TOWARDS THE DEVELOPMENT OF AUTOMATED TECHNIQUES FOR THE DESIGN OF DIGITAL CIRCUITS USING GENETIC ALGORITHM

A THESIS

Submitted by

VIJAYAKUMARI C. K

for the award of the degree

of

DOCTOR OF PHILOSOPHY

DIVISION OF ELECTRONICS ENGINEERING
SCHOOL OF ENGINEERING
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY, KOCHI

SEPTEMBER 2016

CERTIFICATE

This is to certify that the thesis entitled **TOWARDS THE DEVELOPMENT OF AUTOMATED TECHNIQUES FOR THE DESIGN OF DIGITAL CIRCUITS USING GENETIC ALGORITHM** submitted by **Vijayakumari C.K** to the Cochin University of Science and Technology, Kochi for the award of the degree of Doctor of Philosophy is a bonafide record of research work carried out by her under our supervision and guidance at the Division of Electronics Engineering, School of Engineering, Cochin University of Science and Technology. The content of this thesis, in full or in parts, have not been submitted to any other University or Institute for the award of any degree or diploma.

Supervising Guide

Dr. Mythili P.
Associate Professor
Division of Electronics
School of Engineering
Cochin University of Science and Technology
Kochi-682 022

Co- Guide

Dr. Rekha K. James
Professor
Division of Electronics
School of Engineering
Cochin University of Science and Technology
Kochi-682 022

Kochi: 682 022
Date: 09/09/2016

DECLARATION

I hereby declare that the work presented in this thesis entitled **TOWARDS THE DEVELOPMENT OF AUTOMATED TECHNIQUES FOR THE DESIGN OF DIGITAL CIRCUITS USING GENETIC ALGORITHM** is based on original research work carried out by me under the supervision and guidance of Dr. Mythili. P, Associate Professor, Division of Electronics Engineering, and Dr. Rekha K. James, Professor, Division of Electronics Engineering, for the award of degree of Doctor of Philosophy with Cochin University of Science and Technology. I further declare that the contents of this thesis in full or in parts have not been submitted to any other University or Institute for the award of any degree or diploma.

Kochi - 682 022
Date :09/09/2016

Vijayakumari C.K
Division of Electronics Engineering

ACKNOWLEDGEMENTS

At the outset, I would like to give special thanks to God Almighty for providing me the opportunity, wisdom, health and knowledge for successful completion of this research work.

I would like to express my profound gratitude to Dr. Mythili. P, Associate Professor, Division of Electronics Engineering, School of Engineering, Cochin University of Science and Technology, for her valuable guidance, timely advice, suggestions and personal attention as supervising guide. She was always a source of constant encouragement and motivation during the course of this research work. Heartfelt thanks are due to her for all the inspiration and immense patience shown to me for pursuing research.

My deepest gratitude and respect also goes to Dr. Rekha K. James, Professor, Division of Electronics Engineering, School of Engineering, Cochin University of Science and Technology, for her guidance, continuous encouragement and constant support as co-guide. Her creative comments and suggestions from the initial conception till the completion of this work are highly appreciated. I express my sincere thanks to her for the support given to me during the course of research work.

I am very much indebted to Dr. Binu Paul, Head of the Division, School of Engg., CUSAT, for her support in pursuing the Ph. D programme in the department. I am extremely thankful to Dr. S. Mridula, Professor, Division of Electronics Engg. for her valuable comments, suggestions and encouragement as Doctoral committee member. I wish to place on record my profound thanks to Dr. R. Gopika Kumari for giving inspiration and suggestions during the interim presentations.

My sincere thanks are due to the research scholars, Mr. Biju V. G, Mr. Anil Kumar C.V, Mr. Anjith T. A, and Mrs. Rema N. R, Ria, Athira, Roshna and Saira Joseph, SOE, CUSAT for their cooperation and support extended.

I am very much indebted to all Faculty and Staff members of Division of Electronics Engineering, School of Engineering, CUSAT, Kochi for their support and cooperation during the research work. The help rendered by the office staff of School of Engineering, CUSAT in fulfilling all the procedural formalities, is gratefully acknowledged.

I am extremely grateful to Dr. Nisha Kuruvilla and Dr. Deepa J., Associate Professors at College of Engg. Chengannur, for the inspiration and timely advice during the course of my research.

I am extremely thankful to all my colleagues, Head of EE department, and Principal at R I T Kottayam for their constant support in fulfilling this work.

I would like to express my deep sense of gratitude to my friends Prof. Geetha Renjin, Dr. Sathish Kumar, Dr. Reena Murali, Dr. Rajesh R, Prof. Pramela Kumari and Prof. Saina Deepthi, for their continuous emotional support and motivation.

I am greatly indebted to all my near and dear ones for their deep love, care and patience to move on with my research work. I am very much grateful to my husband Dr. Predeep S. V, my son Mr. Gopikrishnan P. and daughter Miss. Sreelakshmi P. Nair for their love, understanding, support and encouragement that helped me to fulfill my dream.

Vijayakumari C K

ABSTRACT

Keywords: Combinational logic circuits; Evolutionary Design; Genetic Algorithm; Synchronous Sequential Circuits.

Due to the very high impact of digital electronics in everyday life, the design of digital circuits has gained a great significance. Thus, finding an optimised fully functional circuit with reduced cost is a challenge for the designer. As nature unveils several diverse and striking phenomena, it has become a great source of inspiration for solving hard and complex problems in computer applications and gave rise to several biologically inspired algorithms. This thesis focuses on developing automated techniques for the design of digital circuits using one of the biologically inspired algorithms, Genetic Algorithm (GA).

The first phase of this research work concentrates on developing an evolutionary technique for the design of Combinational Logic Circuits (CLCs) using gates. A new faster 2 Dimensional (2D) chromosomal representation, its suitable 2D cross over and 2D mutation techniques have been proposed. In this work, gates such as AND, XOR, OR and WIRE are considered for the design. Once a 100% functional circuit is obtained, an additional fitness value is assigned for every WIRE used. This ensures minimum number of gates in the evolved circuit in subsequent generations. A comparison between the proposed method and the existing methods has been made and has been observed that the computation time can be reduced significantly using 2D representations.

The second phase explores the design automation of CLCs using Universal Logic Modules (ULMs) such as 2-1 multiplexer (2-1 mux) / 2-1 Reed Muller ULM (2-1 RM ULM). The objective is to generate fully functional circuits with minimum

hardware using GA as the optimisation tool. Applying several modifications on Shannon's / Davio decomposition techniques, two different methods referred to as Constant Input Method (CIM) and Variable Input Method (VIM) are proposed for the design of CLCs. In CIM, the inputs to the circuit are only 0s and 1s whereas in VIM, the inputs can be 0, 1, variables or their complements. The control signals are selected at random from among the variables, their complements or functions derived from the immediate preceding level, which is not allowed in Standard Implementation (SI) technique. The evolved circuits are synthesised on Xilinx FPGA Spartan3 (XC3S400) device.

