in Tabular technique. This algorithm will find the polarity with minimal number of terms among ENPOS expressions. The ENPOS class is the most general class of OR-ExNOR expressions and has $3^{t/n}$ different expressions where t/n is the number of sums/number of inputs respectively. In this class of DRM expansions, each term may have its own polarity. This technique can be illustrated by means of an example. Consider this example below

Example 4.3: Given 3-variable PPDRM function as follows:

$$\begin{aligned}
 f(x_2, x_1, x_0) &= \odot \prod Q(6,4,2,0) \\
 &= x_0 \odot (x_1+x_0) \odot (x_2+x_0) \odot (x_2+x_1+x_0)
 \end{aligned}$$

1. List all sum terms in a binary form and create two dimensions (3×4) Polarity table which will store the polarity for each variable within each sum term. Initially, reset the Polarity table as shown in Table 4.9.

Table 4.9: Truth table for Example 4.3

PPDRM terms			Polarity table		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0
1	1	0	0	0	0

2. For variable x_0 within all terms, generate new term if and only if changing the polarity of this variable from true form to complement form can generate identical pair. Then cancel the identical pair and

toggle (set or reset) the bit that denotes the variable of the specified term in the Polarity table to indicate that the polarity of this variable is changed from true to complement or vice versa. No terms are generated by the variable x_0 . This is because the new terms are not identical to any other terms.

- The terms generated by the variable x_1 are shown in Table 4.10.

Table 4.10: Terms generated by variable x_1 for Example 4.3

PPDRM terms			Terms generated by x_0			Polarity table		
x_2	x_1	x_0	x_2	x_1	x_0	x_2	x_1	x_0
0	0	0	0	1	0	0	1	0
0	1	0				0	0	0
1	0	0	1	1	0	0	1	0
1	1	0				0	0	0

- The terms generated by variable x_2 are shown in Table 4.11 and the final resulting terms after minimisation are shown in Table 4.12.

Table 4.11: Terms generated by variable x_2 for Example 4.3

Truth table			Terms generated by x_2			Polarity table		
x_2	x_1	x_0	x_2	x_1	x_0	x_2	x_1	x_0
0	0	0	1	0	0	1	1	0
1	0	0				0	1	0

Table 4.12: Resulting terms for Example 4.3

Final Uncancelled Terms			Polarity table		
x_2	x_1	x_0	x_2	x_1	x_0
0	0	0	1	1	0

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

Note:

- In order to cancel any identical terms in the truth table; they should have the same polarity in the Polarity table.
- Unlike RM, in DRM, changing logic 1 to logic 0 for any variable means changing the polarity for this variable within the determined term from true form to complement form.
- 1/0 in the Polarity table means that the polarity of the variable is complement/true respectively.

To prove that the resulting expression represents the same function

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

After minimisation,

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

$$\begin{aligned} p < 0 >, f(x_2, x_1, x_0) &= x_0 \odot (x_1 + x_0) \odot (x_2 + x_0) \odot (x_2 + x_1 + x_0) \\ &= (\bar{x}_1 + x_0) \odot (x_2 + \bar{x}_1 + x_0) \\ &= (\bar{x}_2 + \bar{x}_1 + x_0) \end{aligned}$$

4.3.1 The proposed algorithm for single output Boolean functions

1. List all the sum terms for PPDRM in binary

2. Generate Polarity table to indicate the polarity of each variable for all sums.
3. For PPDRM, select a variable x_j , for every term containing a zero in position j test if the generation of new term with a one in position j , (\bar{x}_j) will result in a duplicate term (check_bit (j,i) in Fig. 4.1). Delete duplicate pairs (Delete_duplicated_term (j,i) in Fig. 4.1) and set/reset position j of the polarity table to indicate the changed polarity of the variable for this sum term from true to complement or vice versa (Toggle_Polarity_table (j,i) in Fig. 4.1).
4. Repeat step 3 for all other sums and for all variables.
5. The resulting uncanceled sums will be the sum terms of the MPDRM. Further; (1/0) in the Polarity table indicates to the polarity (complement/true) of each variable in each term. The pseudo code for this algorithm is detailed in Fig. 4.1.

4.3.2 The proposed algorithm for multi-output Boolean functions

The algorithm in section 4.3.1 can be extended to multi-outputs by adding an array to store the outputs. In order to decide whether the generation of new terms is useful for the minimisation process or not, step 3 in section 4.3.1 will be replaced by the following:

- Generate a new term if and only if {number of zero's in the resulted output < number of zero's of the output for the original term } to know whether the generation of the new term is useful for the minimization or not.

Example 4.4: Consider multi-output PPDRM functions shown in Table 4.13.

Table 4.13: PPDRM terms for Example 4.4

Inputs		Outputs		
x_1	x_0	f_3	f_2	f_1
0	0	1	0	1
0	1	1	1	0
1	0	0	1	1
1	1	1	1	0

$$f_1 = x_1 \odot 0, \quad f_2 = (x_0 + x_1), \quad f_3 = x_0$$

For optimisation of multi-output CPOS Boolean functions, The proposed algorithm starts by generating a new term for each variable. The generated terms will have the same outputs as the original terms. For each generated term, the algorithm checks whether this action is useful for minimisation or not. These steps for minimisation of this example are as follows:

1. The terms generated by variable x_1 are as shown in Table 4.14.

Table 4.14: Terms generated by variable x_1 for Example 4.4

					Terms generated by x_1					Polarity table	
Inputs		Outputs			New Term		Outputs				
x_1	x_0	f_3	f_2	f_1	x_1	x_0	f_3	f_2	f_1	x_1	x_0
0	0	1	0	1						0	0
0	1	1	1	0	$\overline{1}$	$\overline{1}$	1	1	0	1	0
1	0	0	1	1						0	0
$\overline{1}$	$\overline{1}$	1	1	0						0	0

2. Two terms can be generated by the variable x_1 to change its polarity from true form to complement form. These terms are as follows :

- Term (1 0), with output 101, but $(101 \odot 011 = 001)$, Therefore; the term generated exists for two outputs, while the initial one exists for one outputs, That means this generation is not useful for the minimisation.
- Term (1 1) with output 110, but $(110 \odot 110 = 111)$,) which means that this term doesn't exist for all outputs. Cancellation of the identical pair can be done. That means this generation is very useful for the minimisation.

Table 4.15: Resulted Terms for Example 4.4

Inputs		Outputs			Polarity table	
x_1	x_0	f_3	f_2	f_1	x_1	x_0
0	0	1	0	1	0	0
0	1	1	1	0	1	0
1	0	0	1	1	0	0

3. Two terms can be generated by the variable x_0 to change its polarity from true form to complement form. These terms are as follows :
 - Term (0 1), with output 101, but $(101 \odot 110 = 100)$, Therefore; the term generated exists for two outputs, while the initial one exists for one outputs, That means this generation is not useful for the minimisation.
 - Term (1 1) which is not identical to any terms, That means this generation is not useful for the minimisation. Therefore No term generated by changing the polarity of variable x_0 . The resulting terms after minimisation are shown in Table 4.15.
4. The result for this optimisation has 3 terms.

$$f_1 = \bar{x}_1, \quad f_2 = (x_0 + x_1), \quad f_3 = x_0$$

As proof, it is clear that the resulting equations represent the same functions before minimisation as shown below:

f_3 & f_2 are same before and after minimisation

$$\begin{aligned} f_1 &= \bar{x}_1 \\ &= 0 \odot x_1 \end{aligned}$$

4.3.3 Experimental Results

This algorithm was applied to several MCNC benchmark functions [70-72] to minimise the circuit finding the optimal expression with minimal terms among the ENPOS expressions. The results are given in column six of Table 4.16. *Terms for Optimum MPDRM among 3^n expressions* denotes to the optimum MPDRM terms doing exhaustive search using one of the new techniques described in Chapters 2 or 3. The minimal number of terms results using this algorithm are given in *minimal Terms*. The algorithm is tested on the same personal computer in section 4.1.3. The CPU times for all the examples were less than one second.

From the results, it is clear that some, but not all circuits can be minimised very well using this technique.

Table 4.16 shows that in 7 benchmark circuit tested from a total of 14 benchmarks, the results produced are better or same as results obtained using exhaustive search among 3^n MPDRM expressions. Saving in terms is calculated using equation (4.2) and shown in Fig. 4.3.

Table 4.16: Benchmark results for RM_Minimisation technique for DRM

Benchmark Number as in Fig. (4.3)	<i>Name</i>	<i>I/O</i>	CPOS/PPDRM Terms	Terms for Optimum MPDRM	Minimal Terms	Saving %
1	Dc1	4/7	10/3	2	2	0
2	root	5/8	256/136	83	86	-3.4
3	Con1	7/2	118/24	14	12	14
4	Inc	7/9	104/56	34	40	-15
5	Misex1	8/7	128/5	5	4	20
6	Sqrt8	8/4	255/11	11	9	18
7	Rd84	8/4	256/256	108	108	0
8	Risc	8/31	256/21	12	11	8.3
9	Clip	9/5	496/305	174	186	-6.4
10	Apex4	9/19	512/512	407	504	-19.2
11	Life	10/1	372/121	101	93	7.9
12	Ex1010	10/10	825/1023	825	1013	-18.5
13	Table3	14/14	3176/4553	421	1687	-75
14	Alu4	14/8	16384/1091	496	391	21.1
15	Misex3	14/14	12281/6442	803	1523	-47.2
16	Table5	15/17	24572/4421	554	835	-33.6
17	B12	15/9	32768/10	10	10	0

$$Saving = \frac{Asolute(Optimum MPDRM - Minimal terms)}{Largest\ of(Optimum\ MPDRM,\ Minimal\ terms)} \times 100\% \quad (4.2)$$

Largest of (Optimum MPDRM, Minimal terms): chooses either Optimum MPDRM or Minimal terms depending on which one is bigger.

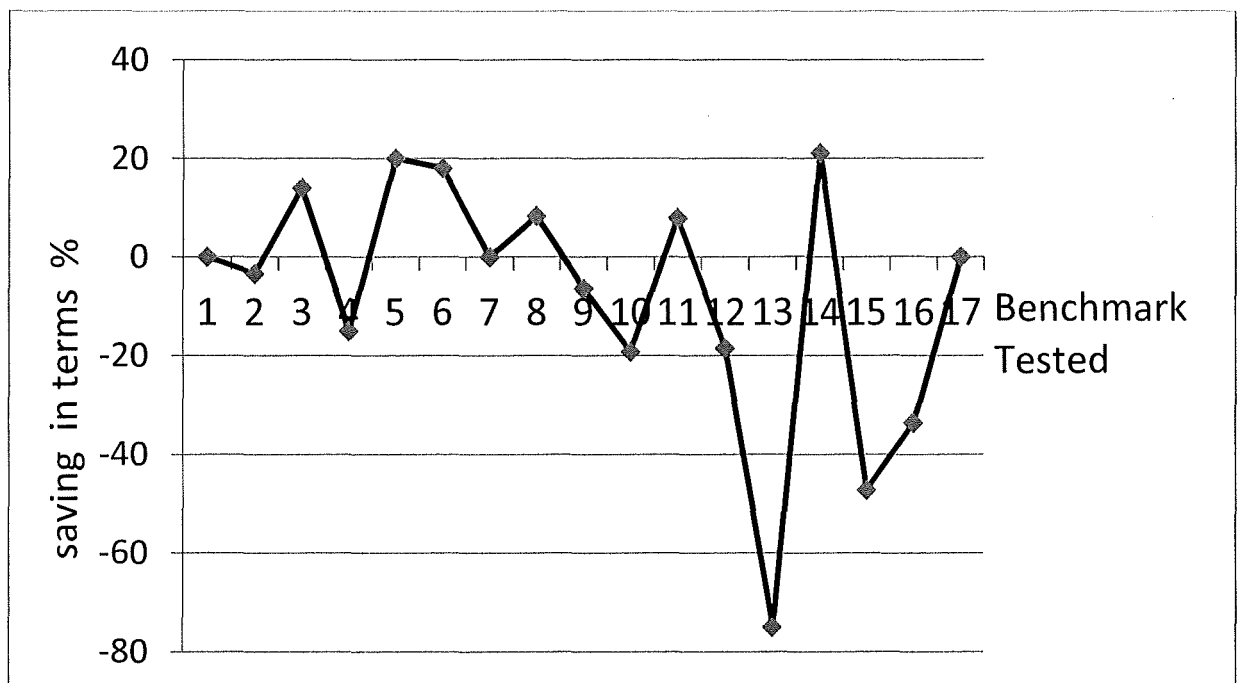


Figure 4.3: Chart for saving of terms for all tested Benchmarks shown in Table 4.16

4.4 Summary

In this Chapter, new techniques and algorithms are presented which can be used to generate reduced RM expressions among ESOP/or ENPOS expressions starting from PPRM/PPDRM, respectively for any number of variables for single and multi-output functions. All algorithms are implemented in C++ and fully tested using standard benchmark examples. These techniques are based on Tabular technique [31] and can be used manually or programmed on computers.

Chapter Five

Genetic Algorithms for Optimisation of Combinational Circuits

5.1 Introduction

Chapters 2 and 3 results show that the CPU time required to find an optimum polarity with minimum terms for Boolean functions, using exhaustive search, increases with the number of inputs. Table 5.1 shows the total number of expansions for MPRM/or MPDRM, also increases with the number of inputs. Therefore, to optimise digital circuits in RM domain with a large number of inputs, running an exhaustive search may take long time to find the optimum polarity among 3^n expressions. The aim for implementing a Genetic Algorithm (GA) based technique is to identify which of the MPRM/or MPDRM expansions contains the fewest product /or sum terms thus avoiding the time consuming exhaustive search.

GA techniques have been applied to a variety of optimisation problems; they can produce good results within a short computation time, when compared to exhaustive search.

Table 5.1: Total number of MPRM/or MPDRM expressions

Number Of Inputs	Total number of MPRM/or MPDRM expansions
5	243
7	2187
10	59049
20	3486784401
25	847,288,609,443

Hill Climbing, Simulated Annealing, and GA were considered for the task of optimisation. Then it is decided to adopt GA in this Chapter as it is more suitable for NP-Hard problems.

For completely specified Boolean functions of n -input variables, there exist 3^n MPRM/or MPDRM expansions with different numbers of terms. For incompletely specified functions, the numbers of MPRMs/MPDRMs increases exponentially with the number of “don’t care” terms. There are $(3^n \times 2^\mu)$ distinct MPRMs/or MPDRMs for n -variable functions with μ “don’t care” terms. The expression with the fewest products/or sums is the optimum expression.

The rest of this Chapter is organised as follows; Section 5.2 gives an introduction to GAs. The definitions for GA parameters are explained in Section 5.3. Section 5.4 describes the proposed GA to find the optimal MPRM expansion among 3^n different polarities for single output Boolean functions. A GA to find the optimal MPRM/and MPDRM expansion among 3^n different polarities for multi output Boolean functions is given in Section 5.5. The proposed GA to find the optimal MPRM expansion among of 3^n different polarities for incompletely specified Boolean functions is given in Section 5.6.

5.2 Genetic Algorithm

Genetic or Evolutionary Algorithms are applied to solve a variety of problems based on the principle of natural selection and genetic inheritance. These problems range from practical applications in industry to leading-edge scientific research [76]. However; evolutionary computation requires studying each problem in depth and expressing it in mathematical form. The proposed GA does not solve the problem directly; rather it

develops strategies for solving the problem, which is known as indirect representation.

Within genetic algorithm, the problem is encoded as a series of bit strings that are manipulated by the algorithm. The genetic algorithm is a stochastic algorithm which maintains a population of individuals that are usually represented as codes. Each individual represents one possible solution to the problem at hand; the solution may be evaluated to give some measure of its fitness. Some members of the population undergo transformation by means of genetic operators to produce new individuals. After a number of generations the algorithm converges, and the best individual is taken as a near-optimum solution. A GA for a particular problem must have the following five components [77]

- Representation for a potential solution to the problem
- A method to create an initial population of solutions
- An evaluation function (fitness) that evaluates the solution which is represented by a chromosome.
- Genetic operators that change the composition of individual such as *mutation* and *crossover* to produce new individuals.
- Values for various parameters that the GA uses such as population size, generation number, and tournament size.

The flow chart for simple GA is as shown in Figure 5.1

5.3 Definitions

Population: represents set of possible individuals (solutions) within the search space for the given problem. Normally, the population is initialised randomly.

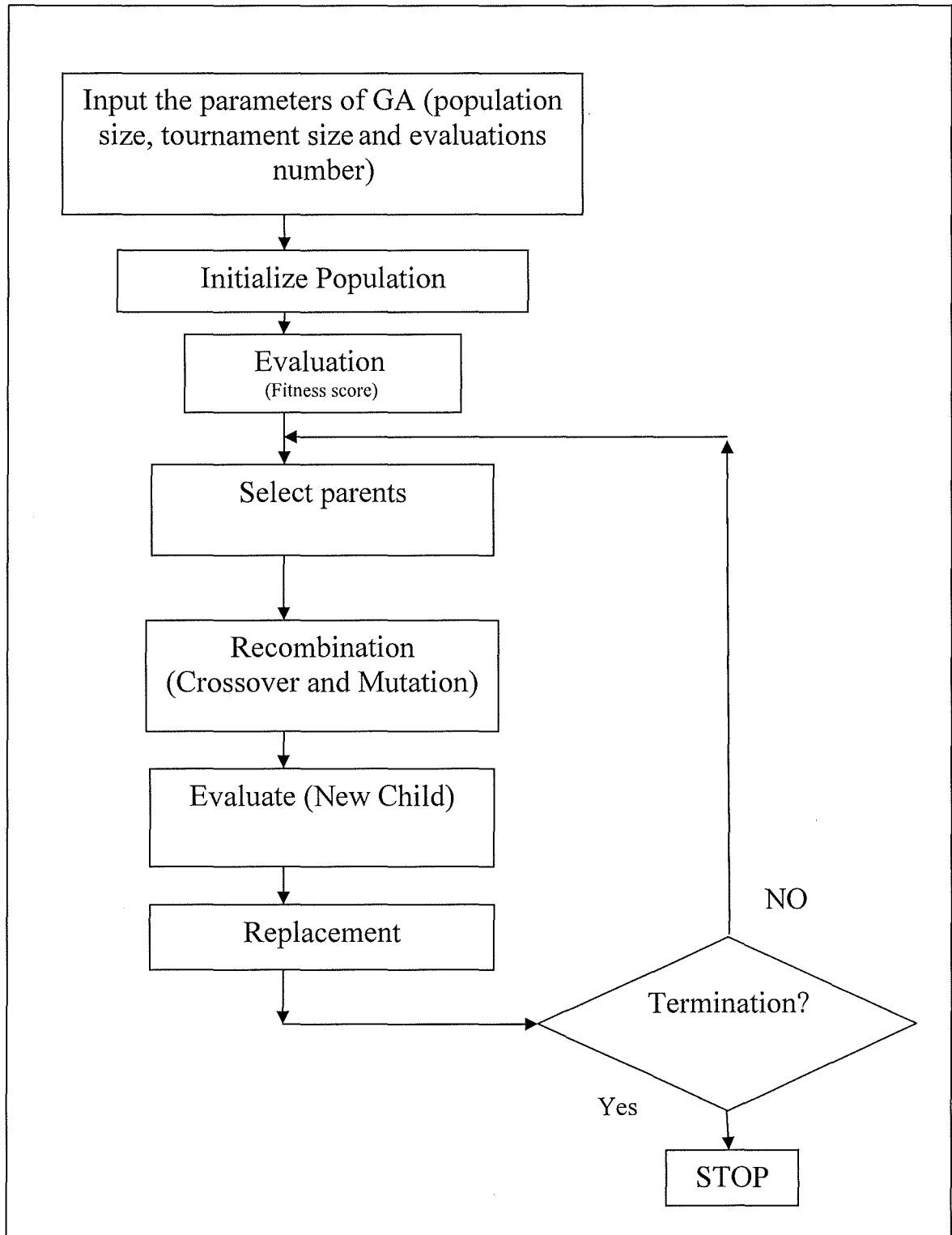


Figure 5.1: Basic flowchart of GA

Initialisation: the process of creation of a population of chromosomes randomly.

Individual: represent one possible solution to the problem. It is also known as chromosome. Individual which represent one member of the population can be coded using real numbers, binary code, bit string or any data structure. Each individual contains a number of genes as shown in Figure 5.2.

Population size: the number of chromosomes within the population which is initialised by the user before running the algorithm.

Search space: the set of all possible solutions for the given function. Each individual denotes a point in a search space.

Evaluation: A fitness function that assess an individual. It is an objective function which prescribes the optimality of the solution (individual) in GA.

Selection: the selection criterion aids the survival of the chromosome. In this stage of a GA, individuals are selected from a population for recombination. Methods of selection include tournament selection, random selection, roulette wheel selection, and ranking selection.

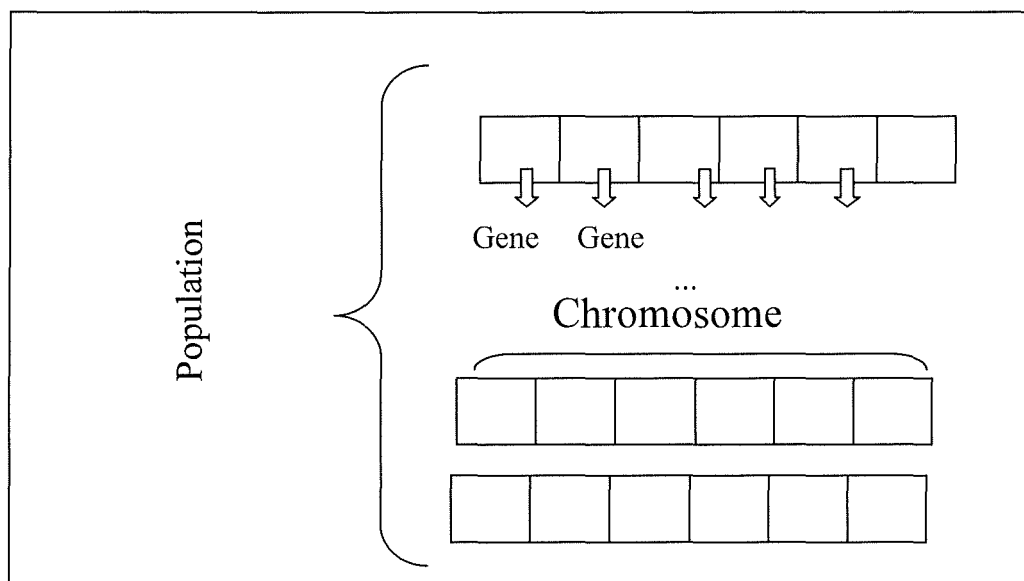


Figure 5.2: Chromosome representation of GA

Genetic Operators: two genetic operators are used to reproduce new individuals (offspring) from selected parents. These operators are recombination and mutation.

Recombination (or Crossover): corresponds to the mating between parents. Methods of crossover include single-site crossover, multi points, and uniform crossover.

Mutation: introduces random variation. Methods of mutation include flip mutation, invert mutation, and swap mutation.

Replacement: It replaces the offspring with one of the individuals within the population. Methods of replacement include tournament replacement, worst chromosome replacement, random chromosome replacement, and parent chromosome replacement.

Termination criteria: it represents the process of deciding whether to continue searching the search space or quit the search. Types of termination are generation number, evolution time, and fitness converges.

5.4 A GA for single output Boolean functions

In this section, a new algorithm is proposed to optimise RM expansions with mixed polarity, using a GA for a single output completely specified Boolean functions, thus avoiding the time consuming exhaustive search. The aim is to find the optimal polarity among MPRM/or MPDRM expansions which contain less terms without need to calculated all the 3^n polarities for n variables. Figure 5.3 gives the pseudo code of the proposed GA.