The third phase investigates into the design of Synchronous Sequential Circuits (SSCs) which involves two stages. i) to determine the optimal state assignment which leads to circuits with minimum hardware and ii) to design the corresponding combinational part to generate the required states for the flip flops and to generate the output. A modified GA has been proposed for the state assignment with a view to minimise the circuit complexity and as a second stage, the combinational part of the FSM is evolved using the techniques proposed in this thesis. The combinational parts are implemented using gates or ULMs such as 2-1 mux / 2-1 RM ULM. Sequence detectors (Mealy machines) and counters (Moore machines) have been designed.

All the above proposed techniques have been validated using benchmark functions and compared with the existing conventional techniques. It is observed that the proposed techniques are better than the methods available in the literature.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

2D	2 Dimensional
ACO	Ant Colony Optimisation
ASC	Asynchronous Sequential Circuit
BDD	Binary Decision Diagram
CIM	Constant Input Method
CLC	Combinational Logic Circuit
DAG	Desired Adjacency Graph
EA	Evolutionary Algorithm
ED	Evolutionary Design
EHW	Evolvable Hardware
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GA	Genetic Algorithm
GP	Genetic Programming
K map	Karnaugh map
MA	Memetic Algorithm
mux	Multiplexer
OBDD	Ordered Binary Decision Diagram
OSA	Optimum State Assignment
PSO	Particle Swarm Optimisation
RAM	Random Access Memory
RM	Reed Muller
ROBDD	Reduced Order Binary Decision Diagram
RWS	Roulette Wheel Selection
SI	Standard Implementation
SLC	Sequential Logic Circuit
SSC	Synchronous Sequential Circuit
STG	State Transition Graph
STT	State Transition Table
TT	Truth table
ULM	Universal Logic Module
VHDL	Very high speed integrated circuit Hardware Description Language
VIM	Variable Input Method
VLSI	Very Large Scale Integration

CHAPTER 1

INTRODUCTION

1.1 DIGITAL CIRCUITS

The present technological period is referred to as the digital age due to the high prominent role of digital systems in our everyday life. Digital systems find applications in communication, business transactions, traffic control, spacecraft guidance, medical treatment, weather monitoring, the internet, and many other industrial and scientific enterprises. Due to the high impact of digital circuits in all of today's computers and devices, the cost of implementation of these circuits is an important design criterion. Thus, finding an optimised fully functional circuit is the major concern of a designer (Mano, 2002). Moreover, with the introduction of Very Large Scale Integration (VLSI) circuits, designers are facing the complex task of packing more functionality into a smaller area and creating a circuit that operates faster compared to the existing ones. So Design Automation (DA) techniques play a vital role in this intricate process of designing digital circuits (Ali, 2003).

Digital systems may be combinational or sequential. A Combinational Logic Circuit (CLC) is an interconnection of logic gates. The output of a CLC at any time is determined by the present combination of inputs. Fig. 1.1 depicts the structure of a CLC. The function of a CLC can be specified in any of the following three ways.

i) Boolean algebra - an algebraic expression that shows the operation of the circuit for a given set of inputs.

ii) Truth Table (TT) – Defines the function by listing the output states in tabular form for each of the possible input combinations.

iii) Logic Diagram – Graphical representation of the circuit that shows the gates used and their connections.

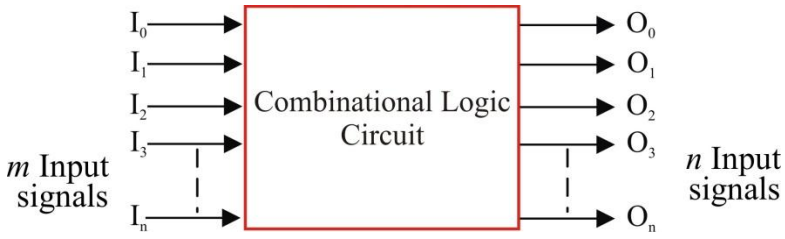


Fig. 1.1 Structure of a Combinational Logic Circuit

All the mathematical operations in computers are performed by CLCs. The most common circuits used in computers are half adders, full adders, half subtractors, full subtractors, multiplexers, demultiplexers, decoders, encoders, etc. Sequential Logic Circuits (SLCs) are those whose outputs are a function of the present values of the inputs and the past values of the outputs. An SLC consists of i) a storage element to store the past output and ii) a combinational circuit to generate the next state. The inputs to the combinational part are the present inputs and the previous outputs fed back from the storage element. The binary information stored in the storage elements at any given time is defined as the state of the sequential circuit at that time. Fig. 1.2 shows the block diagram of an SLC.

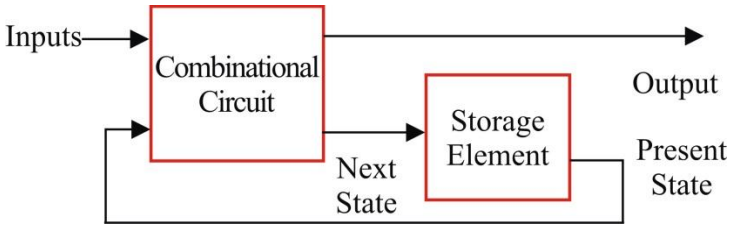


Fig. 1.2 Structure of a Sequential Logic Circuit

SLCs are classified into Synchronous Sequential Circuits (SSCs) and Asynchronous Sequential Circuits (ASCs). SSCs depend on the external clock pulses for their state

transition, whereas no clock is required for the operation of ASCs. The operation of an ASC depends on the propagation delays for the state transitions. Virtually, all practical digital circuits are a combination of sequential and combinational logic. Considering the importance of digital electronics, extreme significance is to be given for its design and implementation. Conventional design techniques will be discussed in the subsequent section.

1.2 CONVENTIONAL DESIGN TECHNIQUES

1.2.1 Combinational Logic Circuits

A CLC can be realised by using an interconnection of logic gates, or Universal Logic Modules (ULMs) such as multiplexers (muxes) or Reed Muller (RM) logic modules. The most popular methods of designing combinational circuits using gates are i) Karnaugh Map (K map) technique ii) Quine-McCluskey method and iii) algebraic reduction rules. K map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimised. Since it is a visual method, it is not suitable for computer implementation. Though K maps are useful in minimising functions with up to six variables, the design of more than four variables is difficult.

Quine-McCluskey method is suitable for any number of variables and can be easily programmed to run on a digital computer (Seda, 2008). Both the K map and Quine - McCluskey methods produce two level circuits. Additionally, with Quine-McCluskey technique, the CPU usage grows exponentially with the number of inputs. Furthermore, once the prime implicants have been found, the algorithm needs to find the minimal set cover, which is known to be an NP-complete problem.

Simplification by applying algebraic reduction rules is difficult for complex functions and is prone to errors. The reduced circuit depends on the selection and application of appropriate theorems / postulates during the minimisation process. There are no general set of rules to aid that selection. This method depends solely on the designer's knowledge. These conventional techniques do not support the use of gates such as NAND / NOR / XNOR / XOR etc. and ULMs like multiplexers or RM blocks. On implementing circuits using these building blocks, the number of units can be reduced which in turn reduces the power and area. Since replication of the same element reduces the manufacturing cost, design using ULMs is a promising alternative in the design of combinational circuits.