5.4.1 Population Initialisation

The population is randomly initialised. The chromosome is represented by the use of ternary code to represent the polarity number for MPRM/or

MPDRM expression. The size of the population is equal to the number of variables for the given function. Figure 5.4 shows the representation of one chromosome for six variables Boolean functions.

```

Procedure GA
{
Input Parameters of GA (population size, tournament size, number
of evaluations)
Read_Terms ( benchmark)
Initialize Population( randomly)
Fitness(all Populations)

Loop until (number of evaluations = 0)
{
    Tournament Select ( ) // select two parents randomly
    Crossover ( ) // recombine the two parents to produce
// the Child
    Mutation( ) // change one bit randomly in the Child
    Fitness( new Child )
    If (Child Fitness!=any existing Fitness)
// != indicates not equal
        Replacement ( )
    Decrease number of evaluations
}
Output Results ( )
}End GA

```

Figure 5.3: Pseudo Code for the proposed GA

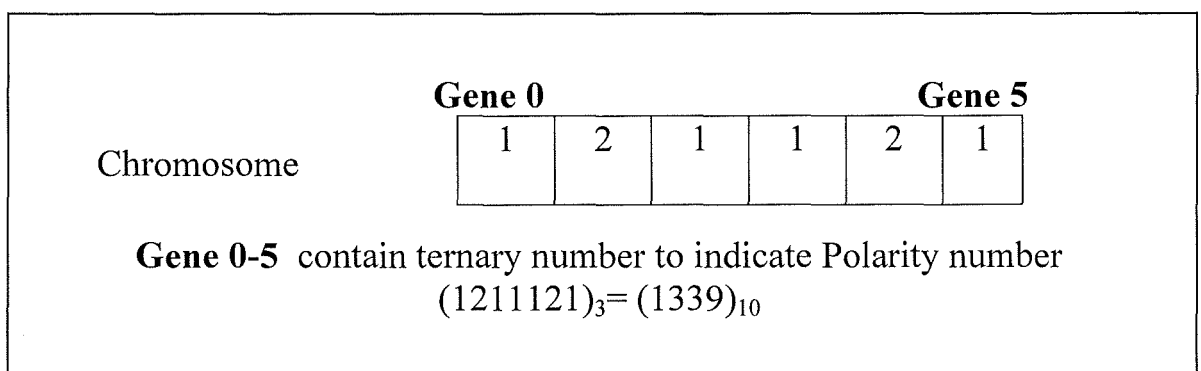


Figure 5.4: Chromosome representation for 6-variables Boolean function

5.4.2 Evaluation and Fitness score

The fitness function assesses the quality of new individuals. Initially, it computes the fitness of all the chromosomes after initialisation. Subsequently; it computes the fitness for the new offspring produced at each generation. The fitness function is implemented by either using Extended_Tabular techniques (Chapter 2) or using Minterm/Maxterm separation techniques (Chapter 3) to calculate the number of terms for the polarity determined in the selected individual.

5.4.3 Selection

The proposed GA uses a tournament selection method where the main parameter of selection is the *tournament size (T)* which can be changed by the operator. It is possible to control the selection pressure by adjusting the tournament size. The algorithm randomly chooses T individuals (independently to their fitness values) from the population and selects the one with the best fitness. If the tournament size is large, weak individuals have a less chance to be selected. The main advantages of using this method of selection are the simplicity to code and ability to adjust the selection pressure.

5.4.4 Reproduction (Crossover and Mutation)

Reproduction for the proposed algorithm produces one child at each evaluation. The two main variation operators are *crossover* (or recombination) which combines the genes of parents, and *mutation*, which slightly perturbs the child.

Crossover is the main genetic operator. It operates by selecting two individuals randomly (tournament selection) and generates one child. The child inherits some of the chromosomes from one parent and the rest from the other parent. The crossover operator chooses a random crossover point

and combines parts from the two parents to form a new child. This method of crossover is called a single point crossover as shown in Figure 5.5.

The mutation operator alters a single gene of the individual randomly. It is carried out by selecting gene at random and changing the selected gene with random ternary number as shown in Figure 5.6. This type of mutation is called Uniform mutation.

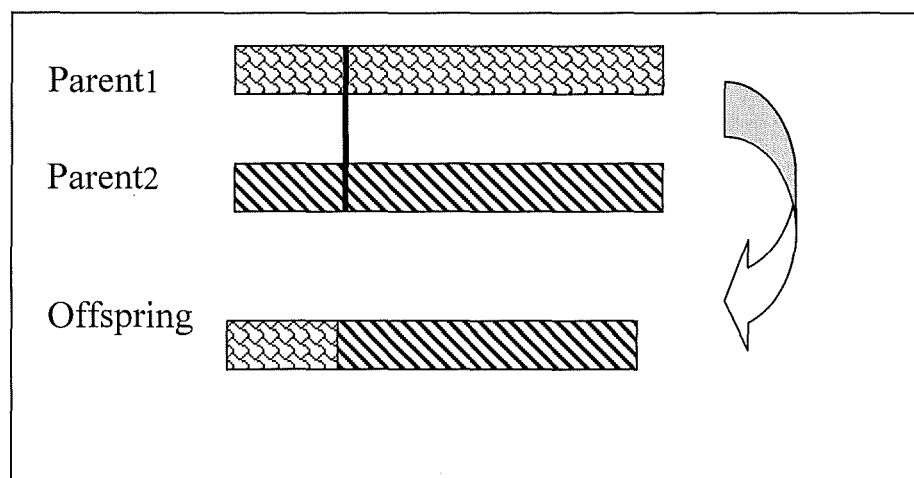


Figure 5.5: Single point Crossover used in the proposed GA

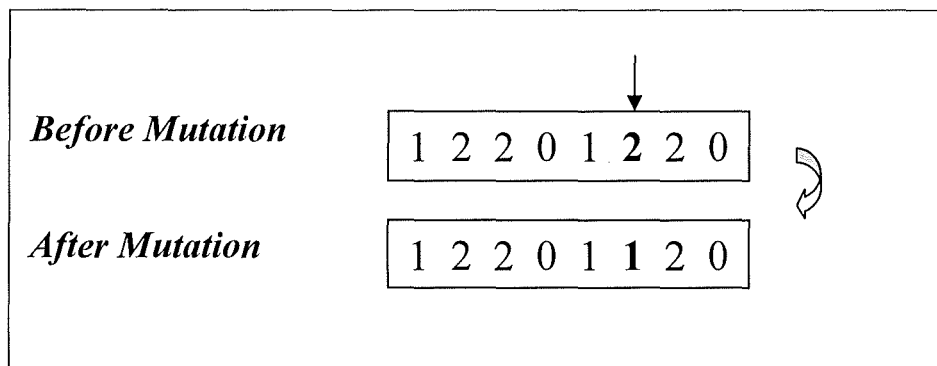


Figure 5.6: Uniform Mutation used in the proposed GA

5.4.5 Replacement

The replacement mechanism controls the composition of the new generation within each evolutionary loop. The algorithm randomly chooses (T) individuals (independently to their fitness values) from the population

and replaces the loser with the new offspring generated. The loser individual is one solution within population which has the worse fitness. One of the biggest problems encountered when using GAs is that of Premature Convergence. After number of generations, the GA converges to sub optimal, resulting in all the individuals within the population having the same fitness. In the proposed GA, to prevent Premature Convergence and avoid the loss of the diversity of the population, the algorithm will not replace the new chromosome if there is another individual with the same fitness in the current population.

5.4.6 Halting Criterion

A common strategy is to stop evolution after a fixed number of evaluations. The number of evaluations is one of the parameters determined by the operator prior to running the algorithm. It may be different for each example depending on the size of the given function. A good control of the halting criterion obviously influences the efficiency of the algorithm, and is as important as a good setting of evolution parameters (population size, crossover and mutation types, and tournament size).

5.4.7 Experimental Results for single output functions

The algorithm was applied to several MCNC [70-72] benchmarks completely specified CSOP Boolean functions to produce optimal MPRM expansion. It is implemented using C++ and is tested using a PC with INTEL CPU, 2.4 GHz clock and 2GB RAM. The results are given in column four of Table 5.2.

I/O denotes the number of inputs/number of outputs of the benchmark. *CSOP Term* denotes the number of sum of products of the benchmark function. The Optimal MPRM terms results using a GA are given in *GA*

MPRM Terms. GA Evaluations/3ⁿ shows how many times the GA is required to evaluate the number of terms for the specified polarity with respect to number of all polarities.

Table 5.2: Benchmark results of GA for single output Boolean functions.

Name	I/O	CSOP Terms	GA MPRM Terms	GA Evaluations/3 ⁿ .	MPRM Terms (Exhaustive)	STD/ Av.	CPU Time
Xor5	5/1	16	5	20 / 243	5	0/5	0.015 sec.
Newill	8/1	142	13	300/6561	13	0/13	0.047 sec.
newtag	8/1	234	6	300/6561	6	0/6	0.047 sec.
9sym	9/1	420	173	120/19683	173	0/173	0.055 sec.
Life	9/1	140	84	200 /19638	84	0/84	0.11 sec.
Sym10	10/1	837	266	200/ 59049	266	0/266	0.266 sec.
T481	16/1	42016	13	550 / 43046721	13	1.13/ 13.7	2.37 min.
Ryy6	16/1	19710	48	550 / 43046721	48	0/48	6.35 min.

The algorithm was implemented in order to evaluate the efficiency of GA for optimisation. All results from the proposed GA are taken after running the algorithm ten times in order to find the Standard deviation and average of these results which are given in *STD /AV*. *CPU Time* gives the average CPU time required to find the optimal terms by running GA ten times.

Exhaustive search was undertaken for each of the examples in Table 5.2 (column six) to identify the optimum solution. When these results were compared with the GA solutions, it was found that the GA found the optimum solutions for all the benchmark examples attempted. This

however cannot be guaranteed in all cases as this depends on the nature of the function and number of generations.

Population size indicates the quantity of chromosomes in the population. If there are too few chromosomes, only a small part of the search space is explored. Consequently, if there are too many chromosomes, the GA slows down. It was found that after a given size, it is not useful to increase the population size, because it does not make solving the problem faster. A good population size was found to be about 20-30 as shown in the following example.

For examples T481 & Ryy6 (MCNC benchmark) which both have 16 variables (i.e. each individual has 16 bits), Figure 5.7 shows that the best population size is around 20.

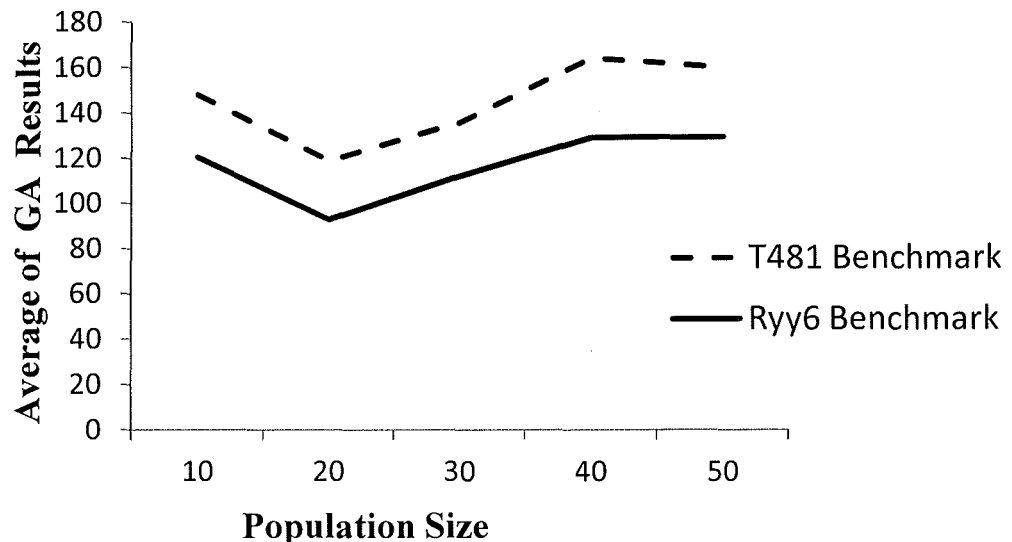


Figure 5.7: Average of GA results with respect to population size for the Benchmarks T481& Ryy6

Figure 5.8 shows that the best result was produced when the tournament size was between (3-7). Therefore the tournament size was chosen in this range for all benchmark examples.

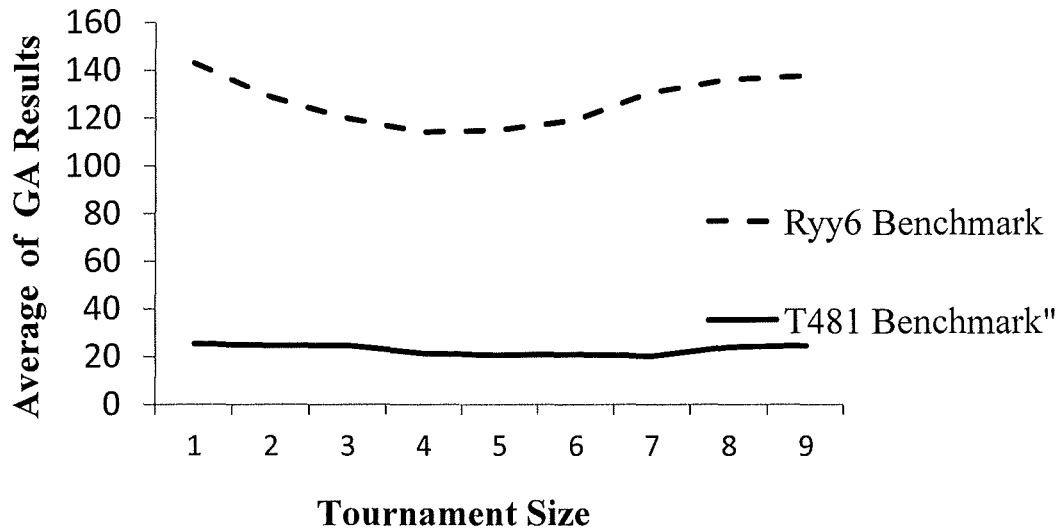


Figure 5.8: Average of GA result with respect to the Tournament size for Benchmarks T481 & Ryy6

The results show the efficiency of the GA (equation (5.1)) in the optimisation which reflects how the number of evaluations (column five of Table 5.2) using GA is much less than the number of evaluations to find all polarities which equals 3^n for n number of variable. From Figure 5.9, it can be seen that the efficiency of GA increase with the number of variables of the specified function. For example Newill benchmark needs to calculate the number of terms for only 300 random polarities using GA to find the optimal one, while it needs to calculate the number of terms for 6561 polarities using exhaustive search.

$$\text{Efficiency} = \frac{3^n - \text{GA evaluations}}{3^n} \times 100\% \quad (5.1)$$

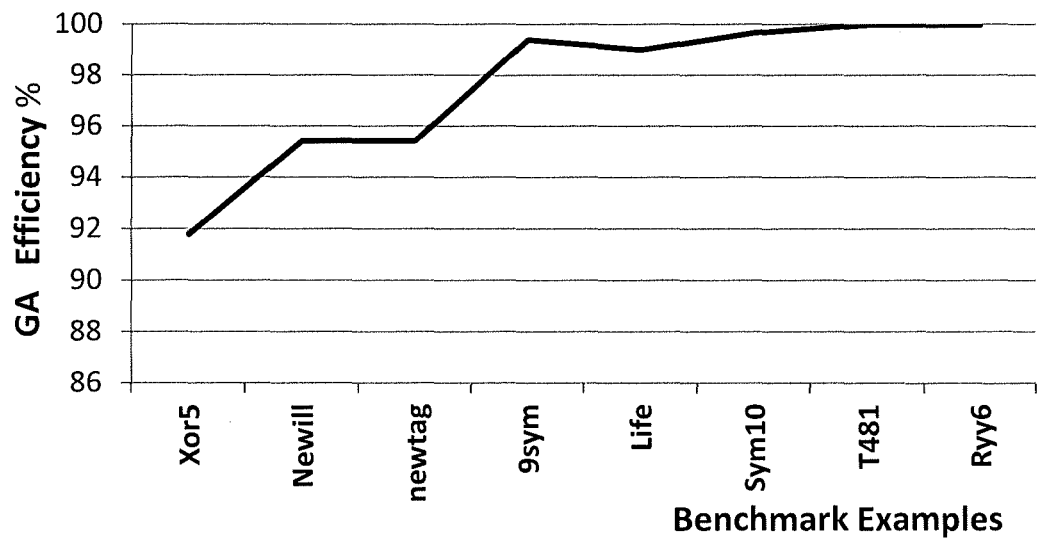


Figure 5.9: Efficiency of GA, calculated as eq. (5.1)

The time required to optimise the function depends on number of variables and minterms. It can be seen from the results shown in Table 5.2, that when the function becomes larger, the algorithm needs more time to find the optimal solution. Therefore when the algorithm was running less than the time required to find the optimal polarity, the result will be sub optimal. Figure 5.10 shows the difference in the average of results for the benchmarks T481 & Ryy6 with respect to the number of GA evaluations as in Table 5.2.

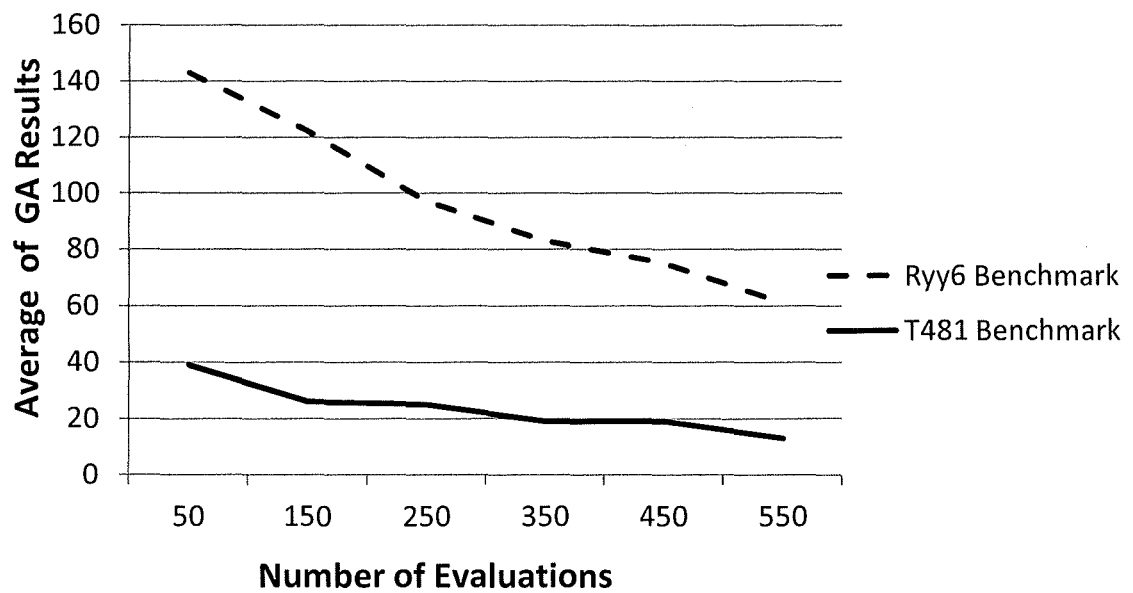


Figure 5.10: Average of GA results with respect to evaluations for the Benchmarks T481& Ryy6

By testing the proposed algorithm with more than one type of crossover and mutation, it was found that the single point crossover and uniform mutation used in the proposed algorithm produce good results compared to other types of crossover and mutation.

Table 5.3 shows the comparison between two point crossover and single point crossover. The comparison is made between two point crossover with producing two offspring for every evolution loop and single point crossover with producing one offspring for every evolution loop. It can be seen from the results of Table 5.3 for single point crossover with one offspring per loop need less number of evaluations to find optimal than two point crossover.

Table 5.3: Experimental results for the pdc Benchmark

GA Single point Crossover and one offspring for every evolution loop		GA Two point Crossover and two offspring for every evolution loop	
<i>Evaluations</i>	<i>Terms</i>	<i>Evaluations</i>	<i>Terms</i>
700	157	700	204
500	162	500	241
400	166	400	241

5.5 GA for multi-output Boolean functions

The GA is implemented to handle multi-output functions using two possible methods for implementing its fitness function. Either using Extended_Tabular techniques (Chapter 2) or Minterm/Maxterm separation techniques (Chapter 3). Both GAs are implemented with the same details as described in section 5.4. It is clear that each output in a multiple output function can be minimised either independently as described in section 5.4 or using Extended_Tabular technique for multiple output as described in section 2.5.2, considering the savings of terms which can often be achieved through sharing of terms between outputs.

5.5.1 GA using Extended_Tabular techniques

In this section; GA is implemented with the fitness calculation based on Extended_Tabular technique (Chapter 2). This algorithm adopts two approaches, allowing outputs to have different polarities in one and the same polarity in the other.

In the first approach, the population represents the polarity number for all outputs at the same time. Therefore, the number of genes within the chromosome will be equal to (number of inputs \times number of outputs). For example, if the CSOP Boolean function has three inputs with four outputs

then the population will be 12 bits (3 bits for each output to represent its polarity) while in the second approach the chromosome represents one polarity for all the outputs. This means that the number of genes within each chromosome will equal the number of inputs. For example, if the standard Boolean function has three inputs with four outputs, then the chromosome contains three genes to hold one polarity which will represent the polarity for all the outputs at same time.

This means, that the algorithm in the first approach will search for a good solution among o^{3^n} polarities to produce a polarity with less number of terms, providing outputs with different polarities (where n and o are the number of inputs and outputs respectively). In the second approach the algorithm will search for the optimum polarity among 3^n polarities resulting in outputs of the same polarity. All other operators of the GA are the same as GA described in section 5.2.

The proposed algorithm was applied to several MCNC benchmark functions [70-72], implemented using C++, and tested using a PC with an INTEL CPU, 2.4 GHz clock and 2GB RAM. The results are given in columns four and five of Table 5.4. *I/O* denotes the number of inputs/outputs respectively for the benchmark. *CSOP Terms* denote the number of product terms for the specified benchmark. The number of MPRM terms which results using GA algorithm for the multi-output functions are given in *Terms*. *Optimum* gives the minimum number of terms for the optimum polarity running exhaustive search to calculate all the 3^n polarities. This algorithm is implemented to test and compare the results obtained using GA with the real optimum polarity running exhaustive search.

Table 5.4: Benchmark results of GA for multi-output Boolean functions

Name	I/O	CSOP terms	Terms / Time <i>(different polarities)</i>	Terms / Time <i>(same Polarities)</i>	Optimum Terms/Time (Exhaustive)
Rd53	5/3	32	20/ 0.017 sec.	20/ 0.015 sec.	20/ 0.015 sec.
Squar5	5/8	29	21/ 2.016 sec.	26/ 0.016 sec.	26/ 0.016 sec.
Con1	7/2	118	13/ 0.074 sec.	14/ 0.022 sec.	14/0.125 sec.
Rd73	7/3	127	63/ 2.76 sec.	63/ 0.039 sec.	63/0.125 sec.
Inc	7/9	104	34/ 2.40 sec.	34/ 0.031 sec.	34/0.140 sec.
5xp1	7/10	128	48/ 8.76 sec.	61/ 0.033 sec.	61/0.144 sec.
Ex5p	8/63	256	216 / 12.087 sec..	104 / 1.062 sec.	104 / 0.985 sec.
Rd84	8/4	255	107/ 2.75 sec.	107/ 0.094 sec.	107 /0.797 sec.
Sqrt8	8/4	255	26/ 0.922 sec.	26/ 0.094 sec.	26/ 0.766 sec.
Misex1	8/7	128	16/ 2.143 sec.	13/ 0.078 sec.	13/0.78 sec.
Clip	9/5	496	137 / 22.56 sec.	182/ 0.110 sec.	182/ 5.67 sec.
Sao2	10/4	511	73/ 1.36 sec.	76/ 0.328 sec.	76/38.96 sec.
Ex1010	10/10	1024	948/ 6.65 min.	810 / 0.172 sec.	810 /43.95 sec.
Table3	14/14	3176	406/ 35.83 min.	401/ 11.39 min.	401/ 1 day & 2 h.
Alu4	14/8	16384	1954/ 1.640 min.	2438/ 16.013 min.	2438/ 1 day & 2h.
Misex3	14/14	12281	1479/ 2.654 min.	1421/ 13.875 min.	1421/ 1 day & 3 h.
Table5	17/15	24572	833/42.43 min.	551/2.575 min.	551/14 days&8 h.