1.2.2 Sequential Circuits

The most generalised model of an SSC includes inputs, outputs and internal states. SSCs are of two types namely, Moore model and Mealy model. They differ only in the way in which the outputs are generated. In mealy machine, the output is a function of both the present state and the inputs, whereas in Moore machine, the output depends on the present state only (Ercegovac, 1985). The two machines are commonly referred to as Finite State Machines (FSMs). The most common examples of Moore machines are counters, which are used in simple digital alarm clocks to computer memory pointers. Best example for Mealy machine is a sequence detector circuit which is useful in many real world applications.

A sequential circuit needs a state table for its specifications, whereas a combinational circuit is completely specified by the truth table. The first step in the design of sequential

circuits is to obtain the Optimal State Assignment (OSA) so that the combinational part needs minimum circuitry. A State Transition Table (STT) is used to evaluate the functionality of the combinational part. The number of possible state assignment grows exponentially with the number of states and hence it is very difficult for assigning the states manually. Thus automated design plays a vital role in this.

1.3 AUTOMATED DESIGN TECHNIQUES

1.3.1 Importance

In conventional design, the efficiency of a system depends on the ability of the designer and is limited to his acquired knowledge. Further, the design space is restricted and varies from designer to designer. Another major drawback in conventional design is that the quality of the circuit is affected by the designer's design habits, imagination, creative thinking and routine. These constraints can be eliminated in automated design techniques. The motivation behind automated design is to evolve more efficient circuits compared to conventional methods. Design automation leads to circuits with minimum number of gates, interconnections, delay and power. Evolutionary design is the popular automated technique for the design of digital circuits.

1.3.2 Evolutionary Design

Research on the evolution of electronic circuits has been started since early1990s. The research area in which evolutionary algorithms are applied in the domain of electronics is termed as Evolutionary Electronics (EE). Application of evolutionary algorithms for the design of reconfigurable electronic circuits is called as Evolvable Hardware (EHW).

The prominent advantage of using evolutionary algorithm for the design of digital circuits is that it allows automatic exploration of a much richer set of possibilities in the design space that are beyond the scope of conventional techniques (Stomeo, 2005). The main inspiration behind evolutionary electronics is the possibility of designing more efficient and simplified circuits compared to conventional methods.

The two major approaches for the synthesis of CLCs using evolutionary algorithms are

i) To obtain an optimised representation of the function using any evolutionary algorithm. E.g.; In Binary Decision Diagrams (BDD), the number of nodes required and hence the complexity of the function representation depends on the order selected for the decision variables. This variable ordering of BDD is done using an evolutionary algorithm (Murukawa *et al*. 1996; Higuchi *et al*. 1997; Thomson and Miller, 1996). Once an optimised representation is obtained, design is done using the algebraic rules of the concerned algebra.

ii) Evolvable hardware approach

The processes involved in EHW are

a) Evaluation process

b) Evolutionary process

c) Evolutionary programming

Evaluation can be performed at gate level or function level. In gate level evolution, gates are used for the evolutionary process whereas in function level approach, high level hardware functions rather than simple logic functions are used as the design elements.

Evolutionary process in which the evolved circuits are built and tested in hardware is termed as intrinsic evolution, whereas extrinsic evolution is referred to as the implementation in software using simulations (Thompson *et al*. 1997; Ali, 2003).

Evolutionary programming involves the use of evolutionary algorithms like GA for the automated design of the circuits. It starts with a particular set of gates fixed by the designer. The required gates and their interconnections are chosen randomly and are allowed to evolve the target functionality. The algorithm decides whether a gate is used or not and how many times a particular gate is used. The synthesised circuit can consist of any set of logic gates. The only criterion is that the generated circuit has to meet the target functionality with minimum hardware.

To be more specific, this approach mimics the nature based on the strategy of survival of the fittest. Each circuit to be designed is considered as an individual in the population represented by a chromosome. An initial population of solutions / circuits is generated randomly. Every individual is assessed to find whether they are fit or not. Fit individuals are selected and genetic operations such as crossover and mutation are applied to obtain a new population. This process is repeated until the fittest circuit (fully functional with minimum hardware) is obtained.

1.3.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are search and optimisation techniques based on the principle of natural evolution. These algorithms have been found to be efficient in solving optimisation and search problems. They are unconstrained search techniques incorporating constraints into their fitness function (Gorai and Pal, 1990; Almaini *et al*. 1992; Miller and Thomson, 1998). With the introduction of reconfigurable devices, the evolvable hardware has gained attention since the last two decades. Any of the popular evolutionary algorithms like Genetic algorithm (GA), Ant Colony Optimisation techniques (ACO), Particle Swarm Optimisation (PSO) etc. can be used as a tool for the design of digital circuits.

Genetic Programming (GP) can be used as an alternative for GA. In GP, a set of functions F and a set of terminals T (constants and variables) are to be defined. Here, the chromosomes are represented as trees with ordered branches in which, the functions are treated as nodes and the terminals as leaves. For example, a set of functions F can be defined as {-, +, /, *} and T as {X, Y, Z}. It adopts the same genetic operators as in GA (Koza, 1999).

Optimum / interesting results can be obtained with evolutionary techniques since the circuits have lesser number of design elements with unusual structure compared to conventional techniques. These circuits can be implemented either in hardware or simulated in software. GA is one of the most efficient evolutionary algorithms used for the evolution of digital circuits. It has shown a high degree of flexibility in dealing with problems that are complex and computationally hard. Coverage of the solution space is more complete and there is less chance for getting stuck at local minimum. It is this feature that makes GA an efficient tool for automated design. This algorithm has received considerable attention regarding its potential as an optimisation technique for complex problems (Ali, 2003). Hence GA has been chosen as the optimisation tool for this work.

1.3.4 Genetic Algorithm (GA)

Genetic Algorithms are search algorithms that are based on the principles of genetics and biological evolution introduced in early1970s by J. Holland. Due to the high potential as an optimisation technique, GA has been widely used in solving complex problems. The problem of implementing the design using GA can be thought of as being equivalent to designing a black box with user defined inputs and outputs as shown in Fig. 1.3.

The black box on presenting the input signals should produce the desired outputs. This black box is encoded into binary chromosomes (circuits) which are generated randomly. On applying genetic operators such as selection, crossover and mutation, new and better individuals are created (Soliman and Abbas, 2003; Coello *et al*. 2000a; Reis *et al*. 2004). The process is repeated until 100 % functional circuits satisfying the corresponding truth table are obtained. Additional fitness is provided to ensure minimum number of components and levels so that the area, power, cost and delay can be reduced.