The results shown in Table 5.4, suggest that the algorithm running for the first approach is better than the second approach regarding the number of terms but it requires more time because of the wide range of polarities (solutions). It was noticed that for some benchmark examples, the algorithm can produce better results if given more time. The average savings in terms compared with CSOP terms are 64% and 60% depending on whether the outputs have different polarities or same polarity respectively.

5.5.2 GA using Minterm/Maxterm separation techniques

In this section, GA's were implemented with a fitness function implemented using Minterm/Maxterm separation techniques (Chapter 3). The GAs finds the optimum polarity among MPRM and MPDRM. The motivation behind the finding of the optimum MPDRM as well as MPRM is that for some functions, these dual forms can have fewer terms than any normal forms.

These algorithms were also applied to several MCNC benchmark functions [70-72] to find optimum polarity among MPRM/ and MPDRM expansions. The number of MPRM/and MPDRM terms which results using the GA based algorithm for the multi output functions are given in *columns 4 & 5* of Table 5.5. *Column 6* shows how many times the GA is required to evaluate number of terms for MPRM. All results from the proposed GA (columns four and five) are taken after running the algorithm ten times in order to find the Standard deviation and average of these results given in *STD /AV. column*.

Note that some of the benchmark examples in Table 5.5 don't have results for MPDRM as these benchmarks don't have CPOS terms.

Table 5.5 Benchmark results of GAs using Minterm/Maxterm separation techniques

<i>Name</i>	<i>In/Out</i>	<i>CSOP Terms</i>	Terms for Optimal MPRM	Terms for Optimal MPDRM	GA evaluation (MPRM)	STD/AV. (MPRM)
Dc1	4/7	10	10	2	100	0/10
Rd53	5/3	32	20	-	200	0 / 20
Con1	7/2	118	14	14	200	0 / 14
Rd73	7/3	127	63	-	200.	7.074 / 66.6
Rd84	8/4	256	107	108	200	13.94/113.6
clip	9/5	496	182	174	300	1.34/182.6
Misex1	8/7	128	13	5	200	0.42/13.2
Sqrt8	8/4	255	26	11	200	1.54/27.2
Sao2	10/4	511	76	-	200	1.62/77
Ex1010	10/10	1024	810	825	200	21.1/816.7
Inc	7/9	104	34	34	200	1.475/34.8
5xp1	7/10	128	61	-	200	0.51/61.5
Apex4	9/19	512	444	407	400	1.032/444.8
Dk17	10/11	64	30	8	400	1.22/31.2
Table3	14/14	3176	401	421	500	0/401
Alu4	14/8	16384	2438	496	700	14.8/2451.5
Misex3	14/14	12281	1421	803	700	16.1/1426.1
Table5	15/17	24572	551	554	600	1.54/552.2
B12	15/9	32768	64	10	1000	0.843/64.4
T481	16/1	42016	13	-	550	1/13

The average CPU times for both GA's (sections 5.5.1 & 5.5.2) are compared in Figure 5.11. Both results are obtained from two algorithms which were running on the same PC and same operating systems.

The only difference between the two GAs is the method used to calculate number of terms for each mixed polarity in the fitness function. The comparison is made between the proposed GAs with two different methods used in its fitness function. The results show that GA using Minterm/Maxterm separation techniques required less time to produce optimal polarity.

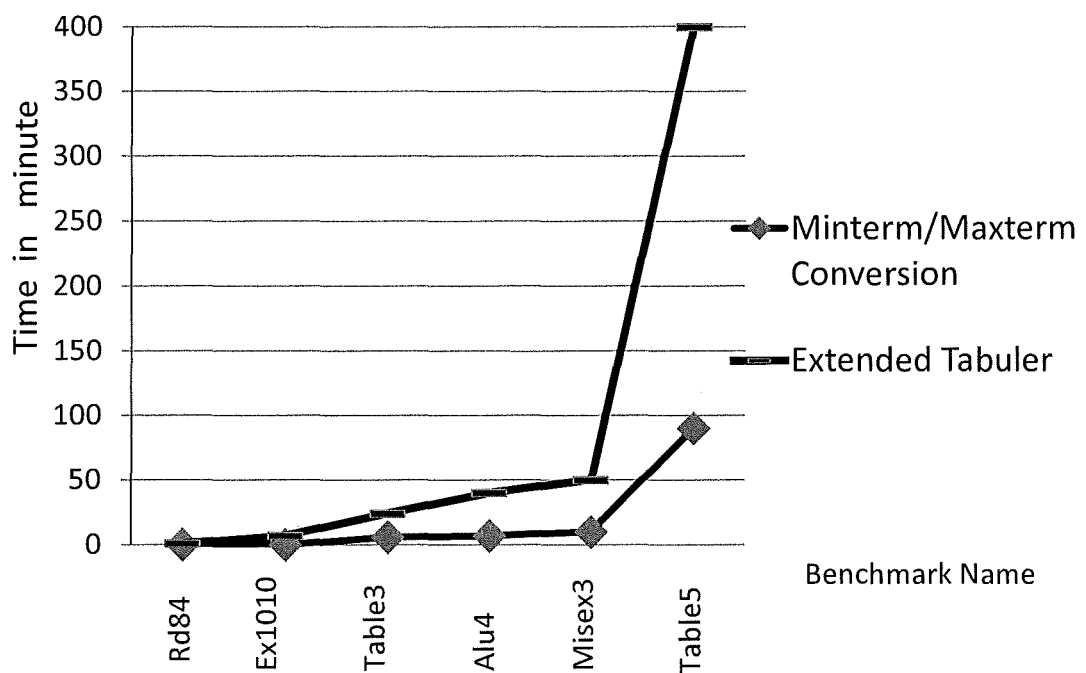


Figure 5.11: Time Comparison between GAs

Table 5.6 shows the time comparison which was made for the large benchmark examples. This time comparison is made between the exhaustive search using techniques explained previously in Chapters 2 & 3 and the proposed GA for finding optimal polarity among 3^n expansions. It is clear that the proposed GA requires less time than time required doing exhaustive search finding the optimum mixed polarity for all tested benchmark although can't guaranteed for other functions. Therefore the

GA can save time and effort to find the optimal mixed polarity without the need to find all the 3^n polarities for the specified function.

Table 5.6 Time Comparison for large Benchmarks

<i>Benchmark Name</i>	<i>I/O</i>	CPU TIME (CH.2)	CPU TIME (CH.3)	CPU TIME (GA)
Apex4	9/19	7.04 sec.	3.625 sec.	0.156 sec.
Dk17	10/11	39.42 sec.	9.25 sec.	0.453 sec.
Alu4	14/8	1 day & 2 h.	2 h. & 35 min.	16.013 sec.
Misex3	14/14	1 day & 3 h.	4 h. & 20 min.	13.875 sec.
T481	16/1	14 days	15 h. & 12 min.	2.37 min.
B12	15/9	10 days & 4 h.	15 h. & 32 min.	1.052 min.
Table5	15/17	14 days & 8 h.	5 days & 4 h.	2.575 min.

Table 5.7 shows a comparison for number of products/or sums between the proposed GA for finding optimal polarity among 3^n expansions and RM_Minimisation techniques explained previously in Chapters 4 for producing reduced MPRM/or MPDRM expansion.

It was found from the results of Table 5.7, that the results produced using RM_Minimisation technique for 12 benchmarks from the total of 20 benchmarks, are the same as or better than results produced by the proposed GA to design the circuits in RM domain. Further, in 8 benchmarks from the total of 14 benchmark examples, RM_Minimisation technique produced same as or better than results of the proposed GA to design the circuits in DRM domain.

Table 5.7 Comparison between GAs and RM_Minimisation techniques

<i>Name</i>	<i>In/Out</i>	GA Optimal MPRM	RM_Minimisation MPRM	GA Optimal MPDRM	RM_Minimisation MPDRM
Dc1	4/7	10	16	2	2
Rd53	5/3	20	20	-	-
Con1	7/2	14	11	14	12
Rd73	7/3	63	63	-	-
Rd84	8/4	107	107	108	108
clip	9/5	182	155	174	186
Misex1	8/7	13	37	5	4
Sqrt8	8/4	26	47	11	9
Sao2	10/4	76	65	-	-
Ex1010	10/10	810	1007	825	1013
Inc	7/9	34	69	34	40
5xp1	7/10	61	60	-	-
Apex4	9/19	444	444	407	504
Dk17	10/11	30	808	8	8
Table3	14/14	401	1067	421	168
Alu4	14/8	2438	1790	496	391
Misex3	14/14	1421	1233	803	1523
Table5	15/17	551	2956	554	835
B12	15/9	64	54	10	-
T481	16/1	13	13	-	-

In Table 5.8, the proposed GA is tested for large MCNC benchmark [70-72] and compared with results produced recently by [48]. Reference [48]

developed algorithm which combined annealing process with genetic operations to minimise MPRM to improve the performance of the GA and achieved better results than simulated annealing and genetic algorithm. The results of comparison show that the GA proposed here achieved better or same results for all benchmark tested except one benchmark in which [48] is better. Experimental results (column four) show that the proposed GA is also efficient for large functions with a total saving of 102 terms compared with [48].

Table 5.8: Comparison between the proposed GA and Reference [48]

Name	In/Out	Optimal MPRM [48]	Optimal MPRM GA
Ryy6	16/1	48	48
T481	16/1	13	13
b9	16/5	105	105
b2	16/17	333	333
alcom	16/38	68	25
spla	16/46	687	628
table5	17/15	559	551
in2	19/10	262	262
shift	19/16	100	108
t1	21/23	209	209
ts10	22/16	136	136
duke2	22/29	209	209
cordic	23/22	1980	1980
Total Terms		4709	4607

5.6 GA for Incompletely specified functions

An incompletely specified Boolean function is a function with one or more minterms with undefined values as in equation (1.7). These unspecified

minterms are known as “don’t care” terms and sometimes can help the process of minimisation.

When incompletely specified Boolean functions are transformed to the RM domain, “don’t care” terms are transformed along with the specified terms and their effect is distributed over several terms of the new representation. Therefore, it is necessary to find an optimum selection of these terms to minimise the number of terms in the expressions.

For incompletely specified functions, the coefficients will be {0, 1, or “don’t care”} terms. However, when the coefficient is undefined, it may take the value 0 or 1 without effecting any change to the output of the function. These undefined products are unspecified and called “don’t care” terms for the given function. Then the RM expansion may be denoted an incompletely specified RM expansion.

5.6.1 Single stage GA for Incompletely specified functions

A Genetic Algorithm based approach is presented to minimise the number of terms of MPRM single and multi-output incompletely specified Boolean functions. The algorithm determines the allocation of “don’t care” terms for the given function resulting in optimal MPRM expansions.

The proposed algorithm uses two populations each with their own representation. The first population is represented using ternary numbers to hold the polarity number of the RM expansion. The size of the polarity population equals n-bits for n-variable functions. The second population is represented using binary numbers to indicate the presence or absence of “don’t care” terms. The size of the second population equates the total number of “don’t care” terms for all outputs of the given functions.

Example 5.1: Consider multi-output functions as shown below:

$$f_1(x_2, x_1, x_0) = \sum m(7,3,1) + d_{ci}(5,4,6)$$

$$f_2(x_2, x_1, x_0) = \sum m(7,2,0) + d_{ci}(3,1)$$

$$f_3(x_2, x_1, x_0) = \sum m(6,5,4) + d_{ci}(3,2,1,0)$$

It is clear that the given functions for three variables have different set of terms and different number of don't care terms. The individuals for both populations will be as shown in Figure 5.12.

Gene (0–2) in Polarity individual within the 1st population are represented by ternary number to indicate polarity number $(121)_3 = (16)_{10}$

Gene (0–8) in “don't care” individual within the 2nd population are represented by binary number to indicate the existence of “don't care” terms.

Gene 0		Gene 2
1	2	1

(a) Polarity individual

Gene 0				Gene 8				
1 st output			2 nd output		3 rd output			
1	1	0	1	0	1	0	0	0

(b) “don't care” individual

Figure 5.12: Details of individuals for Example 5.1

This individual indicates the existence of two “don’t care” terms for the first output, one “don’t care” terms for the 2nd output, and one “don’t care” term for the 3rd one.

Therefore; for these combinations of don’t care, the functions will be as follows:

$$f_1(x_2, x_1, x_0) = \sum m(7,5,4,3,1)$$

$$f_2(x_2, x_1, x_0) = \sum m(7,3,2,0)$$

$$f_3(x_2, x_1, x_0) = \sum m(6,5,3,4)$$

The fitness function computes the number of terms for each individual. Initially it computes the fitness of all the chromosomes and then it computes the fitness for the new offspring of each evaluation. The fitness function is implemented to produce the number of terms for the polarity number determined in the 1st population including the selected “don’t care” terms determined in the 2nd population.

5.6.2 Scheme for Single stage GA

The scheme of the single stage GA for completely and incompletely specified Boolean function is as follows:

- 1 - Input the parameters of the GA and store them. These parameters are text file name (which contains the Boolean function), population size, tournament size (T) and number of evaluations.
- 2 - Open and read the text file to store the number of variables, number of terms, number of “don’t care” terms (if any) and all the terms and “don’t care” terms. The terms are stored in the binary array as ones or

zeros which indicate the existence or not of the terms respectively and the value of the term will be specified by the rows' locations of the array.

- 3 - Convert the function from Boolean expansion to zero polarity FPRM expansion using Tabular technique without considering the "don't care" minterms.
- 4 - Initialise randomly the two populations, the 1st with a random ternary number, and the 2nd (if there are "don't care" terms for the specified function) with binary number to indicate existence of the "don't care" terms.
- 5 - Save number of evaluations which is determined by the user.
- 6 - Evaluate the number of terms for all individuals of the current population which contain polarity number in ternary code. The function here checks the 2nd population (if "don't care" exists) to convert "don't care" terms from the Boolean domain to RM domain and add them to the list of the terms. Then continue to find the number of terms for the polarity determined in the first population.
- 7 - Select two parents from each population by using tournament selection method.
- 8 - Perform single point crossover twice (if "don't care" exist). Once for each population.
- 9 - Mutate the two populations by changing one bit randomly in each one.
- 10 - Evaluate the fitness of the child.

- 11 - Choose randomly T parents and determine the worst one which has higher number of terms.
- 12 - Replace the new child with the worse one selected in step 10. Check if another individual in the current population has the same fitness as the new child then skip this step without replacement. This step is carried out to avoid premature convergence.
- 13 - Evaluations = Evaluations – 1
- 14 - If Evaluations $< > 0$ then go to Step 7.
- 15 - Else stop and print the optimal polarity.

5.6.3 Multi stage GA for Incompletely specified functions

The single stage GA based approach was tested with different benchmarks examples and it was found that the algorithm can produce good result for small functions with small number of don't care terms. For functions with large number of variables and “ don't care ” terms, the problem become more complex due to the large search space which is equal $(3^n \times 2^\mu)$.

To overcome this problem, a two stage GA is proposed. The proposed GA has two search spaces depending on two different variables (n & μ). The 1st search space is 3^n while the other is 2^μ . The GA splits into two stages to cope with large functions. The first stage is to produce the best polarities among MPRM expansions without including “don't care” terms. The number of individuals resulting from the first stage equals to the population size which is one of the GA parameters entered by the user to run the algorithm. The aim of this stage is to minimise the search space saving computation time. The second stage will use the resulting best individuals from the first stage with different collections of “don't care” terms to

further simplify the expressions. The pseudo code of the proposed GA is shown in Figure 5.13.

The fitness function is implemented to convert the specified Boolean function to the RM domain and computes the number of terms for the polarity specified in the polarity population including the selected “don’t care” terms in the 2nd population.

As outlined above, the proposed GA has two stages. Therefore, it has two fitness functions; the 1st fitness function compute the number of terms for the polarity represented by the polarity population without considering the “don’t care” terms. The 2nd fitness function calculates the number of terms for the individuals being evaluated including the don’t care terms specified in the 2nd population. Each new member of the population is based upon individuals selected from the first stage population being combined with the second stage population.

```
GEN_NO      // Number of evaluations
Pop_size    // size of the populations
tnsize      // Tournament size
GA_INCOMPLETELY( )
{
  Input EA-parameters
  Read the given function in Boolean domain
  Randomly initialise the population for the polarity
  Randomly initialise the population for the don't care
  First_stage_evaluation=GEN_NO/3
  Second_stage_evaluation=2* GEN_NO/3
  Loop for (First_stage_evaluation)
  {
    Tournament Select (P1) // select two parents from the
                          // polarity population
    Crossover ( ) // single point crossover to produce
                // Child1
    Mutation( ) // change one bit randomly in the Child1
    Fitness1( Child1 ) // find fitness (No. Of terms) without
                      // including the don't care terms
    If (Child1_Fitness != any existing Fitness)
                          // != indicates not equal
      Replacement( )
    First_stage_evaluation= First_stage_evaluation-1
  }
  If (don't care terms !=0 ) //if there are don't care terms
  {
    Loop for (Second_stage_evaluation)
    {
      Tournament Select (P2) //select two parents from the don't
                            //care population
      Crossover ( ) //single point crossover to produce Child2
      Mutation( ) // change one bit randomly in the Child2
      Tournament Select (P1 ) //select one parent from the
                             // polarity population
      Mutation(P1 ) // change one bit randomly to produce Child1
      Fitness2(Child1,Child2) // find fitness (No. Of terms)
                             // including the don't care terms
      If (Child2_Fitness != any existing Fitness)
        Replacement for the Child1.
        Replacement for the Child2.
    }
  }
}
} End of GA_INCOMPLETELY
```

Figure 5.13: Pseudo code of multi stage GA

Example 5.2: Consider the incompletely specified Boolean function with three inputs and two outputs as shown in Table 5.9.

Table 5.9: Truth table for Example 5.2

Inputs $x_3x_2x_1$	Outputs	
	Y_1	Y_2
000	0	1
001	1	“don’t care”
010	0	1
011	“don’t care”	0
100	1	1
101	“don’t care”	0
110	1	“don’t care”
111	“don’t care”	1

This function can be represented using polarity population with three bits and “don’t care” population with five bits. The two populations are used to represent mixed polarity/“don’t care” terms using ternary/binary code, respectively.

- 1) Assuming that the user specifies population size= 7. Then, 7 individuals to represent polarity are initialized randomly using ternary code as detailed in Table 5.10.

Table 5.10: Initialisation of the polarity population for Example 5.2

Polarity Population			Polarity number (Decimal)
Bit 2	Bit 1	Bit 0	
2	1	1	22
0	1	1	4
1	2	0	15
1	2	2	17
1	0	2	11
2	1	0	21
1	2	1	16

2) Another 7 individuals to represent “dont care” selection are initialized randomly using binary numbers as shown in Table 5.11. Consequently, there are 3^3 possible polarities and for each one of these polarities there are 2^5 possible collections of “don’t care” terms. Hence, there are $(3^3 \times 2^5 = 864)$ possible solutions. Tables 5.11 and 5.12 shows different truth vector for each output for the different solutions.

Table 5.11: Initialisation of the “don’t care” population for Example 5.2

Sol. No.	“don’t care” population					Information
	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
1)	1	0	1	0	1	The 1 st and 3 rd don’t care for the first output exist and the 2 nd don’t care for the second output exists.
2)	0	1	1	1	0	The 2 nd and 3 rd don’t care for the first output exist and the 1 st don’t care for the second output exists.
3)	0	1	1	0	1	The 2 nd and 3 rd don’t care for the first output exist and the 2 nd don’t care for the second output exists
4)	1	0	0	1	0	The 1 st don’t care for the first output exists and the 1 st don’t care for the second output exists.
5)	1	1	1	0	1	All the don’t care for the first output exist and the 2 nd don’t care for the second output exists.
6)	1	1	0	1	1	The 1 st and 2 nd don’t care for the first output exist and all the don’t care terms for the second output exist.
7)	0	0	0	1	0	Only the 1 st don’t care term for the second output exists.

3) The GA splits into two stages to improve its performance. The GA selects two parents from the populations using tournament selection. Crossover is then applied to produce child which has a mutation applied. Finally, it replaces one of the individuals in the population with the child if the number of terms for the child is less than the number of terms for the parents.

Table 5.12: Truth vector for outputs of Example 5.2

<i>Solution No. (as in Table 5.11)</i>	Truth vector of the 1st output for the given function	Truth vector of the 2nd output for the given function
1)	01011011	10101011
2)	01001111	11101001
3)	01001111	10101011
4)	01011010	11101001
5)	01011111	10101011
6)	01011110	11101011
7)	01001010	11101001

4) In the first stage, the algorithm will run for one third of the number of evaluations determined by the user. It will find the number of terms for each individual from the polarity population without considering the “don’t care” terms. The GA will produce the optimal individuals with less number of terms as specified. The fitness function is implemented for converting the function from the Boolean domain to RM domain. When GA is running, the best seven individuals (7 as determined by user for population size) are produced as shown in Table 5.13.

5) In the second stage, the algorithm will run for two third of the number of evaluations determined by the user. It will add the selected “don’t

care” terms to the truth table for the given function as explained in Table 5.10 according to the individuals from the “don’t care” population in Table 5.11. Then convert the function from the Boolean domain to RM domain and calculate the number of terms for the incompletely specified Boolean function for each output considering the sharing between these outputs.

Table 5.13: Best individuals produced from the first stage of the proposed GA for Example 5.2

Polarity number in ternary code			Number of terms
0	1	2	6
1	0	1	5
2	1	2	6
1	2	0	7
1	2	1	6
0	1	0	7
0	0	1	7

5.6.4 Experimental Results for single & multi stage GA

The program was applied to several MCNC benchmark [70-72] functions. The algorithm was executed on a personal computer with an Intel CPU running at 2.4 GHz and 2 GB RAM under Window XP, Professional. The algorithms are implemented using C++. The results of multi stage GA are given in column six of Table 5.14. Each result from GA is taken after running the GA algorithm ten times. *Terms* denotes to the number of canonical sum of products terms for the given benchmark. Number of “don’t care “terms are given in *N.DC*. Minimal number of terms is given in

M. terms. Saving in column 7 denotes percentage saving in terms when “don’t-care” terms are included.

Table 5.14 Benchmark results for the multi stage GA

Name	I/O	CSOP Terms	N. DC	M. Terms / Time without DC	M. Terms / Time (with DC)	Saving %	AV/ STD
Xor5	5/1	16	16	6/0.037 sec.	1 /0.037 sec.	83	1/ 0
Rd53	5/3	31	54	20/ 0.046 sec.	6 /1 min.	70	10.1/ 1.3
Rd73	7/3	127	63	63 /0.891 sec.	63 / 2 sec.	0	63 / 0
Squar5	5/8	29	32	26/0.073 sec.	23 /1 sec.	11	24 /1.1
Sym10	10/1	837	187	306/1.32 sec.	64 / 1min.	92	86/ 16.7
9sym	9/1	420	92	173/1.87 sec.	34 /40.65 sec.	72	52 /8.9
life	10/1	140	372	84 /1.54 sec.	40 / 2.86 sec.	52	44/8 .2
clip	9/5	496	80	182/8.54 sec.	182 /8.78 sec.	0	182 / 0
newtag	8/1	234	22	6/1.65 sec.	1 / 16.53 sec.	66	2 / 0.5

AV. /STD denotes to average number of terms for the ten runs of the proposed GA / Standard deviation. STD tells how closely a set of results is clustered around the average of the results. If all the results during 10 runs of the GA are the same, STD equals 0.