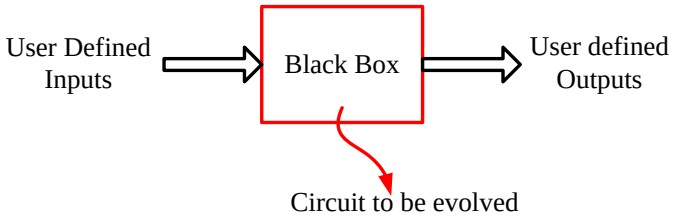


Fig. 1.3 Block Diagram representing evolutionary design

In GA, individuals from the initial population are selected for recombination based on survival of the fittest strategy. Some of the selection techniques include Roulette Wheel Selection technique (RWS), rank method, tournament selection etc. Using any of these techniques, the individuals with more fitness are selected and the weaker ones are discarded. This replicates nature in which fitter individuals have better chances for survival and go for crossover (Goldberg, 1989). RWS technique is adopted in this thesis. Here, each chromosome is given a proportion of the Roulette wheel based on its fitness value. The wheel is spun repeatedly to select the individuals. There is a higher probability for the fittest individuals to be selected multiple times, while unfit individuals have a least chance.

Crossover operation is a recombination process by which a portion of the individuals (parents) are exchanged and two new individuals are created. The newly created individual called offspring possesses the genetic properties of both the parents. The point of crossover is selected at random.

Mutation is an unexpected change in the chromosome pattern in natural evaluation process. In GA, an individual is probabilistically selected from the population and a random gene is altered. Though mutation occurs rarely, it is a main driving force for evolution. The need for mutation is to provide new genes that were not present in the initial population or to replace the genes that were lost from the population during the selection process (Goldberg, 1989; Koza, 1999). This has the effect of increasing the search space and the chance of getting the global optimum solution rather than a local optimum. This may result in a weaker / stronger individual. If it is strong, it tends to survive and the stronger genes are passed on to the subsequent generations and the circuit evolves into an optimum one. Elitist selection assures the survival of the best individual by copying it directly in to the next generation.

1.4 Automated Design of Digital Circuits using GA

In the evolutionary design of digital circuits, as the circuits are generated at random, the final circuits evolved will be unique and completely different from the usual circuits. A fully functional design can be achieved either by using

i) Gates alone

ii) Gates and functional blocks like adders, subtractors etc.

iii) ULMs alone

In the gate level design, any of the two / single input gates or WIRE can be used as design elements. In this work, XOR, AND, OR and WIRE are chosen.

Though the design using gates ensures optimal circuits, the number of components can still be reduced in function level design which involves a combination of gates and high level hardware functions. But, the selection of necessary functional blocks for a particular class of circuits is critical to obtain an optimal circuit and it completely depends upon the designer's capability. The designer should be well aware of the design constraints and should have a prior knowledge of the combinational circuit design and the functionality of each block used.

In VLSI implementation, emphasis is to be given to reduce the manufacturing cost rather than reducing the number of components used (Aguirre *et al*. 2000). Generally the same unit is used repeatedly so as to get the desired functionality even though it leads to large number of gates. When implemented using ULMs alone, 100% functional circuits which are completely independent of the designer's expertise can be generated. As the replication of the same design element reduces the manufacturing cost of VLSI Implementation, this technique is an alternative for the design of CLCs (Aguirre *et al*. 1999; Aguirre *et al*. 2000; Gorai and Pal, 1990).

The focus of this thesis is on CLCs using gates and CLCs using ULMs and the design of SSCs.

1.4.1 CLCs using Gates

As discussed earlier, any of the two / single input gates and wire can be used for the design. Till date, the design of CLC based on GA used linear chromosomal representations. To realise a function of *n* variables, it was assumed that the circuit had

n levels and each level had *n* gates (Louis, 1993). The chromosome was represented in the form of a string or one dimensional array in which any particular gene corresponds to a gate / input. All the genes were arranged in a linear fashion to perform the genetic operations (Coello *et al*. 2000a). Genetic operators such as selection, crossover and mutation were applied over this array. It was found to be difficult to decode or visualise the circuit corresponding to a long string of genes which involves more computational time. This limitation is being addressed in this thesis.

1.4.2 CLCs Using ULMs

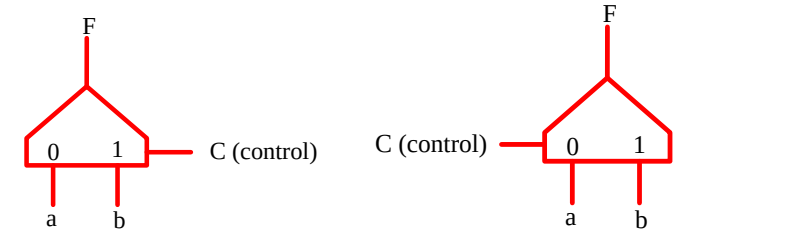
ULMs such as 2-1 mux (binary mux) or 2-1 RM ULM (binary RM ULM) can be used as the basic building block for implementing any CLC in the form of a tree network. Tree network is a graph in which all the nodes are connected either directly or indirectly without forming any closed loop. These are most suitable for VLSI implementation because of the repeated use of a single design element with similar interconnections. Thus the advantage of using ULMs is that, it helps in the ease of design, fabrication and testing of digital circuits.

Design Using Mux

A multiplexer (mux) with *n* selection lines is a combinational circuit that selects data from 2^n input lines and directs it to a single output line. A binary multiplexer has 2 input lines and one control line. They are of “active low” or “active high” denoted as Class A or Class B multiplexers as shown in Figs. 1.4 (a) and (b) respectively. In Class A multiplexer, input labeled as ‘1’ is directed to the output for a high value of control signal and the input which is labeled as ‘0’ is directed to the output for a

low value of control signal (Aguirre and Coello, 2004). Logic for class B multiplexers is exactly opposite to that of class A multiplexers, where the input labeled as '0' is directed to the output for a high value of control signal.

Either class A mux or class B mux or a combination of both can be used. In this work, Class A muxes have been used. The output of a class A mux is given by $F = ac' + bc$ where c′ is the complement of c.



(a) Class A mux (b) Class B mux

Fig. 1.4 Logic symbol of a 2-1 mux

The conventional use of a multiplexer is to route data from one of the *n* sources to a common destination. Besides, it can be used for the realisation of combinational circuits. It is known that, any function can be realised by circuits that exclusively use binary multiplexers by applying Shannon's decomposition technique. The identity used for Shannon's implementation is $F = F'A_{j'} + F''A_j$ where $A_{j'}$ is the complement of A_j. $F' = F(A_{j'})$; it is the value of F for $A_j = 0$; and $F'' = F(A_j)$, is the value of F for $A_j = 1$. Thus, it reduces the original complex problem of order *n* into two simpler ones of order *n*-1. For E.g.; A four variable function when decomposed using an input variable reduces to two sub-functions of three variables. These two sub-functions can be further decomposed into two sub functions of two variables. Repeating the decomposition further with other variables allows the design problem to be reduced further and further to either literals or constants (Almaini *et al.* 1992; Aguirre and Coello, 2004).

For mapping a particular Boolean expansion into its corresponding circuit using binary multiplexers, the variable A_j in the equation, acts as the control signal of the mux at the j^{th} level. Hence any Boolean function is implemented by circuits with only 0s and 1s as inputs and the number of modules needed is 2^n-1. The number of levels or the depth of the array is *n*. The circuit for any three variable function can be realised using Standard Implementation (SI) with 7 units in 3 levels as shown in Fig. 1.5. The inputs to the first level can be 0 or 1. The inputs to the subsequent levels are the outputs of the preceding level. The control signals to all the units in a level are same and all the variables should be used as control signals.

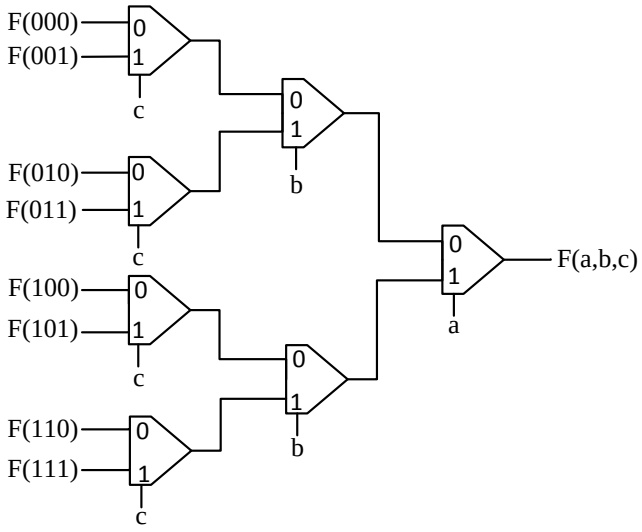


Fig. 1.5 Standard Implementation for a three input function using 2-1 mux

Design Using RM ULM

Most of the researchers have concentrated on designing circuits using mux. For certain applications like arithmetic circuits, error detection circuits etc., which are XOR based, RM representation (AND- XOR logic) is advantageous. This is due to the fact that XOR based circuits are easier to test and requires lesser number of interconnections.

(Chaudhary and Chattopahyay, 2008). On the contrary, XOR gates are slow and require larger area in comparison to OR gates. But with the advancement of new technologies and with the advent of various FPGA devices, XOR / XNOR implementation of circuits have become easier. (Faraj, 2009a; Vijayakumari *et al*. 2015a; Vijayakumari *et al*. 2015b).

The logic symbol of a 2-1 RM ULM is shown in Fig. 1.6, whose output is given by $F = a \oplus b.c$

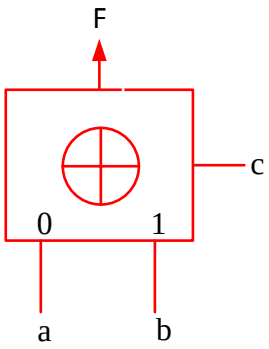


Fig. 1.6. Logic symbol of a 2-1 RM ULM

Basic theorems involved in the design using RM ULM are

$$X \oplus X = 0;\ X \oplus X' = 1 \tag{1.1}$$

$$X \oplus 1 = X';\ X \oplus 0 = 0 \tag{1.2}$$

$$X + Y = X \oplus Y \oplus XY \tag{1.3}$$

The RM representations may be shorter with a reduced number of product terms leading to simpler circuits for certain applications like arithmetic operations, parity checkers etc. Logic functions that cannot be minimised well in Sum of Products (SOP) forms can often be implemented in the RM domain, leading to reduced size and power consumption (Al Jassani *et al*. 2010). Similar to Shannon's decomposition

technique for mux, Davio decomposition technique is used to reduce the functions in AND-XOR form into sub functions. Expressions in terms of sub functions make the implementation of Boolean functions simpler. Based on Davio decomposition technique, a hardware circuit for a function of *n* variables can be implemented using the identity, $F = F' (A_j)' \oplus F'''(A_j)$, where F′ and F‴ are functions of *n*-1 variables. As in Shannon's theorem, applying decomposition repeatedly with each of the variable A_j allows the synthesis problem to be reduced further and further to either literals or constants (Vijyakumari *et al*. (2014)).

In Standard Implementation (SI) technique using mux and RM ULM, units in the same level share the same variable as control signal. Furthermore, a variable assigned for a particular level as control signal in one level cannot be used as control signal for any other levels. Automated design technique looks into the possibilities of reducing the 2^n-1 elements.

1.4.3 Synchronous Sequential Circuits

In SSCs, the output at any given time is a function of both present and past inputs. The behavior of an SSC can be represented by an FSM, which is a mathematical model of a sequential circuit with discrete inputs, discrete outputs and internal states. There are two types of FSM - Moore machine and Mealy machine as mentioned in Section 1.2.2. These machines can be realised using any flip flop along with suitable CLC. In CLC design, a truth table completely specifies the circuit, whereas in an SSC a state table specifies the circuit (Ercevoc, 1985). A unique binary code is to be assigned to each of the states of the FSM. If the number of states is *n*, then the number of state variables **s** is the smallest integer that is equal to or greater than $|\log_2 n|$. Then the total number of possible states is equal to 2^s.

The assignment process decides which of these 2^s codes must be assigned to any particular state in the FSM. The total number of possible encodings is given by (Ali, 2004)

$$T(n,s) = \frac{2^s!}{(2^s-n)!} \tag{1.4}$$

Hence, it is necessary to find the state assignment which results in circuits with minimum hardware. Finding a relationship between the states and bit strings which results in minimal cost is referred to as the problem of OSA. Hence an automated design technique which can find an OSA is important in the design of SSCs. Once an OSA is obtained, the State Transition Table (STT) can be prepared. Based on the STT, the combinational part of the SSC can be generated using gates or universal building blocks such as binary multiplexers / binary Reed Muller blocks as mentioned in previous section.

1.5 TOOLS / PLATFORM

The experiments on the design of digital circuits have been performed on Intel core i5 processor with 2 GB RAM and 2.5 GHz frequency. MATLAB R2012a is used as the software tool. The computational time needed to evolve the circuits depends on the function to be implemented, size of the truth table and type of the fitness function used for optimisation. The circuits evolved using ULMs have been synthesised on FPGA Spartan 3 XC3S400 device using Xilinx ISE 14.2.

1.6 BENCHMARK FUNCTIONS USED

Benchmark functions are usually complex functions which are difficult to simplify using conventional reduction techniques. On mapping these functions into K map,

there will not be any two adjacent *1s* to group together. Hence, K map / Quine - Mc Cluskey method cannot be applied for the simplification. Applying algebraic reduction techniques on these functions is very complicated and is prone to errors. Moreover, the designed circuit by algebraic reduction technique varies from designer to designer and hence it is not compared with the circuits by automated design.

The benchmark functions used for the validation of the proposed methods in this thesis are listed in Table 1.1.