The average saving in the number of terms for multi stage GA is 49 % for MPRM using “don’t care” compared with MPRM without using “don’t care” terms. Although some authors [16, 24, 33, and 36] have implemented algorithms for minimisation of MPRM for incompletely specified Boolean function, no benchmark results have been published that could be compared to the result presented within this Chapter.

By testing a number of examples using single and multi stage GA, it was found that both algorithms have similar results but the time required for the GA with two stages is much less as compared with one stage GA as shown in Table 5.15. For example, the multi stage GA took 2 seconds to find the optimal polarity for the benchmark example life compared to 16 hours using single stage GA. The other difference between the results of these two algorithms is the standard deviation. The standard deviation and average for the result of the single stage GA are much higher than multi stage GA especially when the function has large number of “don’t care” terms.

In reality, most functions have a significant number of “don’t care” terms (reach 1000 or more). Therefore, it is recommended using the multi stage GA to reduce the time taken and to produce good results no matter how large the number of “don’t care” terms. The multi stage GA algorithm for incompletely specified functions was tested with benchmark functions and the tests show better results (average saving 49%) are achieved when “don’t care” terms are taken into account in the attempted examples.

Table 5.15: Comparison between single and multi stage GA

Name	N.DC	Multi Stage GA			Single stage GA		
		M. terms	Time	AV/STD	M. terms	Time	AV/STD
Xor5	16	1	1 sec.	1/0	1	1 sec.	1 / 0
Rd53	54	6	1 min.	10.1/ 1.3	6	12.30 min.	11.8/2.4
Rd73	63	63	2 sec.	63 / 0	63	5 min.	72.8/ 7.1
Squar5	32	23	1 sec.	24 /1.1	23	2 sec.	24.3/1.3 6
Sym10	187	64	1 min.	86/ 16.7	64	~ 22 hours	91.5/ 28.6
9sym	92	34	40 sec.	52 /8.9	36	~ 12 hours	56.6/18. 2
life	372	40	2 sec..	44/ 8.2	48	~ 16 hours	62/11.6
clip	80	182	8 sec.	182 / 0	188	~ 6 hours	209/12.8
Newtag	22	1	16 sec.	2 / 0.5	1	3 min.	4 / 1.2

5.7 Summary

New and efficient GA based approaches are presented to optimise the number of terms of MPRM/or MPDRM single and multi output completely and incompletely specified Boolean functions. The algorithm determines the allocation of don't care terms for the given function resulting in optimal RM expansions. These algorithms are implemented in C++ and fully tested using standard benchmarks. The fitness functions for the proposed GAs are implemented using the two different methods explained in the previous Chapters (2 & 3). The comparison is made between the proposed GAs with two different methods used in its fitness function. The results show that GA

using Minterm/Maxterm separation techniques required less time to produce optimal polarity. The two algorithms were running on the same hardware and same operating system configuration.

For completely specified functions, the overall results show that the GA finds good solutions in a short time compared with the long CPU time required for running exhaustive search especially for large functions. Therefore, the GA can save time and effort to find the optimal mixed polarity without the need to find all the 3^n polarities for the specified function.

For incompletely specified multi-output functions, the process of optimisation of the RM is computationally hard problem, because of the large search space ($3^n \times 2^m$) which increases with number of variables and “don’t care” terms. The problem is further complicated when different outputs have different “don’t care” terms. Firstly, single stage GA was implemented to find the optimal polarity with minimal term for incompletely specified function. It was found that for functions with large number of variables and “don’t care” terms, the number of possible solutions is vast and requiring long time although can’t always guarantee good result.

Therefore, the proposed GA splits into two stages. The first stage produces best individuals without including “don’t care” terms. The second stage deduces the optimal selection for the “don’t care” terms which minimize the number of terms for the individuals produced in the first stage. Then it was found that multi stage GA is more efficient in terms of time and average of results especially for large numbers of variables and don’t cares.

Chapter Six

Optimal State assignment Using Multi-Objective Genetic Algorithm

6.1 Introduction

One of the very important roles in the design of the sequential circuits is the optimisation of the Finite State Machine (FSM). The FSM optimisation has considerable effects on performance of the designed circuit such as power dissipation, area, delay and testabilities of the sequential circuits.

This Chapter describes a Genetic Algorithm (GA) that utilises a Pareto Ranking scheme [78] to find state assignments that minimise both the hardware and power dissipation of the state machine. The Multi Objective Genetic Algorithm (MOGA) [67] is employed to find assignments that reduce both the hardware and power dissipation due to switching activity and leaves it to the designer to give the priority to either power dissipation or logic complexity or select a compromise solution that reduces both but not guarantee absolute minimum in either.

The remainder of this Chapter is structured as follows; section 6.2 defines the state assignment for the sequential circuits. Section 6.3 explains the multi-objective GA. The proposed algorithm is described in section 6.4. Experimental results are given in sections 6.5.

6.2 State Assignments for Sequential Circuits

As previously explained in Chapter 1, sequential circuits are combinational circuits in conjunction with memory storage devices and feedback as in Figure 1.1. The outputs of sequential circuits depend on both present and

past inputs. A FSM represents the transitions between states within the circuits which determine the behaviour of the sequential circuit. The designing of a synchronous sequential circuit can be established by representing the states using binary code for state variables. Synthesis tools are required to give each state a specific binary code. The state assignment refers to the allocations of the binary codes to the present and next states of the sequential circuits. The resulting combinational logic depends on the codes assigned to the states. The inputs with the present states represent the input of the combinational part, while the next states and the outputs represent the outputs of the combinational part of the specified sequential circuits [30, 79]. The total number of different possible assignments for k number of states and s state variables can be calculated by equation (1.29), while equation (1.30) represents the total number of unique state assignments. Table 6.1 shows total and unique number of possible state assignments for a different number of states. It is clear that the size of the solution space increases factorially with the number of states.

Table 6.1: Total and unique number of possible state assignments

k (no. of rows in STT)	s (state variable)	L (as in eq. (1.29))	A (as in eq. (1.30))
4	2	24	12
5	3	6720	140
6	3	20160	420
8	3	40320	840
9	4	451,347,200	10,810,800
10	4	29059430400	75,675,600
14	4	10,461,394,944,000	27,243,216,000
16	4	20,922,789,888,000	54,486,432,000

Furthermore; the optimal encoding for a FSM with 14 states or more can't be found using exhaustive search due to the excessive time required because of the large number of possible state assignments.

The power consumption [64] of a sequential circuit is proportional to its switching activity which can be represented by equation (6.1)

$$P_{ave} = \frac{1}{2} C_L V_{dd}^2 f_{clk} E_{sw} \quad (6.1)$$

where C_L is the physical capacitance of the output for the node, V_{dd} is the supply voltage, E_{sw} is the expected switching activity, and f_{clk} is the clock frequency. Since the register capacitance is fixed and cannot be affected, therefore; we consider the switching activity E_{sw} as cost function C which is one of the proposed objectives [64].

$$C = \sum_{i,j \in S} tp_{(i \leftrightarrow j)} HD_{i,j} \quad (6.2)$$

where $HD_{i,j}$ represents the Hamming Distance between the coding of the two states s_i and s_j , and $tp_{(i \leftrightarrow j)} = tp_{ij} + tp_{ji}$ and tp_{ij} is defined as the total state transition probability from states s_i to states s_j .

The Hamming Distance (HD) [80] between two Boolean vectors a_i, b_i is defined by the number of bits in same position a_i, b_i with different phases as in equation (6.3).

$$HD_{a,b} = \sum_{i=0}^{i=k-1} a_i \oplus b_i \quad (6.3)$$

Therefore; the number of state bits that are altered in every FSM transition is known as switching cost C (equation (6.2)). State assignments that result in a lower C value and a lower number of terms to structure the combinational circuit are considered to be optimal assignments. The switching activity and logic complexity of sequential circuits heavily

depend on the code assigned to the states which is influenced by the HD between states.

The total state transition probability [64] tp_{ij} between two states s_i & s_j , is defined as the probability that the transition from s_i to s_j , occurs in an arbitrary sequence and can be calculated using equation (6.4).

$$tp_{ij} = P_i p_{ij} \quad (6.4)$$

Steady state probability [64] P_i of state s_i is defined as the probability that the state is visited within an arbitrary random sequence.

$$p_{ij} = \text{Prob} (\text{Next} = s_j | \text{Present} = s_i) = \frac{N_{ij}}{\sum_h N_{ih}} \quad (6.5)$$

where p_{ij} represents the conditional state transition probability, N_{ij} is the number of transitions from s_i to s_j , $i, j \in \{0, 1, 2, 3, \dots, k-1\}$, where k is the number of states and $\sum_h N_{ih}$ are all transitions that begin with state s_i .

A finite state Markov process is defined as “a system that can be in one of several (numbered) states, and can pass from one state to another each time step according to fixed probabilities [81]”. The Markov system can be represented by State Transition Graph (STG) where all the states and transitions probabilities are shown. The transition matrix associated with this system is the matrix whose ij^{th} entry is the conditional state transition probability p_{ij} [81].

$$\sum_{i=0}^{i=k-1} P_i = 1 \quad (6.6)$$

$$P_i = \sum_{j=0}^{j=k-1} P_j p_{ji} \quad (6.7)$$

The steady state probabilities P_i can be calculated by solving the set of linear equations (6.6) and (6.7) using Gaussian elimination methods.

The calculations in details of these probabilities are shown in Example 6.1.

Example 6.1: Consider the benchmark Lion [72] which has two inputs, one output and four states. State Transition Graph (STG) for this example is shown in Figure 6.1.

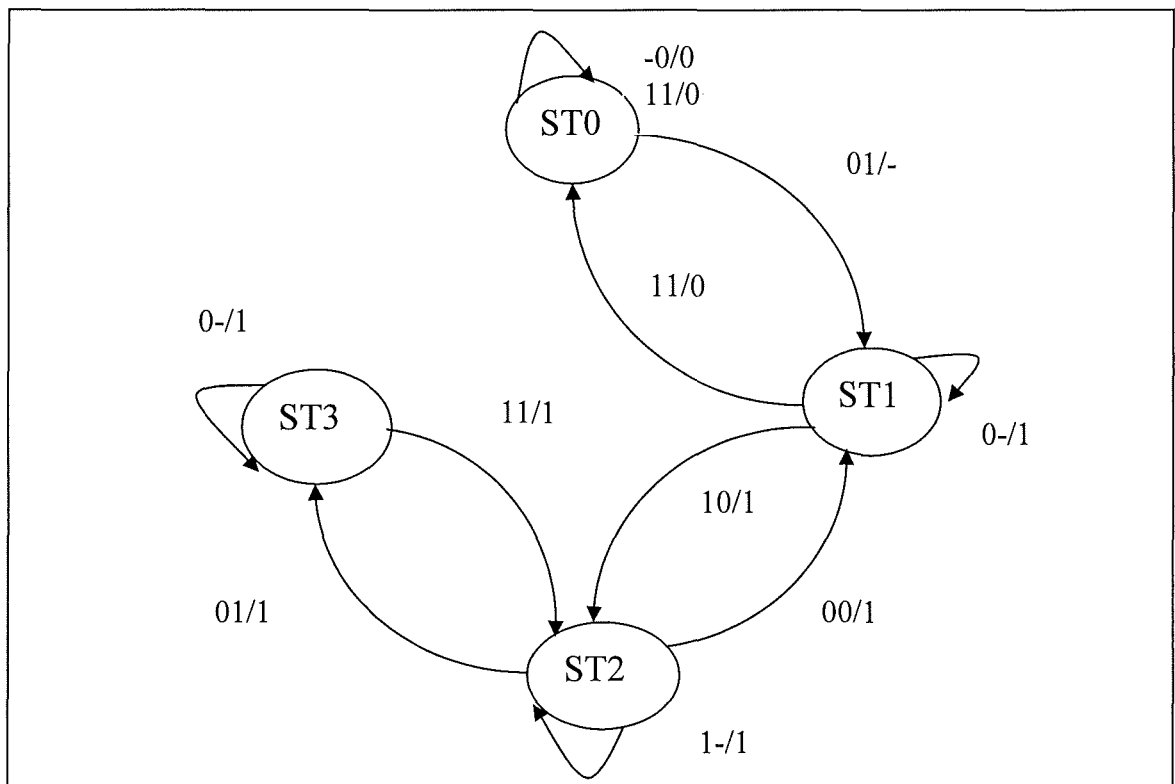


Figure 6.1: State Transition Graph of Lion Benchmark

The State Transition Table (STT) for this example shown in Table 6.2 consists of four symbolically encoded states $ST0$, $ST1$, $ST2$, and $ST3$. These states can be assigned unique codes using two state variables y_1 and y_2 . The inputs can be represented by x_1 and x_2 and the single output is represented by Z . The next states are represented by y_1^+ and y_2^+ .

Note that, if the next states and/or outputs are not specified for any state, the resulting circuit is known as an incompletely specified sequential circuit. These undefined transitions can be denoted by “-”.

Table 6.2: FSM - STT representation of Lion benchmark

Present states y_1, y_2	Next states $(y_1^+, y_2^+) / \text{Output } Z$			
	$(y_1^+, y_2^+) / Z$	$(y_1^+, y_2^+) / Z$	$(y_1^+, y_2^+) / Z$	$(y_1^+, y_2^+) / Z$
	x_1x_2 00	x_1x_2 01	x_1x_2 11	x_1x_2 10
ST0	ST0 / 0	ST1/-	ST0/0	ST0/0
ST1	ST1 / 1	ST1/1	ST0/0	ST2/1
ST2	ST1 / 1	ST3/1	ST2/1	ST2/1
ST3	ST3 / 1	ST3/1	ST2/1	-/-

The conditional state transition probability p_{ij} is calculated by dividing the number of inputs causing this transition by the total number of valid inputs at state s_i as in equation (6.5). The details of these calculations are shown below.

$$p_{11} = \text{Prob} (\text{Next} = s_1 | \text{Present} = s_1) = \frac{N_{11}}{N_{11} + N_{12} + N_{10}}$$

$$p_{11} = \frac{2}{2+1+1} = 0.5$$

$$p_{12} = \text{Prob} (\text{Next} = s_2 | \text{Present} = s_1) = \frac{N_{12}}{N_{11} + N_{12} + N_{10}}$$

$$p_{12} = \frac{1}{2+1+1} = 0.25$$

$$p_{13} = \text{Prob} (\text{Next} = s_3 | \text{Present} = s_1) = \frac{N_{13}}{N_{11} + N_{12} + N_{10}}$$

$$p_{13} = \frac{1}{2+1+1} = 0.25$$

Then all other conditional probabilities can be calculated using the same equations and the values of these probabilities are shown in Figure 6.2.

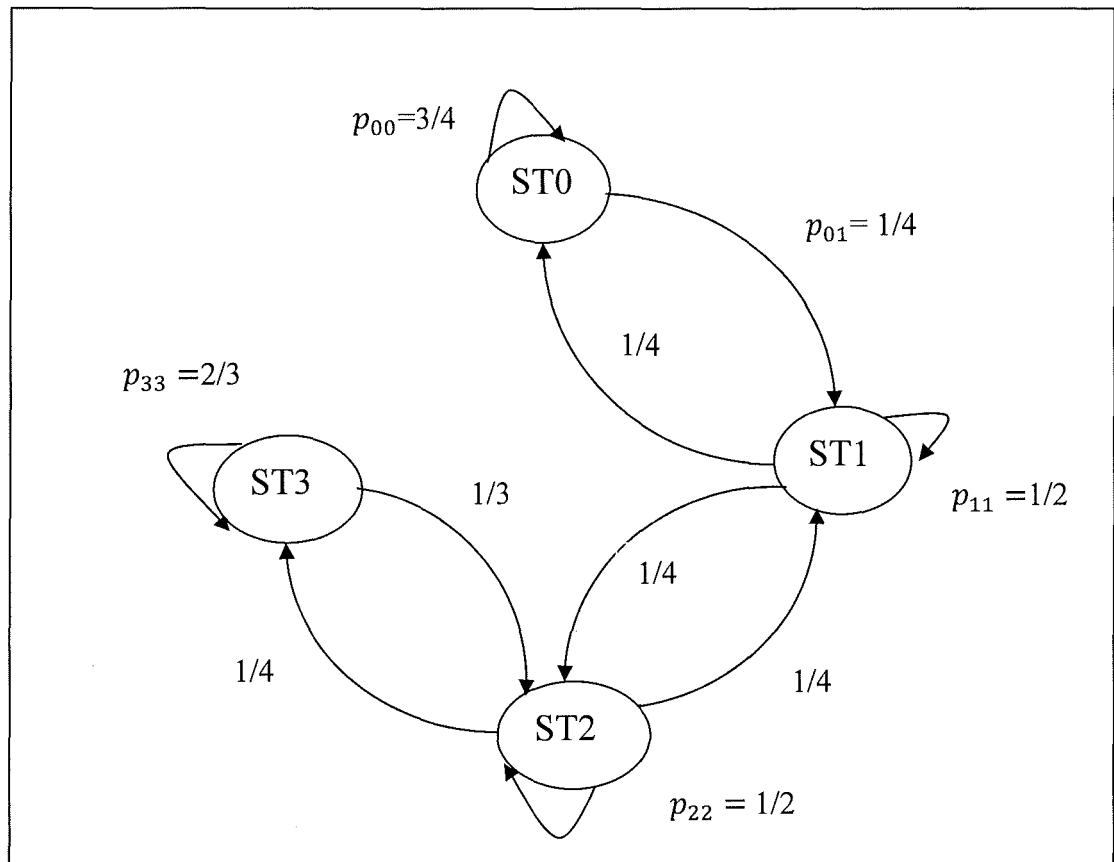


Figure 6.2: Conditional State Probabilities of Lion Benchmark

These conditional state probabilities which represent the ij^{th} entry of 4×4 transition matrix for this example are shown below:

$$\begin{bmatrix} \frac{3}{4} & \frac{1}{4} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{3} & \frac{2}{3} \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

From equations (6.6) and (6.7), steady state probabilities P_i can be calculated as follows:

$$P_0 = \frac{3}{4} P_0 + \frac{1}{4} P_1$$

$$P_1 = \frac{1}{4} P_0 + \frac{1}{2} P_1 + \frac{1}{4} P_2$$

$$P_2 = \frac{1}{4} P_1 + \frac{1}{2} P_2 + \frac{1}{4} P_3$$

$$P_3 = \frac{1}{3} P_2 + \frac{2}{3} P_3$$

$$P_0 + P_1 + P_2 + P_3 = 1$$

Using Gaussian elimination method to solve these five equations and find the steady state probabilities P_i , results in the following:

$$\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} 0.266667 \\ 0.266667 \\ 0.266667 \\ 0.2 \end{bmatrix}$$

The Cost C as function of switching activity for this example can be calculated as follows:

$$C = HD_{s_0,s_1} * (tp_{01} + tp_{10}) + HD_{s_1,s_2} * (tp_{12} + tp_{21}) + HD_{s_2,s_3} * (tp_{23} + tp_{32})$$

Two different random assignments are displayed in Table 6.3.

Table 6.3: Random assignments for Example 6.1

States	1 st assignment	2 nd assignment
ST0	00	10
ST1	01	01
ST2	11	11
ST3	10	00

For the first assignment, the Cost C as function of switching activity is:

$$\begin{aligned}
 C &= HD_{s_0,s_1}(tp_{01} + tp_{10}) + HD_{s_1,s_2}(tp_{12} + tp_{21}) + HD_{s_2,s_3}(tp_{23} + tp_{32}) \\
 &= HD_{s_0,s_1}(P_0p_{01} + P_1p_{10}) + HD_{s_1,s_2}(P_1p_{12} + P_2p_{21}) + HD_{s_2,s_3}(P_2p_{23} + P_3p_{32}) \\
 &= 1(0.06667 + 0.06667) + 1(0.06667 + 0.06667) + 1(0.06667 + 0.06666) \\
 &= 0.4
 \end{aligned}$$

The Lion benchmark in Kiss2 format is shown in Table 6.4.

Table 6.4: Kiss2 format of Lion Benchmark

.i 2			
.o 1			
.p 11			
.s 4			
-0	ST0	ST0	0
11	ST0	ST0	0
01	ST0	ST1	-
0-	ST1	ST1	1
11	ST1	ST0	0
10	ST1	ST2	1
1-	ST2	ST2	1
00	ST2	ST1	1
01	ST2	ST3	1
0-	ST3	ST3	1
11	ST3	ST2	1

The following file (PLA format) is produced by giving the present and next state the code shown in the first random assignment (Table 6.3) to be ready for minimisation using Espresso [73].

```
.i 4 // Primary inputs and present states
.o 3 // Primary outputs and next states
.p 11 // Number of product terms
-000 000          1-11 111
1100 000          0011 011
0100 01-          0111 101
0-01 011          0-10 101
1101 000          1110 111
1001 111          .e
```

After minimisation using Espresso, 1st assignment results in the following:

```
.i 4
.o 3
.p 6
10-1 100          -0-1 011
111- 010          -11- 101
0-10 101          .e
010- 011
```

Considering the sharing of terms, this implementation requires 6 terms. The equations for this circuit after minimisation are as follows:

$$y_1^+ = x_1 \bar{x}_2 y_2 + \bar{x}_1 y_1 \bar{y}_2 + x_2 y_1$$

$$y_2^+ = x_1 x_2 y_1 + \bar{x}_1 x_2 \bar{y}_1 + \bar{x}_2 y_2$$

$$Z = \bar{x}_1 y_1 \bar{y}_2 + \bar{x}_1 x_2 \bar{y}_1 + \bar{x}_2 y_2 + x_2 y_1$$

For the second assignment, the Cost C as function of switching activity is:

$$C = HD_{s_0,s_1}(tp_{01} + tp_{10}) + HD_{s_1,s_2}(tp_{12} + tp_{21}) + HD_{s_2,s_3}(tp_{23} + tp_{32})$$

$$= 2 (0.06667 + 0.06667) + 1 (0.06667 + 0.06667) + 2 (0.06667 + 0.06666)$$

$$= 0.666$$

The following file (PLA format) is produced by giving the present and next state the code of the second random assignment (Table 6.3) to be ready for minimisation using Espresso [73]


```
.i 4 // Primary inputs and present states
.o 3 // Primary outputs and next states
.p 11 // Number of product terms
-010 100 1-11 111
1110 100 0011 011
0110 01- 0111 001
0-01 011 0-00 001
1101 100 1100 111
1001 111 .e
```

After minimisation using Espresso, 2nd assignment results in the following:

```
.i 4
.o 3
.p 10
0110 010 1--1 100
-010 100 1-11 011
1100 111 0--1 001
0-01 010 -0-1 011
1-1- 100
0-0- 001
```

The equations for this circuit after minimisation are as follows:

$$y_1^+ = \bar{x}_2 y_1 \bar{y}_2 + x_1 x_2 \bar{y}_1 \bar{y}_2 + x_1 y_1 + x_1 y_2$$

$$y_2^+ = \bar{x}_1 x_2 y_1 \bar{y}_2 + x_1 x_2 \bar{y}_1 \bar{y}_2 + \bar{x}_1 \bar{y}_1 y_2 + x_1 y_1 y_2 + \bar{x}_2 y_2$$

$$Z = x_1 x_2 \bar{y}_1 \bar{y}_2 + \bar{x}_1 \bar{y}_1 + x_1 y_1 y_2 + \bar{x}_1 y_2 + \bar{x}_2 y_2$$

Considering the sharing of terms, this implementation requires 10 terms. Therefore; the first assignment is better in both objectives for this example. It is clear that the number of terms and switching activity of synthesised circuits has been affected by the FSM state assignment. The assignments used in this example were randomly selected. Furthermore, finding optimal state assignments for circuits with a large number of states becomes computationally complex as well as crucial for larger FSMs. Therefore; GA

will be used to find good assignment that reduces the logic and power in reasonable time without the need to do an exhaustive search.