Table 1.1 Benchmark functions used for validation of the proposed methods

Sl No.	**Name**	**Function**
1.	Majority 3	F (a, b, c) = Σ m (3, 5, 6, 7)
2.	4 bit odd parity checker	F (a, b, c, d) = Σ m (1, 2, 4, 7, 8, 11, 13, 14)
3.	xor5	F (a, b, c, d, e) = Σ m (1, 2, 4, 7, 8, 11, 13, 14, 16, 19, 21, 22, 25, 26, 28, 31)
4.	6one135	F (a, b, c, d, e, f) = Σ m (1, 2, 4, 7, 8, 11, 13, 14, 16, 19, 21, 22, 25, 26, 28, 31, 32, 35, 37, 38, 41, 42, 44, 47, 49, 50, 52, 55, 56, 59, 61, 62)
5.	6one0246	F(a, b, c, d, e, f) = Σ m (0, 3, 5, 6, 9, 10, 12, 15, 17, 18, 20, 23, 24, 27, 29, 30, 33, 34, 36, 39, 40, 43, 45, 46, 48, 51, 53, 54, 57, 58, 60, 63)

1.7 MOTIVATION

In conventional design, the quality of the designed circuit depends on the designer's capability and it varies from designer to designer. Automated design techniques are algebra independent and the designer's expertise is not significant. Several heuristic algorithms like GA permits the utilisation of a large search space which humans cannot

exploit. It can freely explore the space of all possible circuits, thereby evolving circuits which prove to be 100% functional with minimum hardware.

In the automated design of CLC using gates, the existing technique uses linear chromosomal representation and corresponding genetic operators such as crossover and mutation are applied. This involves more computational time which has to be reduced.

In the design of CLC using ULMs, there are no specific reduction techniques available other than Shannon's / Davio decomposition techniques. The circuits designed by these techniques need a lot of hardware which involves more power consumption, area, delay and cost. Sophisticated electronic equipments like palmtop game / media consoles, laptop computers, cell phones etc. have got an increasing demand these days. Hence it is desirable to have the circuits with minimum hardware, area, power and delay.

Thus, the motivation behind the study of automation of digital circuits is

- need for developing automated techniques for optimal design
- to address the issues regarding increase in computational time.

1.8 OBJECTIVES

The objectives of this thesis are to efficiently automate the design of

- Combinational Logic Circuits using
  - i) Gates
  - ii) Binary ULMs and
- Synchronous Sequential Circuits.

1.9 CONTRIBUTIONS OF THE THESIS

The contributions of this thesis towards the automation of digital circuits are

- A new faster technique for the gate level design of CLCs which ensures minimum computational time
- Two new GA based techniques for the design of CLCs using ULMs
- GA based OSA technique for SSCs
- Implementation of the combinational part of SSCs using gates / binary ULMs

1.10 OUTLINE OF THE THESIS

The proposed thesis is organised in 6 chapters.

Chapter 2 deals with the review of research done in the area of evolutionary design of combinational and sequential circuits.

Chapter 3 discusses the aspects of GA based design of combinational circuits using gates. A 2D representation of chromosomes and the corresponding 2D crossover and mutation technique for the design of combinational logic circuits is proposed. Gates such as XOR, AND, OR and WIRE are used to evolve the circuits. Circuits including benchmark functions with inputs up to 6 variables have been evolved. A comparison of the convergence time between the proposed technique and the conventional method has been made.

Design of combinational circuits using universal building blocks such as binary mux and binary RM ULM is discussed in chapter 4. Two new GA based techniques are proposed and the evolved circuits are synthesised and implemented on Xilinx FPGA Spartan3 family. Results are validated using benchmark functions.

Chapter 5 covers the design of synchronous sequential circuits. A modified GA has been proposed to obtain the OSA which determines the circuit complexity of an SSC. Once OSA is done, the combinational part was optimised using i) gates and ii) universal building blocks such as 2-1 mux and 2-1 RM ULM.

Chapter 6 contains concluding remarks, which review the major contributions of this work and its future scope.

CHAPTER 2

LITERATURE REVIEW

This chapter explores the history and important milestones in the design automation of digital circuits. It discusses the need for investigations based on the efficiency of evolved circuits. Evolutionary algorithms are used to synthesise and optimise EHW. The need for evolutionary design is discussed and the research work done in this area is explored.

2.1 EVOLUTIONARY DESIGN

To minimise logical functions up to 6 variables, K-map is a very efficient graphical tool. Since this method is based on the visual recognition of adjacent cells, it is not suitable for automated processing with computers. Quine-McCluskey method can be used for any number of variables and is suitable for computer implementation. With this technique, the CPU usage grows exponentially with the number of inputs. Furthermore, once the prime implicants have been found, the algorithm needs to find the minimal set cover, which is known to be an NP-complete problem. Functions can be minimised by applying algebraic reduction techniques, but it is very difficult to optimise complex functions and is prone to errors. Thus there was an urgent need for computer oriented design for minimising the circuit.

In 1990s the research on EHW started, the objective was to evolve a fully functional circuit from a randomly generated set of circuits with the help of evolutionary algorithms. Later on, attempts were made to reduce the complexity / number of gates needed to realise the circuits. The quality of the evolved circuits was estimated based

on the number of gates used for implementation. The principle of evolutionary design and its applications were well discussed in (Miller *et al*. 2000a; Miller *et al*. 2000b).

Sekanina, (2009) reviewed the fundamental principles of evolutionary algorithms, pointed out the major drawbacks and came up with some applications of evolvable hardware. Haddow and Tyrrell, (2011) described in their work, the applications of evolvable hardware and the advantages of using them. The work mentioned that the major challenge to be addressed was scalability and the authors proposed various alternatives like divide and conquer, function level evolution etc. Though the paper discussed the above mentioned issues, no detailed study was reported to support this. Yan *et al*. (2011), investigated the application of cultural algorithms (an evolutionary algorithm in which the evolution adapts to their environment at a higher rate than biological evolution based on genetic inheritance alone) in the design of electronic circuits. No comparative study was made with other evolutionary algorithms.

Theory of EHW can be applied to combinational as well as sequential circuits. A lot of work has been done in the area of design of combinational circuits compared to sequential circuits.

2.2 DESIGN OF COMBINATIONAL CIRCUITS

2.2.1 Design Using Gates

Optimisation tools like GA, ACO technique, PSO technique etc. can be applied for the design of CLCs. GA is found to be more suitable for the design of digital circuits compared to the rest because of its inherent advantages as mentioned in Section 1.3.3. Most of the research work reported in literature concentrates on the design of CLCs using gates.

(Louis *et al*. 1993) is the earliest source that reports the use of GAs to design CLCs. The linear chromosomal representation introduced by Louis is still popular and is used by many researchers. In 1996, Koza *et al.* proposed a genetic programming approach to design CLCs. His research was concentrated on the generation of fully functional circuits. Their focus was not on achieving optimal circuits. In all the above mentioned work, the chromosomes were represented in the form of a binary string. The size of the binary string increased exponentially with the number of inputs/outputs which leads to increase in computational time. Coello *et al*. (1996) presented a GA based approach in which, the individuals can be represented in any number system such as octal, decimal, hexadecimal etc. which was proved to be effective in larger circuits. Miller (1999) used evolutionary technique for designing a multiplier circuit for multiplying two three bit numbers. In (Kalganova, 2000*a*), the evolutionary design had been extended for the design of multiple valued logic. A three valued one bit adder was implemented and it was the first article to work on multiple valued logic.