6.3 Multi-Objective Genetic Algorithm

Single objective optimisation seeks to find the optimal (highest or lowest) value of the defined objective. For many problems, there is a need for simultaneous optimisation of several, possibly conflicting objectives. Therefore, if there are two objectives to be optimised, it may be possible to find two solutions; one of these solutions being optimal in terms of the first objective while the other is the optimal for the second objective [82, 83].

Multi-objective GA's may be applied to many complex engineering optimisation problems. A number of different evolutionary algorithms were suggested to solve multi-objective optimisation problems [84, 85].

In this research, there are two objectives to be optimised. Using a Pareto scheme [77, 82], it is convenient to classify all the potential solutions into dominated and non-dominated (Pareto optimal set) solutions. "The solution \vec{u} is dominated if there is a feasible solution \vec{v} not worse than \vec{u} for all objectives f_ω ($\omega = 1, \dots, r$), where r is the total number of objectives [83]". This could be expressed in mathematical form by equation (6.8)

$$f_\omega(\vec{u}) \leq f_\omega(\vec{v}) \text{ for all } 1 \leq \omega \leq r \quad (6.8)$$

"If a solution is not dominated by any other feasible solution in at least one objective, we call them non-dominated (or Pareto optimal set) solutions [82]".

Therefore, Pareto dominance normally requires that a solution \vec{u} dominates a solution \vec{v} if \vec{u} is better than \vec{v} in at least one objective and \vec{u} is as same as \vec{v} in all other objectives.

There is another approach for multi objective optimisation using objective weighting (aggregating function) [82, 83], which combines the multiple objectives into a single composite function using the weighted sum method. The weight w for each objective is ($0 < w_{\omega} < 1$), $\sum_1^r w_{\omega} = 1$, and different weight vectors lead to different solutions. The problem becomes one of finding the solution which minimizes $\sum_1^r w_{\omega} f_{\omega}$. The advantages of this method are, its simplicity, systematic structure of objectives, and it may work properly with few objective functions. Normally, this method produces one single solution. The drawback being the difficulty in setting the weights that can scale the objectives for the problem to suit the requirements which could be different from one problem to other [82]. The designer will need to have prior knowledge of the weight value for each objective and to run the algorithm many times to find one of the different solutions each time. For instance, not all people involved in decision making, agree on relative importance of parameters. Therefore, multiple run of aggregating function will be required to offer the designer different solutions although can't guarantee that the produced solutions will cover all Pareto- optimal solutions.

The proposed algorithm produces a set of optimal solutions (known as Pareto-optimal solutions), instead of a single optimal solution. Each one of these Pareto-optimal solutions has different diversity. Without knowing what the user requirements are, it cannot be said that any one of these solutions is better than the other. Therefore, the proposed algorithm has the ability to find multiple Pareto-optimal solutions in one single simulation run. The proposed technique is one of the most popular evolutionary multi objective optimisation techniques. The main advantages are its simplicity

to implement and the overall good performance. The weakness of this technique is the controlling of the diversity of population [82].

The proposed MOGA in this Chapter has two objectives, the first being to reduce the number of components required to design the combinational part of the sequential circuit. The second is to reduce the switching activities.

6.4 Proposed Algorithm

The MOGA is proposed to optimise the state assignment for completely and incompletely specified sequential circuits without doing an exhaustive search. The aim is to identify the good state assignments which can be used to design the circuit with fewer components and reduced switching activity simultaneously. The MOGA algorithm is implemented in C++, and tested with 15 benchmark examples [72] of up to 48 states. The search space of the proposed algorithm is defined by equation (1.29).

The proposed algorithm for finding the optimal state assignment represents a candidate solution using a chromosome containing the code for each state of the sequential circuit. Decimal numbers are used to represent each chromosome. The length of each chromosome is equal to 2^s , where s is number of state variables. Each gene in the chromosome holds the decimal code for the states used including the “don’t care” states.

In MOGA, non-dominated individuals are assigned rank 1. Therefore, in every population, there is at least one individual with rank 1 and the maximum rank assigned to the individuals within population will be equal or less than population size.

Example 6.2: a circuit with 6 states; requires 3 flip flops. Therefore; this circuit has two don't care (DC) states. The length of the chromosomes = $2^3=8$ bits. If the chromosome is | 3 4 2 1 0 6 5 7 |, then the state assignment is shown in Table 6.5.

Table 6.5: One possible state assignment for Example 6.2

States	Chromosome	assignment
ST0	3	011
ST1	4	100
ST2	2	010
ST3	1	001
ST4	0	000
ST5	6	110
(DC) ST6	5	101
(DC) ST7	7	111

The MOGA with the two objectives has two fitness functions. The first fitness function “*Fitness_term (c_i)*” calls the Espresso tool [73] to minimise the combinational part of the circuit and to produce the terms for the minimised circuit. The second fitness function “*Fitness_switching (c_i)*” calculates the switching activity as given by equation (6.2).

The Pareto ranking is integrated into the proposed algorithm by replacing the chromosome fitness by the Pareto ranks. This scheme is based on several layers of classifications [82]. All non dominated solutions are given rank one. Figure 6.3 gives the pseudo code of the proposed algorithm.

The GA uses a tournament selection method where the main parameter of selection is the tournament size (*T*) which can be changed by the operator.

```
Procedure MOGA ( )
{
Input Parameters of GA (benchmark file, population size,
                        Tournament size, & number of Evaluations)
Read_Terms (benchmark)
Randomly_Initialise_population( )
Fitness_terms(all population)
Fitness_Switching(all population)
Set_Rank (all population)

Loop until (Number of Evaluations = 0)
{
Tournament Select (T) //select two parents
Crossover (Child ) //Uniform crossover to produce the Child
Mutation(Child ) //Swap two gene randomly in the new Child
Fitness_terms (Child ) // Calculate Number of terms
Fitness_Switching (Child) // Calculate Switching Activity
If (Child Fitness!=any existing Fitness)
//!= denotes not equal
{
Tournament Replacement ( )
}
Set_Rank( );
Number of Evaluations = Number of Evaluations - 1
}

Output Results ( )
} End MOGA

Set_Rank ( )
{
Current_Rank=1;
All=pop_size;
Loop For (i =0 to i=pop_size)
{
If (NonDominated (i, All))
{
Rank[i] =Current_Rank;
}
Remove (i);
All = All-1;
Current_Rank = Current_Rank+1;
}
}End Set_Rank
```

Figure 6.3: Pseudo code for the Proposed MOGA

A number of individuals (T) are selected from the population randomly and the one with the smaller rank (i.e. best rank) is then used as the selected individual. Crossover is the principle genetic operator. Uniform crossover shown in Figure 6.4 is adopted. A string of binary bits is initialized by the proposed algorithm randomly. The length of the binary string is equal to the length of the chromosome which determines whether the genes are copied from the first parent or from the second parent.

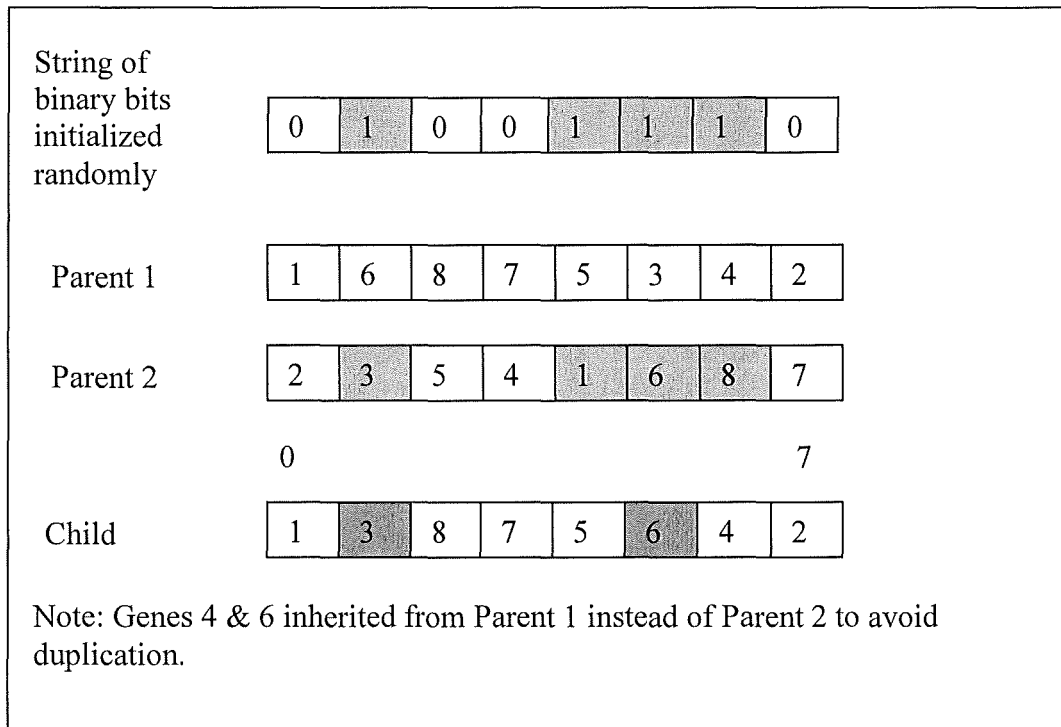


Figure 6.4: Uniform crossover used in the proposed algorithm

The child inherits the gene from the first parent if the corresponding bit in the binary string is zero while the child inherits the gene from the second parent if the corresponding bit in the binary string is one. A continuous check is required before the inheritance for each gene avoiding the repetition of the same coding for different states which is not allowed. The mutation operator swaps the positions of two randomly chosen genes as shown in Figure 6.5.

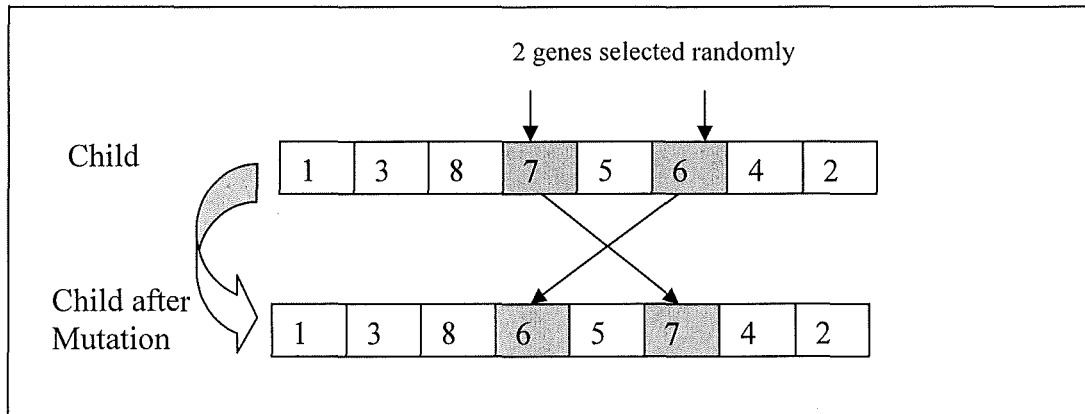


Figure 6.5: Mutation used in the proposed algorithm

A replacement strategy controls the composition of the new generation for each evolution loop. The proposed algorithm uses a tournament replacement method, which randomly, chooses T individuals (independently of their ranks) from the population and replaces the chromosome which has the worst rank with the new child.

The successful application of GA depends on the diversity of the whole population in the search space. It may be difficult for a GA to find the global optimum solution, if it can't hold its diversity well; this may result in the premature convergence to a localised optimal solution.

Premature Convergence is one of the major problems associated with GAs. Losing of genetic variation could results in Premature Convergence which may happen after a number of evaluations, when all chromosomes within the population have the same number of terms and same switching activity. In this case, the algorithm will not be able to generate child with the aid of genetic operators from the selected parental solutions due to this problem. To avoid this problem, which is one of the biggest problems of GA, criteria

is used by defining a condition for the replacement. Which is in case that any one of the current chromosomes has the same rank as that of the new child, the new child will not be replaced.

For the halting criteria, a usual strategy is to stop evolution after a fixed number of evaluations, which is determined by the user.

Example 6.3: Consider the FSM-STT for the benchmark bbtas [72] which has six states as shown in Table 6.6.

Table 6.6: FSM - STT representation of bbtas Benchmark

Present states y_1, y_2	Next states $(y_1^+, y_2^+)/$ Outputs Z_1Z_2			
	$(y_1^+, y_2^+)/ Z_1Z_2$	$(y_1^+, y_2^+)/ Z_1Z_2$	$(y_1^+, y_2^+)/ Z_1Z_2$	$(y_1^+, y_2^+)/ Z_1Z_2$
	x_1x_2 00	x_1x_2 01	x_1x_2 11	x_1x_2 10
ST0	ST0/00	ST1/00	ST1/00	ST1/00
ST1	ST0/00	ST2/00	ST2/00	ST2/00
ST2	ST1/00	ST3/00	ST3/00	ST3/00
ST3	ST4/00	ST3/01	ST3/11	ST3/10
ST4	ST5/00	ST4/00	ST4/00	ST4/00
ST5	ST0/00	ST5/00	ST5/00	ST5/00

Figure 6.6 shows numbers of terms and switching activities for 17 different assignments randomly initialised. It is clear that some assignments produce the same number of terms with different switching activity, such as assignments 4, 5, 7 and 12.

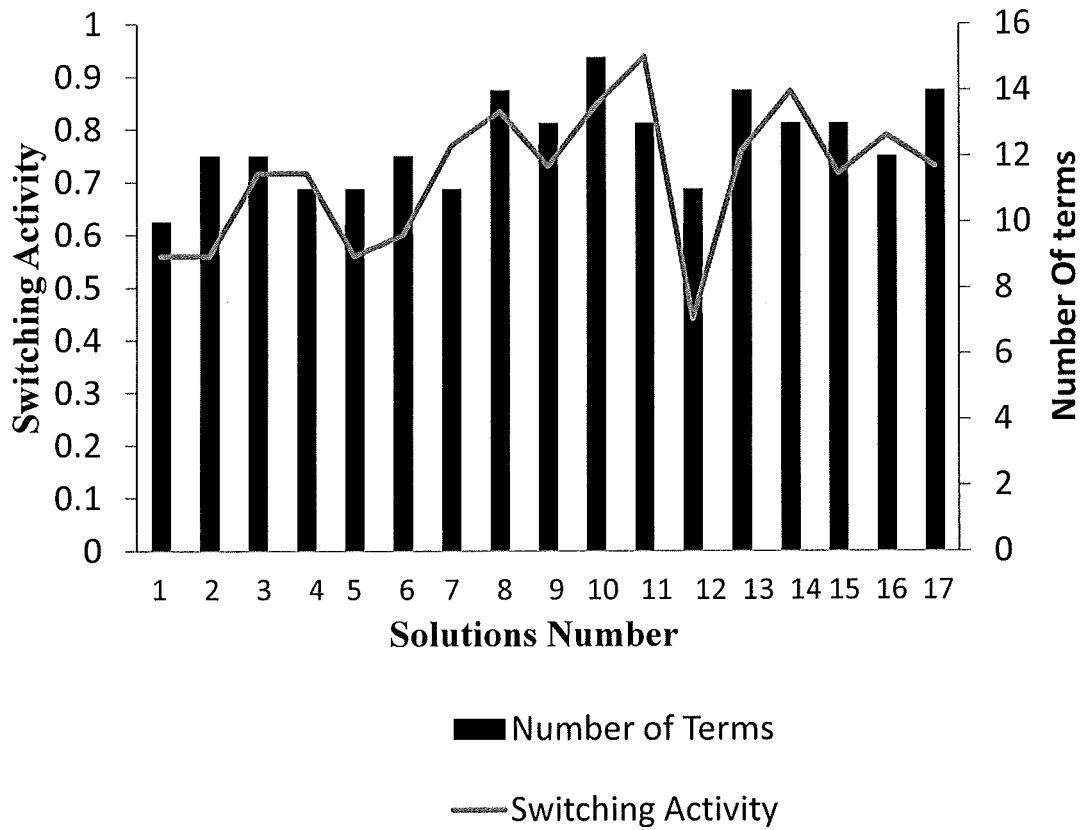


Figure 6.6: Terms and Switching activities for different assignments of Example 6.3.

Table 6.7 shows the codes and ranks for these different assignments shown in Figure 6.6.

Table 6.7 Codes and ranks for different state assignments for Example 6.3

Solution Number as in Fig. (6.6)	Codes for different state assignments		Terms	Switching Activity	Ranks
	st ₇	st ₀			
1	0 5 3 1 4 2 6 7		10	0.56	1
2	1 4 5 6 0 2 3 7		12	0.56	3
3	1 0 6 4 3 2 7 5		12	0.717	5
4	4 2 0 3 6 7 1 5		11	0.717	3
5	3 1 7 4 5 0 2 6		11	0.56	2
6	5 7 0 1 3 2 4 6		12	0.6	4
7	0 5 3 1 6 4 2 7		11	0.769	4
8	6 4 5 1 7 0 3 2		14	0.834	10
9	4 2 5 6 7 1 3 0		13	0.73	7
10	2 7 0 3 4 1 5 6		15	0.847	11
11	4 7 6 1 0 3 5 2		13	0.939	9
12	0 2 4 5 1 3 7 6		11	0.44	1
13	1 0 2 4 7 5 6 3		14	0.76	9
14	4 0 7 2 5 6 1 3		13	0.873	8
15	0 6 3 2 1 4 7 5		13	0.717	6
16	4 2 0 5 6 1 3 7		12	0.79	6
17	4 7 6 5 1 2 3 0		14	0.7304	8

The first five ranks are shown in Figure 6.7. Solutions which are the optimal either in number of terms, switching activity or both have rank of one.

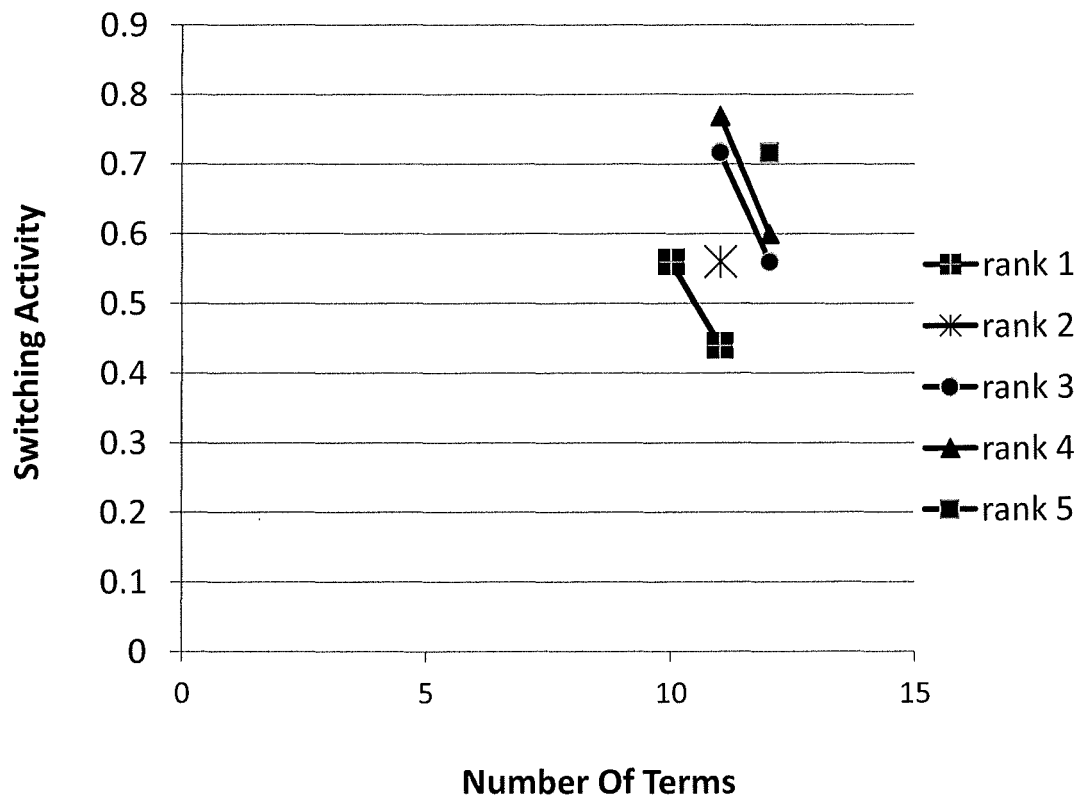


Figure 6.7: Different ranks for Example 6.3

It is obvious from Figure 6.7, that each rank has different number of solution depending on values of switching activity and number of terms for each solution of this running, such as rank1, 3, and 4 has two solutions while rank 2 and 5 has one solution only. This Figure shows that one solution having rank 1 is better than the solution of rank 2 in term of number of term while the other solution of rank 1 is better than the solution of rank 2 in term of switching activity. The solution having rank 2 is better than both solutions of rank 3 in term of switching activity and number of terms. The values of number of terms and switching activity for these 5 ranks are all shown in Table 6.7.

After running the proposed algorithm, it produces three results with rank of one as below.

0	5	3	1	4	2	6	7	terms = 10	switching activities =0.56
6	1	5	4	0	3	2	7	terms = 9	switching activities =0.613
0	2	4	5	1	3	7	6	terms = 11	switching activities =0.44

The user has the choice to select one of these results depending on requirements. The time required to produce these solutions is one minute only. The results are obtained using population size=30, tournament size=3 and 300 for the number of evaluations. These parameters are determined after testing various population sizes and different tournament sizes.

6.5 Experimental Results

The proposed algorithm is applied to 15 MCNC benchmark [72] functions. The algorithm is implemented using C++ and is tested using a PC with INTEL CPU, 2.4 GHz clock and 2GB RAM. Test results are given in column three of Table 6.8. Espresso is used to minimise the circuit for each state assignment. The second column in Table 6.8 denotes the number of inputs, number of output and number of states for the benchmark given in the first column. The set of results produced by the MOGA is given in column 3. *PT* denotes to number of product terms and *C* refers to the cost as function of switching activity. It is obvious that the number of solutions produced is different from one example to another and depends on how many non dominated solutions having rank one are produced by the proposed algorithm.

Table 6.8: Experimental results for the proposed MOGA

Benchmarks	In/out/ No. of states	Results of MOGA		Result of NOVA[87]		Results of Ref. [53]		Results of Ref.[54]		Saving compared to Ref. [87]		Saving compared to Ref[53]		Saving compared to Ref[54]	Time
		PT	C	PT	C	PT	C	PT	PT	C	PT	C			
bbtas	2/2/6	9	0.613	8	0.815	---	---	9	-11%	25%	---	---	0%	1 min.	
		10	0.56						-27%	46%					
		11	0.44												
bbara	4/2/10	22	0.49	24	0.459	22	0.317	23	8%	-6%	0%	-	4%	8 min.	
		27	0.39						-11%	15%	-18%	24% -20%			
opus	5/6/10	15	0.49	16	0.809	15	0.556	12	6%	40%	0%	12%	-20%	40 min.	
		17	0.49						0%	40%		-25%			12%
		16	0.488												
Lion9	2/1/9	10	0.34	-	--	---	---	11	--	--	---	---	9%	8 min.	
Dk16	1/2/27	57	2.1	-	--	---	---	68	--	--	---	---	16%	6 hours & 3 min.	
		68	1.64												
		59	1.7												
keyb	7/2/19	46	0.98	48	1.469	46	0.674	46	4%	33%	0%	-31%	0%	3 hours & 32 min.	
		47	0.75						-12%	63%		-16%			20%
		55	0.54												
Cse	7/7/16	43	0.39	46	0.602	43	0.355	45	7%	35%	0%	-9%	4%	3 hours & 9 min.	
		49	0.32						-15%	50%		-20%			15%
		54	0.30												

Table 6.8: Continued

Benchmarks	In/out/ No. of states	Results of MOGA		Result of NOVA[87]		Results of Ref. [53]		Results of Ref.[54]	Saving compared to Ref. [87]		Saving compared to Ref[53]		Saving compar ed to Ref[54]	Time
		PT	C	PT	C	PT	C		PT	C	PT	C		
donfile	2/1/24	22 26	1.375 1.29	28	1.75	36	1.6	31	21% 7%	21% 26%	39% 27%	14% 19%	29%	6 hours & 4 min.
Ex1	9/19/20	48 49 51	0.78 0.63 0.621	44	1.338	52	0.842	47	-8% -14%	42% 54%	8% 2%	7% 26%	-2%	6 hours & 7 min.
Ex4	6/9/14	13 14	0.568 0.468	19	1.310	14	0.421	15	32% 26%	57% 64%	7% 0%	-25% -10%	13%	6 hours &1 min.
Modulo 12	1/1/12	10 11	0.75 0.58	12	1.00	12	0.583	10	16% 8%	25% 42%	8% 8%	-22% 0%	0%	5 hours & 56 min.
S1	8/6/20	43 53 60	1.37 1.19 1.04	80	1.698	66	1.48	68	46% 25%	19% 39%	35% 9%	-7% 30%	37%	6 hours
S1a	8/6/20	29 30	1.21 1.174	--	--	---	---	66	--	--	---	---	56%	5hours &19 min
stry	9/10/30	78 79 88	1.1 0.93 0.674	94	1.278	88	0.943	78	17% 6%	14% 47%	11% 0%	-14% 29%	0%	6 hours & 5 min.
Planet	7/19/48	81 82 87	2.49 2.09 1.69	87	2.833	86	2.24	84	7% 0%	12% 40%	6% -1%	-10% 33%	4%	25 hours & 23 min.

In Table 6.8, the comparison is made between the results produced by the proposed algorithm and the results published by other references as shown in columns 4, 5 & 6.

The results are compared with NOVA tool [86] results, which were published in [87]. It is obvious that MOGA results for 9 out of 12 benchmark examples tested are better than NOVA results in terms of hardware components and switching activity. While for the other benchmark examples, MOGA results are better than NOVA in either number of terms or switching activity but not both. From Table 6.8, first set of MOGA results for all benchmarks tested, it can be seen that on average MOGA produces results requiring 21% fewer product terms and 15% less switching activity compared to NOVA.

The GA in [53] was developed for finding good assignment to minimise area and power for the FSM. The author combined the two objectives into a single composite function using the weighted sum method. Table 6.8 shows that the proposed algorithm compared to [53] can achieve more saving in terms of hardware components and switching activity for 6 out of 11 benchmark examples tested. While for the other 5 out of 11 benchmark examples tested, the MOGA results achieve more saving in either number of terms or switching activity but not both.

Reference [54] presented a GA for finding good assignment to reduce the area requirement. Comparing MOGA results and results obtained from this reference, it is found that the MOGA results could save terms in 8 out of 15 benchmark examples tested with reduction in the switching activities. It is also obvious that saving in terms becomes larger for the large functions, (56% in one case).

The time required to produce the good assignment is different for each example and depends on the complexity of the circuit. The time required by the proposed algorithm is large due to the fact that MOGA has to communicate with Espresso to minimize the logical expressions. For each evaluation of the GA, the proposed algorithm calls the Espresso to minimise the circuit for each assignment. Even allowing for this overhead, the time required to produce a good assignment is still acceptable. It is in the range of 1 minute for the circuit with 6 states to 25 hours for the circuit with 48 states.

Figure 6.8 shows the number of Evaluations for every benchmark tested in Table 6.8. Evaluations number denotes to how many evaluation required within the MOGA to produce the optimal state assignments. This Figure shows the efficiency of the proposed algorithm which is clear from the big difference of number of evaluations compared with total number of assignments. For example, considering Cse Benchmark which has 16 states, doing exhaustive search, the total number of unique state assignment is 54,486,432,000, while the proposed MOGA needs 60000 evaluations to produce good assignment. The number of evaluations actually used compared to the total number of unique state assignments gives an indication of the efficiency of the proposed MOGA.

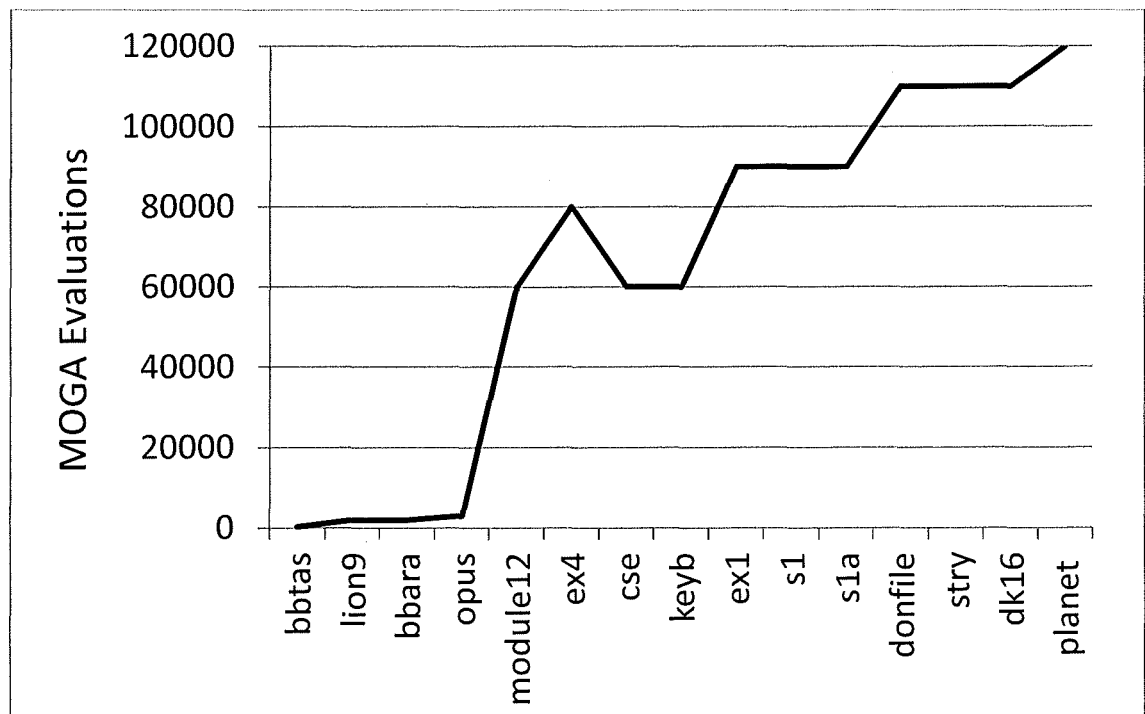


Figure 6.8: Number of MOGA Evaluations for different Benchmarks

6.6 Summary

In this Chapter, a new approach using a Multi Objective Genetic Algorithm is proposed to determine the optimal state assignment with less area and power dissipations for completely and incompletely specified sequential circuits. The goal is to find the assignments which reduce the component count and switching activity. The proposed MOGA employs a Pareto ranking scheme and produces a set of state assignments, which are optimal in both objectives. The Espresso tool is used to optimise the combinational parts of the sequential circuits.

Experimental results are given using a personal computer with an Intel CPU of 2.4 GHz and 2 GB RAM. The algorithm is implemented using C++ and fully tested with benchmark examples. The experimental results show that saving in components and switching activity are achieved in most of the benchmarks tested compared with recent publications.

Chapter 7

Conclusions and Future Work

The aim of this research is to develop various techniques and algorithms for logic synthesis and optimisation of combinational and sequential logic circuits. These algorithms could become part of new commercial ECAD package for future VLSI digital designs. All algorithms are implemented in C++ and fully tested using standard benchmark examples on a personal computer with an Intel CPU of 2.4 GHz and 2 GB RAM under Window XP professional compiled by using Bloodshed DevC++ platform.

7.1 Review of Algorithms and Techniques

The main contributions of this thesis are as follows:

In Chapter 2, new and simple techniques and algorithms called Extended_Tabular techniques are presented for bidirectional conversions between FPRM/FPDRM logic functions and MPRM/MPDRM respectively and to derive any mixed polarity from another MPRM for any number of variables. Therefore, two mixed polarities can be derived from each other without the need to go back to the original Boolean function in the CSOP form, thus saving memory and computer time. These techniques are based on a Tabular technique which has been developed to calculate the coefficients of FPRM expansions from CSOP expansion. These new techniques could be used for both single and multi-output Boolean functions. The implemented algorithms for these new techniques are tested with benchmark examples of up to 16 inputs and up to 31 outputs using exhaustive search. Time required to do exhaustive search reached up to 14 days for benchmark with 16 inputs and one output. The advantage of these techniques is its simplicity and there is no restriction of number of

variables. The experimental results for running exhaustive search (Tables 2.16 and 2.17) show that the optimum polarity among 3^n MPRM/or MPDRM is always same or better than the optimum polarity among 2^n FPRM/FPDRM expansions resulting in saving of components and power consumptions. The experimental results demonstrate that, in some cases, the circuit can be better minimised in DRM expansion using OR/EXNOR forms whereas for other circuits, the reverse is the case. Further, in other cases, the circuit can be better simplified in the standard CSOP Boolean function using Espresso tools. The average saving in components for the tested benchmarks compared with Espresso terms are 17% for MPRM and 27% for MPDRM. Moreover, the average saving in components for the tested benchmark compared with PPRM/PPDRM terms is 49% for MPRM and 41% for MPDRM.

Chapter 2 also shows a new technique developed for computing the coefficients of 3^{2^n-1} PKRO_RM expansions starting from PPRM expansion based on Extended_Tabular techniques. This technique could also be used to generate any polarity among PKRO-RM class from the CSOP Boolean function saving time and the effort required for converting from CSOP to FPRM. This technique can be used manually for small examples or programmed on a computer. This technique is tested manually with small examples.

In Chapter 3, fast and simple techniques called Minterm/Maxterm separation techniques are developed to generate MPRM/MPDRM starting directly from the truth vector of standard form CSOP/CPOS Boolean functions. These new techniques could also be used to compute the coefficients of MPRM/MPDRM starting from FPRM/FPDRM forms,

respectively. The advantages of these techniques are being able to process functions with any number of variables, their simplicity, and their efficiency in terms of CPU time. The other advantage of these techniques is being able to convert the standard form of Boolean functions directly to any MPRM/or MPDRM expansion without the need to convert it to FPRM/or FPDRM form. The experimental results (Tables 3.3 and 3.4) show that the time required running an exhaustive search to find optimum polarity among 3^n MPRM/MPDRM using the Minterm/Maxterm separation is less than the time required using the Extended_Tabular technique especially for large functions. For example benchmark with 15 input and 17 outputs required 14 days and 8 hours to run exhaustive search using Extended_Tabular technique while it required 5 days and 4 hours using Minterm/Maxterm separation technique.

New techniques and algorithms in Chapter 4 called RM_Minimisation techniques are presented to generate reduced RM expansions starting from PPRM/PPDRM for any number of variables for completely specified single and multi-output Boolean functions. The aim for developing these techniques is to minimise the number of terms/or sums. The proposed technique provides expressions with minimal terms/or sums among 3^{tn} ESOP/ENPOS expressions in which each term/or sum has its own polarity. This is achieved by deriving the polarity for each term/or sum instead of deriving polarity for each variable as in Tabular technique. These techniques are tested with different benchmark examples with inputs up to 16 and outputs up to 31 as shown in Tables (4.8) and (4.16).

The experimental results show that RM_Minimisation techniques produced good results within very short time searching a field of 3^{tn} ESOP/ENPOS expressions, although not all functions can minimise well using this technique depending on nature and complexity of the function. The results

show that in the RM domain, for 12 out of 18 tested benchmarks, the results produced are better or same as results doing an exhaustive search among 3^n MPRM expressions. Furthermore; results in DRM domain show that in 7 out of 14 tested benchmarks, the results produced are better or same as results obtained using exhaustive search among 3^n MPDRM expressions. For all tested benchmarks, these techniques achieved CPU time within one second or less even for large functions to provide these results.

The average saving in components for the tested benchmark compared with PPRM/PPDRM terms is 37.7% for both RM/DRM.

Genetic Algorithms (GAs) are presented in [Chapter 5](#) to produce the optimal polarity with a minimal number of terms/or sums among 3^n MPRM/or MPDRM expansions for single and multi-output completely specified Boolean functions. The GA can save time and effort when finding the optimal mixed polarity, thus avoiding the exhaustive search. The GAs for multi output functions are implemented in two different ways. The first is by assuming that the outputs can have different polarities while in the second all the outputs have the same polarity. The proposed GAs are tested with different benchmark examples with up to 16 inputs and up to 19 outputs (Tables 5.2, 5.4 and 5.5). The experimental results show that the GA results are the same as the optimum polarity running exhaustive search among 3^n MPRM/or MPDRM expansions, though this can't be guaranteed for other examples. Further, the GA time required to produce results is much less than CPU time required for running exhaustive search, especially for large functions.

For example, finding the optimal MPRM expansion for benchmark with 16 inputs and one output took 14 days using the Extended_Tabular technique, 5 hours & 12 minute using the Minterm separation technique, and 2.37

minute only using the proposed GA. More comparisons between other examples are shown in Table 5.6

The proposed GA is also tested with large functions for inputs up to 23 and outputs up to 46. The Experimental results (Table 5.8) for large functions are compared with results of reference [48] which used GA combined with simulated annealing. The results show that the proposed GA produces same or better results with 102 terms saving in total.

Testing GA for multi-output Boolean functions (Table 5.4), show that assigning different polarity to each output of the benchmark could achieve better result at the expense of longer CPU time.

Comparison of number of products was made between the proposed GA for MPRM and RM_Minimisation techniques (Table 5.7). It was found that in 12 out of 20 benchmark examples tested, same or less number of products are required to design the circuits in RM domain using RM_Minimisation technique. Further, comparison between the proposed GA for MPDRM and RM_Minimisation technique in the same table found in 8 out of 14 benchmark examples tested, RM_Minimisation technique produced same or less number of sums to design the circuits in DRM domain.

A multi stage GA is proposed also in Chapter 5 to produce the optimal polarity with minimal number of terms for incompletely specified Boolean functions. The proposed GA found the best choice of the “don’t-care” terms resulting in minimal MPRM expansions. For incompletely specified multi-output functions, the search space equals $(3^n \times 2^\mu)$ which is vast and increases with the μ number of “don’t care” terms for the specified

function. Therefore; the process of optimisation of the RM for incompletely specified Boolean function is computationally hard problem. The problem is further complicated when different outputs have different “don’t care” terms. The reasons for splitting GA into two different stages are to improve the performance and to reduce the computation time. The first stage conducts a polarity search to find the best MPRM expansions with fewer terms without considering the “don’t care” terms. In the second stage, different collections of “don’t care” terms are allocated to further minimise the circuits.

The proposed GA was tested with different benchmark examples up to 10 inputs and up to 8 outputs having different number of “don’t care” terms of up to 372 terms as shown in Table 5.15. The experimental results show better results (average saving 49%) are achieved when “don’t care” terms are taken into account in the attempted examples.

In Chapter 6, a Multi Objective Genetic algorithm (MOGA) approach to the state assignment problem is presented with the aim of minimising gate count and power dissipation for completely and incompletely specified sequential circuits. The target for this algorithm is to find the best assignments which require less number of components to design the combinational part of the sequential circuit, with reduced switching activity. The aim is therefore to minimise the power dissipation and the area, simultaneously. The Pareto ranking scheme has been integrated with the MOGA by creating a set of integral ranks for all chromosomes in the population which are used by the GA as fitness. The main advantage of using Pareto ranking scheme is , the proposed algorithm produces a set of Pareto-optimal solutions, instead of a single optimal solution. Therefore, the user has the ability to decide which one of these solutions is better to

use depending on the requirements. The proposed algorithm has the ability to find multiple Pareto-optimal solutions in one single simulation run. The other advantages are its simplicity to implement and the overall good performance.

The MOGA has been tested with different benchmarks for up to 9 inputs, 19 outputs and 48 states. The results are compared with NOVA tool [87] results, and the results published by other [53, 54]. The average percentage savings are 21% for number of terms and 15% for switching activity compared with NOVA [87]. The experimental results show that the proposed algorithm, compared to [53], can achieve more saving in terms of hardware components and switching activity for 6 out of 11 benchmark examples tested. While comparing MOGA results (Table 6.8) and results obtained from [54], it is found that MOGA results could save more cubes in 8 out of 15 benchmark examples tested.

The time required to produce the good assignment is different for each example and depends on the complexity of the circuit. The time required by the proposed algorithm is large due to the fact that MOGA has to communicate with Espresso to minimise the logical expressions. For each evaluation of the MOGA, the proposed algorithm calls the Espresso to minimise the circuit for each assignment. Even allowing for this overhead, the time required to produce a good assignment is still acceptable. It is in the range of 1 minute for the circuit with 6 states to 25 hours for the circuit with 48 states.

7.2 Future Works

The work which has been undertaken within this research may be continued along the following lines:

- The techniques presented in Chapter 2 for bidirectional conversion between MPRM/MPDRM and FPRM/FPDRM respectively can be further generalised for incompletely specified Boolean functions.
- Implement and test the technique presented in Chapter 2 for calculating the coefficients of PKRO_RM expansions.
- The Minterm/Maxterm separation techniques presented in Chapter 3 for calculating the coefficients of MPRM/or MPDRM expansions starting from the standard form of CSOP/or CPOS completely specified Boolean functions, respectively can be extended to incompletely specified Boolean functions. They can also be extended to multi output Boolean functions.
- Genetic Algorithm based approach is developed and implemented to find optimal polarity among 3^n MPRM/or MPDRM expansions for single and multi-output completely and incompletely specified Boolean functions. Another GA can be developed to find optimal polarity among 3^{2^n-1} PKRO_RM expansions.
- Develop technique for calculating the coefficients of 3^{th} polarities related to ESOP/ENPOS class which is the most general class of RM.
- All algorithms developed in this research can be collected together in a package for Computer Aided Synthesis and Optimisation of Electronic Logic circuits.

Publications

The following list shows the papers published during the research:

- [1] Al Jassani B.A., Urquhart N., & Almaini A.E.A., 2008. Optimization of MPRM functions using Tabular Techniques and Genetic Algorithms, *Mediterranean Journal of Electronic and Communications : MEDJEC*, **4**(4), pp.115-125.
- [2] Al Jassani B.A., Urquhart N., & Almaini A.E.A., 2009. Minimization of Incompletely Specified Mixed Polarity Reed Muller Functions using Genetic Algorithm, *IEEE 3rd International Conference on Signals, Circuits and Systems, Tunisia*
- [3] Al Jassani B.A., Urquhart N., & Almaini A.E.A., 2010. Manipulation and Optimisation techniques for Boolean logic, *IET computer and digital techniques* , **4**(3), pp. 227-239.
- [4] AL Jassani B. A., Urquhart N., & Almaini A.E.A., 2011. State assignment for Sequential Circuits using Multi-Objective Genetic Algorithm, *IET computer and digital techniques*, **5**(5), pp.296-305.
- [5] Al Jassani B.A. & Almaini A.E.A., 2011. Computing the Pseudo Kronecker Reed Muller Expansions, *International Conference on Electronics, Information and Communication Engineering*, World Academy of Science, Engineering and Technology, Paris / France 27-29 July 2011, pp.1367-1373.

References

- [1] Pappas N. L., 1994. *Digital Design*. United States of America. West Publishing Company.
- [2] Micheli G. D., 1994. *Synthesis and Optimization of Digital Circuits*. New York. Mc-Graw-Hill Inc.
- [3] Richard R., 1996. Tutorial: Design of a Logic Synthesis System, *33rd Design Automation conference*, Las Vegas.
- [4] Sasao T., 1999. *Switching Theory for Logic Synthesis*. Boston. Kluwar Academic Publishers.
- [5] Lala P. K., 1996. *Practical Digital Logic Design and Testing*. London. Prentice Hall International (UK) limited.
- [6] Almaini A. E. A., 1994. *Electronic Logic Systems*. London. Prentice Hall.
- [7] Green D.H., 1986. *Modern Logic Design*. Wokingham, England. Addison-Wesley Publishing.
- [8] Sasao T. & Besslich P., 1990. On the Complexity of MOD-2 sum PLA's, *IEEE Transactions Computer*, **39**(2), pp. 262 – 266.
- [9] Reddy S. M., 1971. Easily Testable Realisation for Logic functions, *IEEE Transactions Computer*, 1971, **C-21**(11), pp. 1183–1188.

References

- [10] Sasao T., 1993. *Logic Synthesis and Optimization*. London. Kluwar Academic Publishers.
- [11] Khan Md. M. H. A., 1998. "Development of algorithms for synthesis and minimization of EXOR-based logic", Ph.D. thesis, Department of Computer Science and Engineering. Bangladesh University. Text on p. 39-40.
- [12] Green D.H., 1994. Dual forms of Reed Muller expansions, *IEE proceeding on Computer and Digital Techniques*, **141**(3) May 1994, pp 184-192.
- [13] Tan E.C. & Yang H., 1998. Fast tabular technique for fixed-polarity Reed-Muller logic with inherent parallel processes, *International Journal of Electronics*, **85**(4), pp. 511-520.
- [14] Faraj K. & Almaini A.E.A, 2007. Minimization of Dual Reed-Muller forms using Dual property, *WSEAS Transaction on Circuits and Systems*, **6**(1), pp. 9–15.
- [15] Wang L., Almaini A.E.A., & Bystrov A., 1999, Efficient polarity conversion for large Boolean functions, *IEE proceeding on Computer and Digital Techniques*, **146**(4), July 1999. pp. 197–204.
- [16] Habib M. K., 2002. A new approach to generate Fixed-Polarity Reed- Muller expansions for Completely and Incompletely specified Functions, *International Journal of Electronics*, **89**(11), pp. 845-876.

References

- [17] Xu H., Yang M., Wang L., Tong J. R., & Almaini A.E.A, 2007. An Efficient Transformation Method for DFRM Expansions, *ASICON Proceeding of the 7th IEEE International conference on ASIC: ASICON 2007*, pp. 1158-1161.
- [18] Almaini A.E.A. & Ping S., 1997. Algorithms For Reed-Muller expansions of Boolean functions and Optimization of Fixed Polarities', *The fourth International Conference on Electronics, Circuits and system design: ICECS, Cairo, EGYPT, Dec. 1997*, pp. 148-153.
- [19] Yang M., Wang L., Tong J.R., & Almaini A.E.A, 2008. Techniques for Dual forms of Reed-Muller expansion Conversion, *Integration, VLSI Journal*, **41**, pp. 113–122.
- [20] Faraj K. & Almaini A.E.A , 2007. Optimal Expression for Fixed Polarity Dual Reed-Muller forms, *WSEAS Transactions on Circuits and Systems*, **6**(3), pp. 364–371.
- [21] Almaini A.E.A., & McKenzie L., 1996. Tabular techniques for Generating Kronecker expansions, *IEE Proceeding-Computer and Digital technology*, **143**(4) July 1996. pp. 205–212.
- [22] McKenzie L. & Almaini A.E.A., 1997. Generating Kronecker expansions from reduced Boolean forms using tabular methods, *International Journal of Electronic* , **82**(4), pp. 313-325.