In (Coello *et al*. 2000*a*; Coello *et al*. 2000*b*; Coello *et al*. 2000*c*), an effort has been made on the design of CLCs to minimise the number of gates by introducing certain modifications in the conventional GA. Coello *et al*. (2000*d*) proposed the design of CLCs at gate level using ACO technique and compared with the results of GA based design. The authors reported that the results were similar to GA based design and much better than human based design with K-map and Boolean algebraic rules. Hounsell and Arslan (2000) proposed GA for high performance arithmetic circuits and used macro blocks in addition to gates for the evolution of CLCs. The selection of this block is critical so as to evolve circuits with easily. Shanthi *et al*. (2002) discussed an evolutionary approach towards the design of CLCs in detail. This work demonstrated

three different levels in which fault tolerance can be supported in the evolutionary design of digital circuits.

Reis *et al.* (2004) extended the technique proposed by Louis for designing CLCs using gates. The author could support this technique with circuits up to 4 variables. Slowik and Bialko (2008) discussed the state of the art, main problems, challenges and future trends of evolutionary algorithms for the design of combinational circuits. He pointed out the scalability issues in the case of circuits with large number of inputs / outputs. Since evolutionary design is based on generate and test model, as the number of inputs increases, the number of possible output combinations also increases. It explained the need for decomposing the desired circuits into several less complex sub circuits and to design each of them independently. Reis and Machado (2007) dealt with the implementation of logic circuits using the evolutionary algorithms like GA and PSO and Memetic Algorithm (MA). Gate level implementation was adopted for the design. Each algorithm was analysed based on the complexity of the CLC. Benchmark functions were not considered for validation in their work. A Genetic Programming approach for the design of Combinational logic circuits using gates was proposed in (Karakatic *et al.* 2013). The results obtained were compared with the conventional ones for functions up to four variables.

Vassilev and Miller (2000) discussed the problem of scalability in detail and implemented a 3 bit multiplier in which sub circuits were used as building blocks. These sub circuits were evolved separately and found to be much faster reducing the scalability issues. It was reported that the selection of sub circuit was critical and the

designer should be well aware of the design rules. The disadvantage of this method is the increase in the number of gates as the number of sub circuits needed is more.

In (Kalganova, 2000*a*), a function level approach for evolvable hardware is discussed where high level functions such as adders, multipliers, etc. are used as primitive functions instead of simple logic functions. The issue of scalability in evolutionary methods is discussed and an extrinsic EHW approach has been proposed for the design of a number of functions using gates. Liu *et al.* (2006) suggested a method to reduce the time of computation by applying a modified mutation technique. Sagar and Vathsal (2013) reported the design of combinational circuits based on three different evolutionary algorithms (GA, ACO and PSO) using gates. These approaches were compared with each other for the speed of convergence and quality of solution and GA was reported to be superior.

In all the literature cited above design of CLC was done using logic gates and involves the scalability issues and increase in computational time. Computational time can be reduced by adopting different types of chromosomal representations.

Circuits can be generated by replacing the gates by Universal building blocks such as multiplexers or Reed Muller Logic Blocks (Yau and Tang, 1970). Since only one type of design element is used, manufacturing cost can be reduced. It is established that any Boolean expression of n variables can be realised by using 2^n-1 binary multiplexers (Aguirre *et al.* 1999). Several works have been done to reduce the number of units so as to reduce the cost, delay, power etc.

2.2.2 Design Using Multiplexers

The use of multiplexers as ULMs for realising Boolean functions has attracted researchers since 1970s. Most of the works were concentrated on obtaining minimal circuits using linear programming, numerical methods, etc. (Yau and Tang, 1970). Multiplexer is a circuit which selects one out of many input lines. A 2-1 multiplexer (2-1 mux) is a circuit having 2 input signals, one select (control) signal and one output signal. Any of the input signals can be routed to the output based on the select signal. A brief introduction of multiplexers is given in Section 1.4.2.

Pal (1986) formulated an algorithm based on ratio parameters for realising Boolean functions with a single multiplexer of minimum size. Ratio parameter is the ratio of number of ones to number of zeros in one column of the minterm table of the logic function. As an extension of this work, circuits were implemented using a cascade network of multiplexers (Gorai and Pal 1990; Pal 1986). This method terminates if the function is not realisable in the cascade form and was not based on the evolutionary algorithm. Almaini *et al.* (1992) proposed a programmed algorithm for the synthesis of CLCs using a cascade or a combination of cascade and tree network of multiplexers. The algorithm attempted level by level optimisation by selecting the control signals which results in minimum number of continuing branches. The algorithm was not based on evolutionary principles.

In (Aguirre *et al.* 1999 and 2000), a genetic programming approach has been made for the logic synthesis of Boolean functions using multiplexers. Simple functions were chosen for realisation. Later, in (Aguirre and Coello 2004) CLCs were synthesized with multiplexers using genetic programming and the results were superior compared

to SI technique. Analysis for design metrics such as area / delay was not performed. Even though the number of units required was less, the circuits were still not optimal.

Synthesis of combinational circuits using multiplexers was dealt in (James *et al*. 2006). An optimisation algorithm was proposed for the realisation of Boolean functions using universal building block such as 2-1 multiplexer. This was not based on any evolutionary algorithm. Functions up to five variables were reported. Moreover, benchmark functions were not used for validation. In (Li *et al*. 2008) a controller was designed to generate select signals dynamically so that power consumption of the mux tree is minimised. As the controller itself consumes some power, the power required by the controller might be more than the power consumption of the circuit in the case of smaller circuits, which is a disadvantage.

The other universal logic block, RM ULM has been widely used by researchers to overcome the scalability issues in the design using gates. The subsequent section gives a brief survey about the RM ULM in the realisation of digital circuits.

2.2.3 Design Using RM ULM

Research on synthesis and optimisation of logic circuits in RM domain based on XOR logic is still producing new and useful techniques (Faraj, 2009*a* and 2009*b*). The reasons for this include: AND / XOR or OR / XNOR logic requires fewer terms than Sum of products or Product of sums respectively (Al Jassani *et al*. 2010). Moreover, testing of XOR / XNOR based circuits is easy and efficient (Drechsler *et al*. 1999).

Binary Decision Diagram (BDD) is a well- known technique for the optimisation of digital circuits. Decision Diagrams were first proposed by Akers *et al*. (1978) and

further modified by Bryant (1985). It is a graphical way of representing switching functions and provides an alternative optimisation technique (Yanagiya, 1995). The approach, although useful to test functional equivalence (generation of the same truth table) and other circuit properties, it does not fully minimise a circuit (Almaini and Zhuang, 1995*a*; Almaini *et al*. 1995*b*). In (Xia *et al*. 2003*a* and 2003*b*) a frame for power dissipation estimation was presented and an algorithm was proposed to select the polarity so as to minimise the power and area of circuits.