References

- [23] Green D.H.,1990. Reed-Muller canonical forms with mixed polarity and their manipulations, *IEE Proceedings on Computers and Digital Techniques*, **137**(1), Part E, pp. 103-113.
- [24] Habib M. K.,1993. 'Efficient and fast algorithm to generate minimal Reed-Muller Exclusive-OR expansions with mixed polarity for completely and incompletely specified function and its computer implementation', *Computers Electronic Engineering*, **19**(3), pp. 193-211.
- [25] Al Jassani B.A., Urquhart N., & Almaini A.E.A., 2008. Optimization of MPRM functions using Tabular Techniques and Genetic Algorithms , *Mediterranean Journal of Electronic and Communications: MEDJEC* , **4**(4), pp.115-125.
- [26] Al Jassani B. A., Urquhart, N., &Almaini, A.E.A., 2010. Manipulation and Optimization techniques for Boolean logic, *IET Computer and Digital techniques*, **4**(3), pp. 227-239.
- [27] Hartmanis J. & Stearns R.E., 1966. *Algebraic Structure theory of sequential machines*', United State of America. Prentice-Hall, Inc.
- [28] Dolotta T.A., & McCluskey E.J., 1964. The Coding of Internal States of Sequential Circuit', *IEEE Transactions on Electronic Computers*, October 1964, pp. 549-562.
- [29] Ashar P., Devadas S., & Richard Newton A., 1992. *Sequential Logic Synthesis*, London. Kluwer Academic Publishers.

References

- [30] Friedman A. D. & Menon P. R., 1975. *Theory and Design of switching Circuits.* , 1st. United States of America. Pitman Publishing limited.
- [31] Almaini A.E.A., Thomson P. & Hanson D., 1991. Tabular Techniques for Reed-Muller logic, *International Journal of Electronics*, **70**(1), pp. 23-34.
- [32] Green D.H., 1995. Reed-Muller variable-entered vectors and map, *International Journal of Electronics*, **78**(1), pp. 161-186.
- [33] Helliwell M. & Perkowski M., 1988. A Fast algorithm to Minimize Multi -Output Mixed Polarity Generalized Reed-Muller Forms, *Proceedings 25th IEEE/ACM Conference on Design Automation*, Anaheim, CA, pp. 427- 432.
- [34] Wang L. & Almaini A.E.A., 2002. Exact minimisation of large multiple output FPRM functions, *IEE Proc.-Computer Digital Techniques*, **149**(5) Sep. 2002, pp. 203-213.
- [35] Wang L. & Almaini A.E.A., 2002. Optimization of Reed-Muller PLA implementations, *IEE Proc.-Circuit Devices System*, **149**(2) April 2002, pp.119-128.
- [36] McKenzie L., Almaini A.E.A., Miller J.F., & Thomson P., 1993. Optimization of Reed-Muller logic functions, *International Journal of Electronics*, **75**(3), pp. 451-466.
- [37] Yang M., Xu H., Wang L., Tong J.R., & Almaini A.E.A, 2006

References

- Exact Minimization of Large Fixed Polarity Dual Form of Reed-Muller Functions, *Proceedings of the 8th IEEE international conference on solid-state and integrated circuit technology, China*, pp. 1931-1933.
- [38] Faraj K. & Almaini A.E.A., 2008. Optimal Polarity for Dual Reed-Muller Expressions, *7th WSEAS International Conference on MICRO Electronics, Nanoelectronics, Optoelectronics, Istanbul, Turkey*, pp. 102–109.
- [39] Cheng M., Chen X., Faraj K., & Almaini A.E.A., 2003. Expansion of Logical function in the OR-coincidence System and the transform between it and Maxterm expansion, *IEE proceeding on Computer and Digital technology*, **150**(6) Nov. 2003, pp.397- 402.
- [40] Pengjun W., 2006. Tabular Technique for OR-coincidence Logic, *Journal of Electronic, China*, **23**(2), pp. 267-273.
- [41] Almaini A.E.A. & Zhuang N., 1997. Variable ordering of BDDs for Multi output Boolean Functions using Evolutionary Techniques. *Fourth IEEE- ICECS'97 conference, Cairo, EGYPT*, December 1997, pp. 1239-1244.
- [42] Xia Y., Ye X., Wang L., Zou Z., & Almaini A.E.A., 2005. Novel synthesis and optimization of multi-level mixed polarity Reed-Muller functions', *Journal of Computer Science and Technology*, **20**(6) Nov. 2005, pp. 895- 900.

References

- [43] Oh P. & Almaini A.E.A., 2007. Decision diagrams Using 2 Variable Nodes, *WSEAS Transactions on Circuits and Systems*, **6**(3) March 2007, pp 372-379.
- [44] Aborhey S., 2001. Reed-Muller tree based minimisation of fixed polarity Reed-Muller expansions, *IEE Proc.-Computer Digital Techniques*, **148**(2) March 2001, pp. 63-70.
- [45] Drechsler R., Becker B., 1998. *Binary Decision Diagrams*, Boston. Kluwer Academic Publishers.
- [46] Drechsler R., Becker B., & Göckel N., 1995. A Genetic Algorithm for 2- Level AND/EXOR Minimization, *Conference on Synthesis and System Integration of Mixed Technologies, Nara*, August 1995, pp. 49-56.
- [47] Drechsler R., Becker B., & Drechsler N., 2000. Genetic Algorithm for minimisation of fixed Polarity Reed-Muller expressions, *IEE Proc.-Computer Digital Techniques*, **147**(5) Sep. 2000, pp. 349-353.
- [48] Pengjun W., Hui L., & Zhenhai W., 2010. MPRM Expressions Minimisation Based on Simulated Annealing Genetic Algorithm, *IEEE International Conference on Intelligent systems and knowledge Engineering, Hangzhou*, pp.261-265.
- [49] Chein-Chung T., Malgorzata M.-S., 1996. Logic Synthesis for Testability, *IEEE proceedings, 6th Great Lakes Symposium on VLSI, USA*, March 1996, pp.118-121.
- [50] Hartmanis J., 1959. On the State Assignment Problem for

References

- Sequential Machines I', *IRE Transactions on Electronic Computers*, **EC-8**, pp.439-440.
- [51] Ali B., Almaini A.E.A., & Kalganova T., 2004. Evolutionary Algorithms and their Use in the Design of Sequential Logic Circuits, *Genetic Programming and Evolvable Machines*, **5**, pp.11-29.
- [52] Almaini A.E.A, Miller J.F., Thomson P., & Billina S., 1995. State assignment of Finite State Machines using a Genetic Algorithm, *IEE Proceeding on Computer and Digital Techniques*, **142**(4), pp. 279-286.
- [53] Xia Y. , Almaini A.E.A, & Wu X., 2003. Power Optimisation Of Finite State Machines based on Genetic Algorithm, *Journal of Electronics*, **20**(3), May 2003, pp. 194-201.
- [54] Chattopadhyay S., 2005. Area Conscious State Assignment with Flip- Flop and Output Polarity Selection for Finite State Machine Synthesis-A Genetic Algorithm Approach, *Computer Journal*, **48**(4) May 2005, pp.443-450.
- [55] Olson E. P., 1995. "Optimal State Assignment of Sequential Circuits using Genetic local search with Flexible Cost Functions", Ph.D. thesis, Department Of Computer Science, University of Illinois, Urbana-Champaign, 1995.
- [56] Shiue W. -T., 2004. Novel State Minimisation and State

References

- Assignment in Finite State Machine Design for low-power Portable Devices, *Integration, the VLSI Journal*, **38**, pp. 549-570.
- [57] Chow S.-H., Ho YI.-CH., Hwang T., & Liu C.L., 1996. Low Power Realization of Finite State Machines—A Decomposition Approach, *ACM Transactions on Design Automation of Electronic Systems*, **1** (3), pp. 315–340.
- [58] Rho J.-K., Hachtel G.D., Somenzi F. , & Jacoby R.M., 1994. Exact and Heuristic Algorithms for the Minimisation of Incompletely Specified State Machines, *IEEE Trans. on Computer-Aided Design of integrated circuits and systems*, **13**(2) Feb. 1994, pp.167-177.
- [59] Gořen S. & Ferguson F. J., 2007. On State Reduction of Incompletely specified Finite State Machines, *Computers and Electrical Engineering*, **33**, pp. 58–69.
- [60] Xia Y., Ye X. , Wang L., Tao W., & Almaini A.E.A., 2006. A Uniform Framework of Low Power FSM Approach, *IEEE International Conference on Communication, Circuits and Systems*, China, June 2006, pp. 2642-2646.
- [61] Cho S. & Park S., 2003. A New Synthesis technique of Sequential Circuits for low power and testing, *Current Applied Physics*, pp.83-86.
- [62] Aly W.M., 2009. Solving the State Assignment Problem Using Stochastic Search Aided with Simulated Annealing, *American J. of Engineering and Applied Sciences*, **2**(4), pp. 710- 714.

References

- [63] Athanasopoulou E. & Hadjicostis C. N., 2008. Bounds on FSM Switching Activity, *Journal of Sign Process System, Springer*, **53** June 2008, pp. 411-418.
- [64] Chattopadhyay S. & Reddy P. N., 2003. , Finite State Machine State Assignment targeting Low Power Consumption, *IEE Proceeding on Computer and Digital Techniques*, **1**(1), pp 61-70.
- [65] Al Jassani B.A. & Almaini A.E.A., 2011. Computing the Pseudo Kronecker Reed Muller Expansions, *International Conference on Electronics, Information and Communication Engineering, World Academy of Science, Engineering and Technology*, 27-29 July 2011, pp.1367-1373.
- [66] AL Jassani B. A., Urquhart N., & Almaini A.E.A., 2009. Minimization of Incompletely Specified Mixed Polarity Reed Muller Functions using Genetic Algorithm, *IEEE 3rd International Conference on Signals, Circuits and Systems*, Tunisia, CE5-02.
- [67] AL Jassani B. A., Urquhart N., & Almaini A.E.A. 2011. State assignment for Sequential Circuits using Multi-Objective Genetic Algorithm', *IET Computer and Digital techniques* , **5**(5),pp.296-305.
- [68] Loureiro G.V., 1993. "Digital Systems Design for Testability Based on Reed-Muller Three-Circuit Approach", Ph.D. thesis, UMIST, Department of Electrical Engineering and Electronics, UK.

References

- [69] Green D.H., 1996. Hybrid forms of Reed-Muller expansions, *International Journal of Electronics*, **81**(1), pp. 15-35.
- [70] Yang, S.,1991. Logic Synthesis And Optimization Benchmarks User Guide., *Technical Report 3, Microelectronics Centre of North Carolina*.
- [71] Robert L. ,1988. Logic Synthesis and Optimization Benchmarks User Guide, *Technical Report 2, Microelectronics Centre of North Carolina*, Dec. 1988.
- [72] Centre of Electronic System Design, The Donald O. Pederson, © 2002-2011, UC Regents, accessed from <http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm>, on May 2011
- [73] Sontrak, Technical software, Espresso software, accessed from http://sontrack.com/downloads_esp.aspx, on March 2010.
- [74] Edwards C.R., 1971. The logic of Boolean matrices, *Computer journal*, 15, pp. 247- 253.
- [75] Green D.H., 1994. Dual forms of Reed Muller expansions, *IEE Proceedings on Computer and Digital Technology*, **141**(3) May 1994, pp 184-192.
- [76] Eiben A.E. & Smith J.E., 2003. *Introduction to Evolutionary Computing*. 2nd. Berlin. Springer.

References

- [77] Z. Michalewicz, 1995. *Genetic Algorithms + Data Structures = Evolutionary Programs*, 3rd, Berlin, Springer.
- [78] Goldberg D.E., 1989. *Genetic Algorithms in Search, Optimisation, and Machine Learning*, 1st, Boston, Addison Wesley.
- [79] Douglas L., 1985. *Design of Logic Systems*. 1st. London. T.j. Press (Padstow) Ltd , Padstow, Cornwall.
- [80] Benini L. & Micheli G. De., 1994. State assignment for Low Power Dissipation, *IEEE Custom Integrated Circuits Conference*, **30(3)**, pp. 136-139.
- [81] Finite Mathematics & Applied Calculus, summary of Chapter 9, accessed from http://people.hofstra.edu/stefan_waner/RealWorld/Summary8.html, on 7/3/2011.
- [82] Zbigniew M. & David B.F., 2004. *How to Solve it: Modern Heuristics*. 2nd. Berlin. Springer.
- [83] Ruhul S., Masoud M., & Xin Y., 2002. *Evolutionary Optimization*, Boston, Kluwer Academic Publishers.
- [84] Deb K., Pratap A., Agarwal S., & Meyarivan T., 2002. A Fast and Elitist Multi objective Genetic Algorithm: NSGA-II, *IEEE Trans. on evolutionary computation*, **6(2)**, pp182-197.

References

- [85] Abdullah K. , David W. C., & Smith A. E., 2006. Multi-objective optimization using genetic algorithms: A tutorial, *Reliability Engineering and System Safety*, 91, pp. 992–1007.

- [86] Villa T. & Sangiovanni-Vincentelli A., 1990. NOVA: State assignment offinite state machine for optimal two-level logic implementation, *IEEE Transactions on Computer_Aided Design of Integrated Circuits and Systems*, 9(9), pp.905-924.

- [87] Hong S.K. , Park I.C. , Hwang S.H. , & Kyung C.M., 1994. State assignment in finite state machines for minimal switching power consumption, *IEE Electronic letter*, 30(8), pp 627-629.

Disk Containing the Programs

The following programs are developed in this thesis and written in C++ language. The attached disk contains the following programs and electronic version of the thesis.

1. Bidirectional Conversion between FPRM and MPRM for single and multi-output completely specified Boolean functions (**ALLMPRM**): this program read the benchmark file in PLA format. Initially, this program converts the CSOP coefficients into PPRM (FPRM_Polarity 0) coefficients using Tabular technique and stores the coefficients of PPRM in memory. Then, it converts the PPRM coefficients to all 3^n polarities related to MPRM class. This program performs exhaustive search and produces the optimum polarity among 3^n MPRM polarities. More details can be found in Chapter 2.
2. Bidirectional Conversion between FPDRM and MPDRM (**ALLMPDRM**) for single and multi-output specified Boolean functions: This program read the benchmark file in PLA format. Initially, this program converts the CPOS coefficients into PPDRM (FPDRM_Polarity 0) coefficients using Tabular technique and stores the coefficients of PPDRM in memory. Then, it converts the PPDRM coefficients to all 3^n polarities related to MPDRM class. This program performs exhaustive search and produces the optimum polarity among 3^n MPDRM polarities. More details can be found in Chapter 2.
3. Polarity Conversion between CSOP and MPRM for single and multi-output specified Boolean function (**Minterm Separation**): This program reads the PLA format of Benchmark examples and stores it in

memory. Then, it converts the CSOP expansions into all MPRM expansions. This program runs exhaustive search among 3^n MPRM to produce the optimum polarity with less number of terms. More details can be found in Chapter 3.

4. Polarity Conversion between CPOS and MPDRM for single and multi-output completely specified Boolean function (**Maxterm_Separation**): This program reads the PLA format of Benchmark examples and stores it in memory. Then, it converts the CPOS expansions into all MPDRM expansions. This program runs exhaustive search among 3^n MPDRM to produce the optimum polarity with less number of terms. More details can be found in Chapter 3.
5. Minimisation technique for RM form (**RM_Minimisation**): this program reads the benchmark file (in PLA format). Initially, this program converts the CSOP coefficients into PPRM (FPRM_Polarity 0) coefficients using Tabular technique and stores the coefficients of RMPPRM in memory. Then it minimises it using RM_Minimisation technique (Chapter 4) to generate expansions with minimal number of terms.
6. Minimisation technique for DRM form (**DRM_Minimisation**): this program reads the benchmark file (in PLA format). Initially, this program converts the CPOS coefficients into PPDRM (FPDRM_Polarity 0) coefficients using Tabular technique and stores the coefficients of PPDRM in memory. Then it minimises it using DRM_Minimisation technique (Chapter 4) to generate expansions with minimal number of sums.
7. GA for single and multi-output completely specified Boolean functions (**GA_samePol**): this program is implemented to produce optimal

polarity among MPRM for single and multi output using Extended_Tabular (Chapter 2) technique in its fitness function. It produces an expansion which is optimal among MPRM with all outputs having same polarity considering the sharing between terms. More details can be found in Chapter 5.

8. GA for single and muti-output specified Boolean functions (**GA_diffPol**): this program is implemented to produce optimal polarity among MPRM for single and multi output using Extended_Tabular (Chapter 2) technique in its fitness function. It produces an expansion which is optimal among MPRM with all outputs having different polarities trying to achieve better results. More details can be found in Chapter 5.
9. GA for single and muti-output specified Boolean functions (**GA_MINSEP**): this program is implemented to produce optimal polarity among MPRM for single and multi output using Minterm separation (Chapter 3) technique in its fitness function. It produces an expansion which is optimal among MPRM. More details can be found in Chapter 5.
10. GA for single and muti-output completely specified Boolean functions (**GA_MAXSEP**): this program is implemented to produce optimal polarity among MPDRM for single and multi output using Maxterm separation (Chapter 3) technique in its fitness function. It produces an expansion which is optimal among MPDRM. More details can be found in Chapter 5.

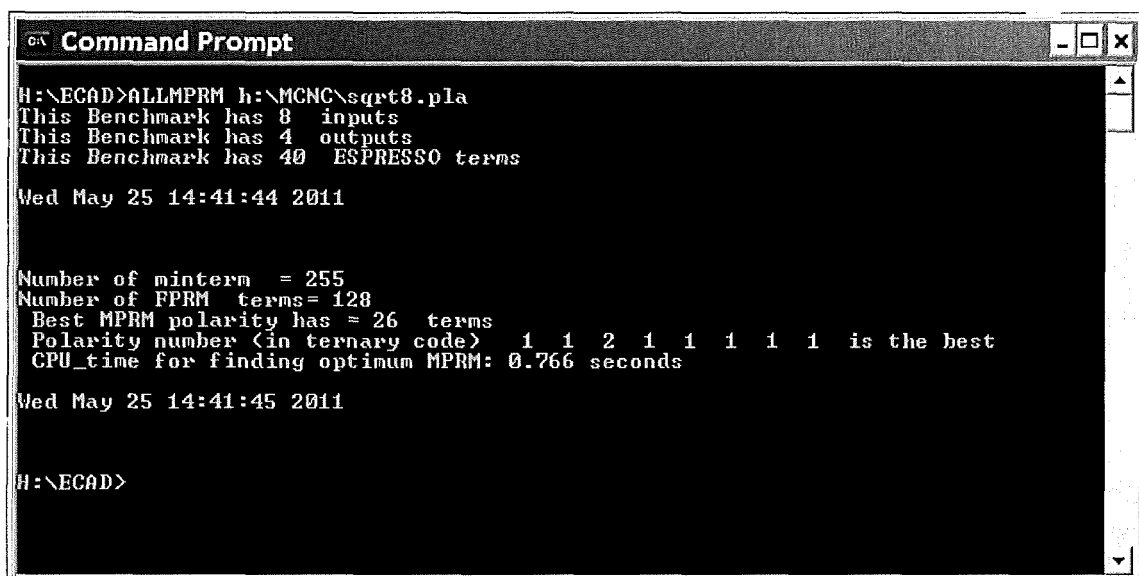
- 11.**GA for multi-output completely and incompletely specified Boolean functions (**GA_RM**): this program is implemented to produce optimal polarity among MPRM for single and multi output completely and incompletely specified Boolean functions using Minterm separation (Chapter 3) technique in its fitness function. This GA is split into two stages to improve its performance and to achieve better results. More details can be found in Chapter 5.

- 12.**Muti Objective GA for completely and incompletely specified sequential logic circuits (**SEQ_GA**): this program reads the FSM benchmark (KISS format) and stores it in memory. This program is implemented using Pareto ranking scheme to produce solutions which are the best in terms of number of terms and switching activity. More details can be found in Chapter 6.

APPENDIX

An Example of Running of Each Program

1. The following run for ALLMPRM program is to find optimal MPRM polarity for benchmark sqrt8.pla. The program found that con1.pla benchmark has optimum polarity number is $(11211111)_3 = (2523)_{10}$ which has 26 terms only. The time required for finding this result is 0.766 milli seconds. One parameter is needed to run this program which is the name of the benchmark file. The program produces the number of minterms for the benchmark, number of FPRM terms, optimum polarity in ternary number, and number of terms for the optimum polarity, CPU time in milliseconds which is counted as difference in CPU time before and after producing these results, time and date before and after producing results.



```
H:\ECAD>ALLMPRM h:\MCNC\sqr8.pla
This Benchmark has 8 inputs
This Benchmark has 4 outputs
This Benchmark has 40 ESPRESSO terms

Wed May 25 14:41:44 2011

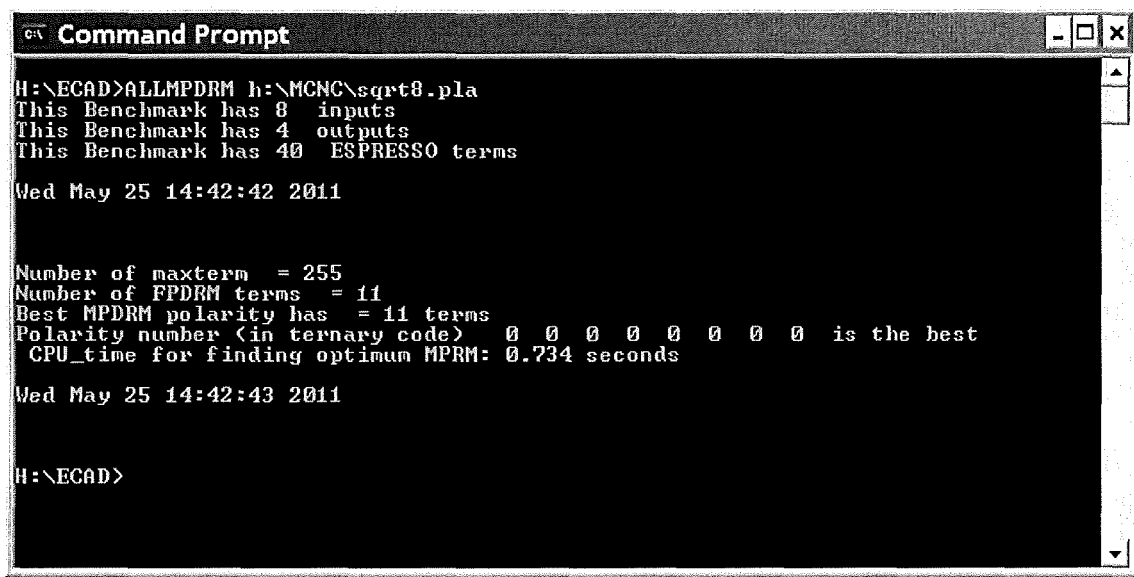
Number of minterm = 255
Number of FPRM terms= 128
Best MPRM polarity has = 26 terms
Polarity number (in ternary code) 1 1 2 1 1 1 1 1 is the best
CPU_time for finding optimum MPRM: 0.766 seconds

Wed May 25 14:41:45 2011

H:\ECAD>
```

Appendix an Example of Running of Each Program

2. The following run for ALLMPDRM program is to find optimal MPDRM polarity for benchmark con1.pla. The program found that sqrt8.pla benchmark has optimum polarity number is $(00000000)_3 = (0)_{10}$ which has 11 terms only. The time required for finding this result is 0.734 milliseconds. This program is the same as ALLMPRM program in terms of parameter needed to run this program and the results produced after running this program.



```
Command Prompt
H:\ECAD>ALLMPDRM h:\MCNC\sqr8.pla
This Benchmark has 8 inputs
This Benchmark has 4 outputs
This Benchmark has 40 ESPRESSO terms

Wed May 25 14:42:42 2011

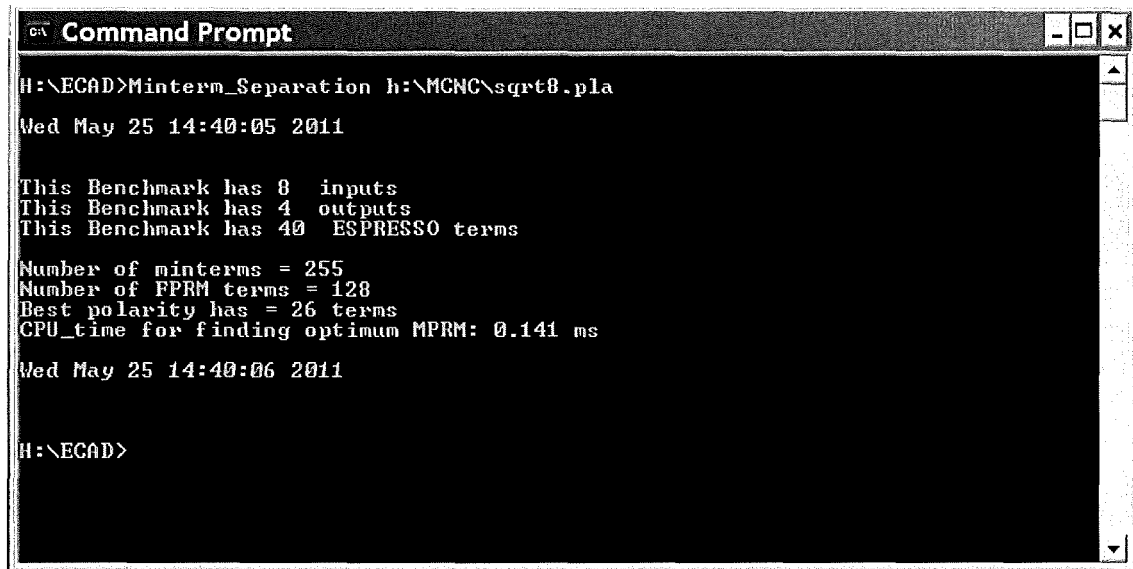
Number of maxterm = 255
Number of FPDRM terms = 11
Best MPDRM polarity has = 11 terms
Polarity number (in ternary code) 0 0 0 0 0 0 0 0 is the best
CPU_time for finding optimum MPRM: 0.734 seconds

Wed May 25 14:42:43 2011

H:\ECAD>
```

3. The following run for Minterm_Separation program is to find optimal MPRM polarity for benchmark sqrt8.pla. The program found that optimum polarity for this benchmark has 26 terms only. The time required for finding this result is 0.141 milliseconds.

Appendix an Example of Running of Each Program



```
Command Prompt
H:\ECAD>Minterm_Separation h:\MCNC\sqr8.pla
Wed May 25 14:40:05 2011

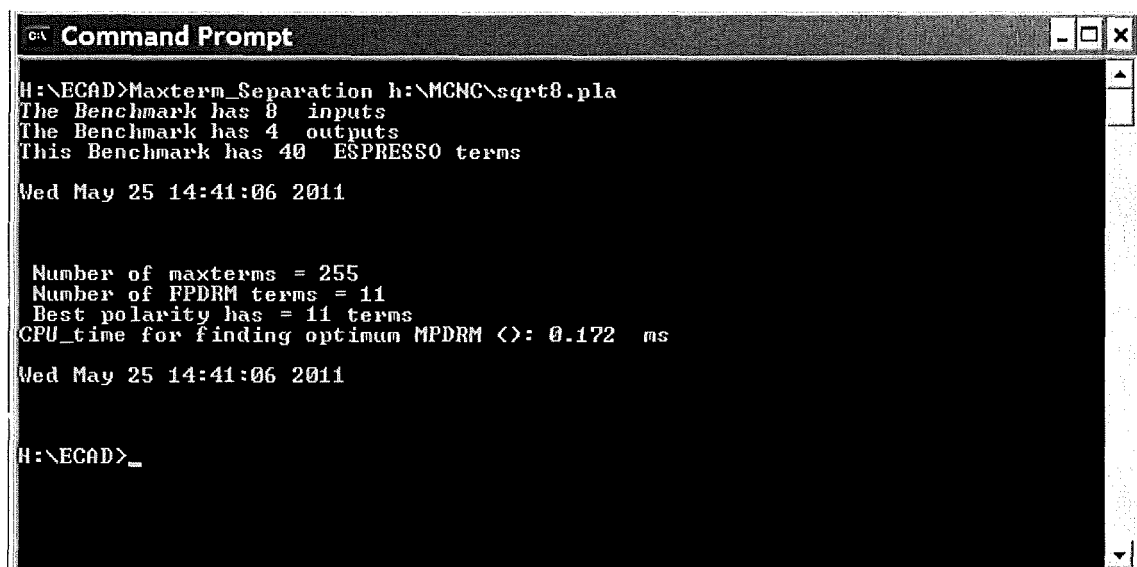
This Benchmark has 8 inputs
This Benchmark has 4 outputs
This Benchmark has 40 ESPRESSO terms

Number of minterms = 255
Number of FPRM terms = 128
Best polarity has = 26 terms
CPU_time for finding optimum MPRM: 0.141 ms

Wed May 25 14:40:06 2011

H:\ECAD>
```

4. The following run for Maxterm_Separation program is to find optimal MPDRM polarity for benchmark sqr8.pla. The program found that optimum polarity for this benchmark also has 11 terms only. The time required for finding this result is 0.172 milliseconds.



```
Command Prompt
H:\ECAD>Maxterm_Separation h:\MCNC\sqr8.pla
The Benchmark has 8 inputs
The Benchmark has 4 outputs
This Benchmark has 40 ESPRESSO terms

Wed May 25 14:41:06 2011

Number of maxterms = 255
Number of FPDRM terms = 11
Best polarity has = 11 terms
CPU_time for finding optimum MPDRM (<): 0.172 ms

Wed May 25 14:41:06 2011

H:\ECAD>_
```

5. The following run for RM_Minimsation program is to generate expansion with minimal number of terms in RM domain for the

Appendix an Example of Running of Each Program

Benchmark con1.pla. This program produces number of terms for the optimal expression. It also produces the resulted terms after minimisation process for all outputs and polarities for each inputs to be able to derive the minimal expansion as detailed in Chapter 4. The program found that this Benchmark can be designed using 11 terms only. The time required for finding this result is 0.016 milliseconds.

```
Command Prompt
H:\ECAD>RM_Minimisation h:\MCNC\con1.pla
Wed May 25 14:44:33 2011

This Benchmark has 7 inputs
This Benchmark has 2 outputs
This Benchmark has 9 ESPRESSO terms

Number of minterm = 118
number of FPRM = 19
Number of terms after Minimisation in RM domain = 11
CPU_time for the minimisation process = : 0.016 seconds

Wed May 25 14:44:33 2011

The Resulted Terms are as follows :

Terms          Outputs    Polarity Table
-----
0000000        10         0000000
0100010         01         0000000
0100110         01         0000010
0111000         01         0110000
1000101         10         1000000
1001100         10         0000000
1011000         01         0000000
1100001         10         1000000
1100110         01         0000100
1101100         10         0001000
1111000         01         0000000
H:\ECAD>
```

6. The following run for DRM_Minimisation program is to generate expansion with minimal number of terms in DRM domain for the Benchmark con1.pla. This program produces number of terms for the optimal expression. It also produces the resulted sums after minimisation process for all outputs and polarity for each input to be able to derive the minimal expansion as detailed in Chapter 4. The

Appendix an Example of Running of Each Program

program found that this Benchmark can be designed using 12 terms only. The time required for finding this result is 0.015 milliseconds.

```
Command Prompt
H:\ECAD>DRM_Minimisation h:\MCNC\con1.pla
This Benchmark has 7 inputs
This Benchmark has 2 outputs
This Benchmark has 9 ESPRESSO terms

Wed May 25 14:46:05 2011

Number of maxterm = 118
number of FPDRM = 24
Number of terms after Minimisation in DRM domain = 12
CPU_time for the minimisation process = : 0.015 seconds

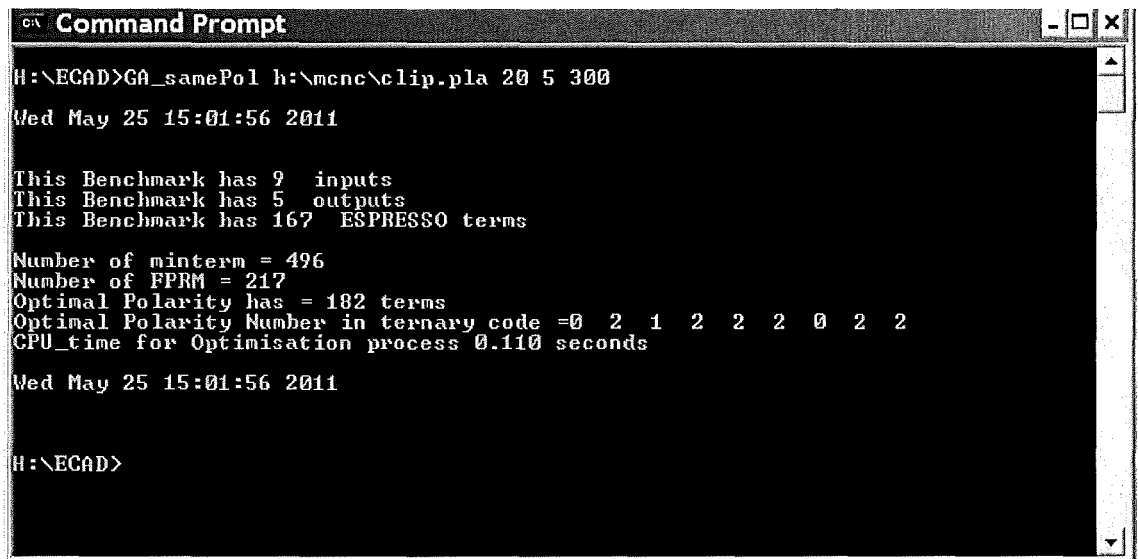
Wed May 25 14:46:05 2011

The Resulted Terms are as follows :
Terms      Outputs      Polarity Table
-----
0000011    10          0001100
0010011    11          1000100
0011001    10          0100000
0011110    01          0000001
0011111    10          0000000
0100011    10          0001000
0110011    10          1000000
0111010    01          0000000
0111111    01          0000000
1000011    10          0101000
1111011    01          0000000
1111111    10          0000000
H:\ECAD>
```

7. The following run for GA_samePol program is to optimise multi-output completely specified Boolean function. This program produces optimal expansion with minimal number of terms in MPRM domain with all outputs having same polarity. It is tested bellow using Benchmark clip.pla. This program produces number of terms for the optimal expression. Four parameters are needed to run this program which are the name of the benchmark file, population size, tournament size, and number of evaluations. It produces the optimal polarity number in ternary code, number of terms for this optimal polarity and time required to produce this results. The program found that con1.pla benchmark has optimum polarity number for all outputs is $(021222022)_3$

Appendix an Example of Running of Each Program

= (5813)₁₀ which has 182 terms only. The time required for finding this result is 0.11 milli seconds.



```
Command Prompt
H:\ECAD>GA_samePol h:\menc\clip.pla 20 5 300
Wed May 25 15:01:56 2011

This Benchmark has 9 inputs
This Benchmark has 5 outputs
This Benchmark has 167 ESPRESSO terms

Number of minterm = 496
Number of FPRM = 217
Optimal Polarity has = 182 terms
Optimal Polarity Number in ternary code =0 2 1 2 2 2 0 2 2
CPU_time for Optimisation process 0.110 seconds

Wed May 25 15:01:56 2011

H:\ECAD>
```

8. The following run for GA_diffPol program is to optimise multi-output completely specified Boolean function. This program produces optimal expansion with minimal number of terms in MPRM domain with outputs having different polarity. It is tested bellow using Benchmark clip.pla. This program produces number of terms for the optimal expression. Four parameters are needed to run this program which are the name of the benchmark file, population size, tournament size, and number of evaluations. It produces the optimal polarity number in ternary code for each output, total number of terms for for all outputs considering the sharing between terms, and time required to produce this results. The program found that con1.pla benchmark can be designed using 151 terms only. The time required for finding this result is 3.687 milliseconds.

Appendix an Example of Running of Each Program

```
GA Command Prompt
H:\ECAD>GA_diffPol h:\menc\clip.pla 20 5 9000
Wed May 25 16:09:58 2011

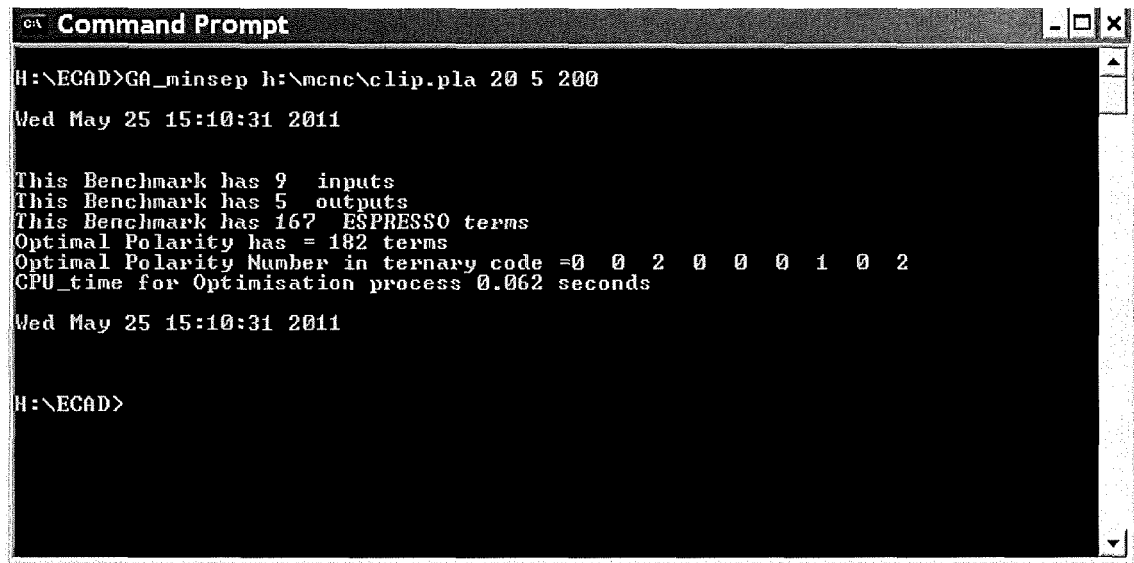
This Benchmark has 9 inputs
This Benchmark has 5 outputs
This Benchmark has 167 ESPRESSO terms

Optimal Polarity has = 151 terms
Polarity of Output 1 in ternary number=2 1 2 1 1 1 2 0 0
Polarity of Output 2 in ternary number=2 0 1 1 1 1 2 2 0
Polarity of Output 3 in ternary number=2 0 1 0 0 1 2 1 0
Polarity of Output 4 in ternary number=2 0 1 0 0 0 2 1 1
Polarity of Output 5 in ternary number=0 0 1 0 0 2 2 1 1
CPU_time for Optimisation process 3.687 seconds
Wed May 25 16:10:02 2011

H:\ECAD>
```

The following run for GA_MINSEP program is to find optimal polarity among 3^n MPRM for single and multi-output completely specified Boolean function. The fitness function of this program is implemented using Minterm separation technique explained in Chapter 3. It is tested below using Benchmark clip.pla. Four parameters are needed to run this program which are the name of the benchmark file, population size, tournament size, and number of evaluations. It produces the optimal polarity number in ternary code, number of terms for this optimal polarity and time required to produce this results. The program found that con1.pla benchmark has optimum polarity number for all outputs is $(002000102)_3 = (1469)_{10}$ which has 182 terms only. The time required for finding this results is 0.062 milliseconds

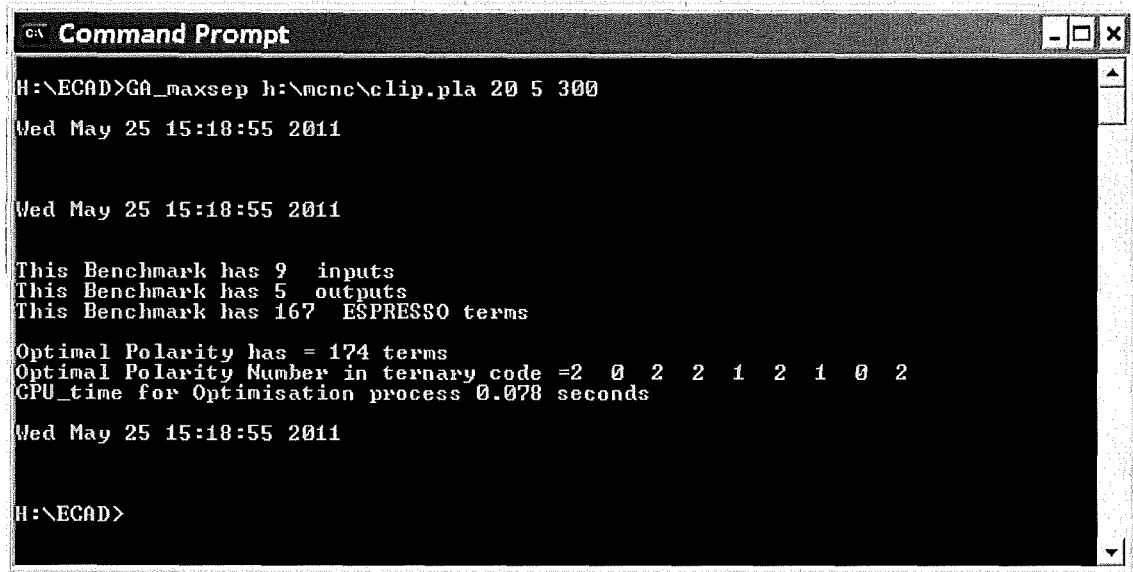
Appendix an Example of Running of Each Program



```
CA Command Prompt
H:\ECAD>GA_minsep h:\menc\clip.pla 20 5 200
Wed May 25 15:10:31 2011
This Benchmark has 9 inputs
This Benchmark has 5 outputs
This Benchmark has 167 ESPRESSO terms
Optimal Polarity has = 182 terms
Optimal Polarity Number in ternary code =0 0 2 0 0 0 1 0 2
CPU_time for Optimisation process 0.062 seconds
Wed May 25 15:10:31 2011
H:\ECAD>
```

9. The following run for GA_MAXSEP program is to find optimal polarity among 3^n MPDRM for single and multi-output completely specified Boolean function. The fitness function of this program is implemented using Maxterm separation technique explained in Chapter 3. It is tested below using Benchmark clip.pla. Four parameters are needed to run this program which are the name of the benchmark file, population size, tournament size, and number of evaluations. It produces the optimal polarity number in ternary code, number of terms for this optimal polarity and time required to produce this results. The program found that con1.pla benchmark has optimum polarity number for all outputs is $(202212102)_3 = (15212)_{10}$ which has 174 terms only. The time required for finding this results is 0.078 milliseconds

Appendix an Example of Running of Each Program



```
Command Prompt
H:\ECAD>GA_maxsep h:\mcnc\clip.pla 20 5 300
Wed May 25 15:18:55 2011

Wed May 25 15:18:55 2011

This Benchmark has 9 inputs
This Benchmark has 5 outputs
This Benchmark has 167 ESPRESSO terms

Optimal Polarity has = 174 terms
Optimal Polarity Number in ternary code =2 0 2 2 1 2 1 0 2
CPU_time for Optimisation process 0.078 seconds

Wed May 25 15:18:55 2011

H:\ECAD>
```

10. The following run for multi stage GA_RM program is to find optimal polarity among 3^n MPRM for single and multi-output completely and incompletely specified Boolean function. The fitness function of this program is implemented using Minterm separation technique explained in Chapter 3. It is tested below using Benchmark rd53.pla. Four parameters are needed to run this program which are the name of the benchmark file, population size, tournament size, and number of evaluations. It produces the optimal polarity number in ternary code, number of terms for this optimal polarity, time required to produce this results, and selected don't care terms to produce this results. The program found that rd53.pla benchmark has optimum polarity number $(11111)_3 = (121)_{10}$ which has 10 terms only. The time required for finding the results is 0.062 milliseconds.

Appendix an Example of Running of Each Program

```
Command Prompt
H:\ECAD>GA_rm h:\mcnc\rd53.pla 40 5 99000
Wed May 25 15:28:34 2011

Total number of dont care terms = 54
Best Polarity has 10 terms
Optimal Polarity Number in ternary Code=1 1 1 1 1

Selected dont_care terms to produce the above Optimal Polarity are as follows:

term 0 for output 0
term 1 for output 0
term 2 for output 0
term 3 for output 0
term 7 for output 0
term 8 for output 0
term 10 for output 0
term 11 for output 0
term 12 for output 0
term 13 for output 0
term 14 for output 0
term 16 for output 0
term 17 for output 0
term 19 for output 0
term 21 for output 0
term 22 for output 0
term 24 for output 0
term 25 for output 0
term 26 for output 0
term 28 for output 0
term 0 for output 1
term 3 for output 1
term 6 for output 1
term 10 for output 1
term 12 for output 1
term 15 for output 1
term 17 for output 1
term 18 for output 1
term 23 for output 1
term 27 for output 1
term 29 for output 1
term 30 for output 1
term 1 for output 2
term 2 for output 2
term 15 for output 2
term 23 for output 2
term 27 for output 2
term 29 for output 2
term 30 for output 2
term 31 for output 2

Wed May 25 15:28:36 2011

H:\ECAD>_
```

11. The following run for Multi-Objective GA for sequential circuits SEQ_RM program is to find optimal state assignment for single and multi-output completely and incompletely sequential circuits.. It is tested bellow using Benchmark bbtas. Kiss2. Four parameters are needed to run this program which are the name of the benchmark file, population size, tournament size, and number of evaluations. It produces the Number of flip flop required to design this circuit, Best state assignments with its switching activity and number of terms, and time required to produce the results.

Appendix an Example of Running of Each Program

```
c:\ Command Prompt
H:\ECAD>SEQ_GA h:\MCNC\bhtas.kiss2 20 5 300
Wed May 25 16:38:18 2011

This Benchmark has 2 Inputs
This Benchmark has 2 Outputs
This Benchmark has 24 States
Number of the required flip flops = 3

Best assignments are as follows:
  1  5  6  2  0  3  4  7
which has 8 terms & Switcting Activities =0.808696
  2  5  6  4  0  1  3  7
which has 9 terms & Switcting Activities =0.443478

Wed May 25 16:39:21 2011

H:\ECAD>
```

The Format of Input Benchmark Files

- In order to run all programs for synthesis and optimisation of combinational circuits, the input file in PLA format has to be as shown below. The first line gives number of inputs, the second line gives number of outputs, and the 3rd line gives the number of terms for this benchmark.

```
5           // Number of inputs
1           // Number of outputs
16          // Number of terms
11111 1
01110 1
10110 1
00111 1
11010 1
01011 1
10011 1
00010 1
11100 1
01101 1
10101 1
00100 1
11001 1
01000 1
10000 1
00001 1
```

- In order to run the last programs for optimisation of sequential circuits, the input file in KISS2 format has to be as shown below. The first line gives number of inputs, the second line gives number of outputs, the 3rd line gives number of transitions for all combinations of inputs, and the 4th line gives number of states used starting from st0.

Appendix an Example of Running of Each Program

```
2          // Number of inputs
2          // Number of outputs
24         // Number of transitions
6          // Number of states
00 st0 st0 00
01 st0 st1 00
10 st0 st1 00
11 st0 st1 00
00 st1 st0 00
01 st1 st2 00
10 st1 st2 00
11 st1 st2 00
00 st2 st1 00
01 st2 st3 00
10 st2 st3 00
11 st2 st3 00
00 st3 st4 00
01 st3 st3 01
10 st3 st3 10
11 st3 st3 11
00 st4 st5 00
01 st4 st4 00
10 st4 st4 00
11 st4 st4 00
00 st5 st0 00
01 st5 st5 00
10 st5 st5 00
11 st5 st5 00
```