Shahana *et al*. (2005), proposed an optimisation algorithm to implement logic functions using RM-ULMs. Functions up to four variables were considered and no benchmark functions were used for validation of results. Oh and Almaini (2007) dealt with the implementation of Boolean functions using 2Variable ROBDD technique. Boolean functions were implemented using RM ULMs and it was observed that the circuits could still be reduced. Based on the literature survey, it was observed that there is a scope for further reduction in hardware.

Pradhan and Chattopadhyay (2008) used GA to select the polarities of the variables of the AND-XOR network. The polarities are selected based on the optimisation of area, dynamic power and leakage power of the resulting circuit. Chaudhary and Chattopadhyay (2008) proposed a GA based scheme for implementation of Fixed Polarity Reed Muller (FPRM) functions using AND - OR / XOR with minimum area and power. No evolutionary technique was applied for implementation.

Much research has been done on the evolutionary design of CLCs, but research on the synthesis of sequential circuits using EHW has started recently. Though a number of

authors have given valid contributions, its research is still in the infant stage (Al Jassani *et al*. 2011; Tao *et al*. 2013).

2.3 DESIGN OF SEQUENTIAL CIRCUITS

EHW has been applied in various fields such as Digital and analog circuit design; Control and robotics; Communication systems; Pattern recognition; Prediction application etc. As EHW needs no knowledge of circuit design, it provides an excellent way for sequential circuit design. Size of the circuit and cost are the major issues in digital design. The focus of researchers was mainly towards reduction in the number of gates / design elements so as to minimise the on-chip area and cost (Ali *et al*. 2004).

An FSM is defined as a mathematical model of a system with discrete inputs, discrete outputs and a definite number of internal states. The states of a system determine its behavior on the application of subsequent inputs (Miller, 1999). Assigning binary codes for each of the states of the FSM is termed as state assignment.

The complexity of the combinational component of an FSM depends very much on the state assignment and selection of memory elements. Finding the OSA is the most important optimisation problem in the automated design of sequential circuits since it has a key influence on the power, area, speed and testability of the circuit (Ali, 2003; Czerwiński and Kania, 2010). The problem of state assignment has been studied extensively by several researchers to aid in the design of complex sequential circuits. Mc-Cluskey and Unger (1959) derived a formula to find the number of state assignments possible for a given function. Stearns and Hartmanis (1961) and

Story *et al*. (1972) suggested several methods to arrive at an OSA but none of them were based on any evolutionary algorithm. De Micheli *et al*. (1985) proposed a computer aided design tool for OSA.

In (Hartmanis, 1961), state assignment was based on the technique of partition and decomposition and was limited to state machines having useful closed partitions compared to other machines. Amaral *et al*. (1995) used GA to obtain the state assignment of FSMs. But his method took quite a long time to converge. Almaini *et al*. (1995) used GA for the generation of OSA for a synchronous FSM and they could evolve results as good as MUSTANG.

Ahmad and Dhodhi (2000) proposed a method to find the OSA which is based on Mean Field Annealing which combines the characteristic of simulated annealing and Hopfield neural network. They claimed to have superior results compared to NOVA and Mustang. Digalakis and Margaritis (2001) proposed a GA based technique for state assignment. A new selection mechanism was introduced and suitable cross over and mutation operators were proposed. Experimental results on the performance evaluation of some benchmark functions were discussed.

Soliman and Abbas (2004) proposed a GA based approach for the design of a 3 bit up counter. Instead of evolving the combinational part and the memory elements separately, the entire SSC was evolved as a single unit. The authors reported only one experiment as validation. In (Al Jassani *et al.* 2011) a method based on MOGA to obtain the OSA was proposed. The ESPRESSO tool was used to optimise the combinational parts of the sequential circuits.

Usually the number of logic elements needed was considered as the design metric, but in the case of VLSI implementation, reducing the manufacturing cost is the major criterion rather than minimising the number of units (Sarrafzadeh and Wong, 1996). Soleimani *et al*. (2011*a*) and (2011b) dealt with the design and optimisation of synchronous sequential circuits. They used D Flip flops and the design was based on GA. It was reported that average generations could be reduced due the limited search space. In (Tao *et al*. 2013), evolutionary design of synchronous sequential circuits based on a module level three stage approach was proposed. GA was used to obtain the state assignment in the first stage and a number of circuits were evolved in the second stage using Genetic programming. In the final stage, complexity of circuit was reduced by re-evolution. Sequence detectors, modulo-*n* counters and other benchmark circuits were used to test the three-stage approach.

In all the above work, the combinational part of FSM was realised using gates as the design element. The time of computation needed is more as the length of chromosome increases. As mentioned in the last paragraph of Section 2.2.1, computational time can be reduced by adopting different chromosomal representation and the corresponding genetic operators such as cross over and mutation. It was also observed that there is a need for an efficient method for evolving the OSA.

2.4 SUMMARY

In this chapter, a detailed review on various evolutionary design techniques of digital circuits has been done. Most of the research till date is on the design of combinational circuits, while the research on SSCs is still in the infant stage.

As far as CLCs are concerned, most of the works were concentrated on the design using gates. The computational time involved is more as the number of inputs / outputs of the function increases. Hence there is a need for reducing the computational time for which a new faster technique has been proposed in this thesis.

With the advent of reconfigurable devices, design using universal building blocks has become necessary. The advantage of using ULMs for the design of CLCs is elaborated in Section1.4.2. With the introduction of VLSI circuits, it has become a tough task for the designer to pack more functionality on a chip of smaller area. Moreover, the delay associated with the circuit should be less to have a faster operation of the system. In this work, two new techniques have been proposed for the implementation of circuits using building blocks such as 2-1 mux and 2-1 RM ULM with GA as the optimisation tool, so as to evolve circuits with minimum area and delay.

OSA is an important prerequisite for the design of FSMs. Earlier several tools were used to obtain the state assignments corresponding to minimum circuit complexity. Later on search algorithms have been introduced to obtain OSA which is responsible for having circuits with minimum hardware. The combinational part of FSM can be implemented using gates and ULMs. Implementation using ULMs is really an advantage as the replication of the same design element reduces the manufacturing cost. In this work, for gate based design, to have faster convergence the above mentioned method has been used and a new crossover technique has been proposed for the OSA. Moreover, the combinational part has been evolved using i) gates and ii) ULMs.

CHAPTER 3

COMBINATIONAL LOGIC CIRCUIT DESIGN USING GATES

3.1 INTRODUCTION

With the increasing need of high quality digital circuits in everyday life, new methodologies have to be introduced for its design. The existing popular methods for the design of CLCs using gates include i) K map technique - a graphical representation of Boolean functions ii) Quine-McCluskey method - a tabular method and iii) algebraic reduction rules as mentioned in Section 1.2.1. Design of electronic circuits is usually done by experienced designers having meticulous knowledge regarding the design rules and reduction techniques. But such methods entail limitations such as inexperience or lack of adequate knowledge which can affect the quality of the circuits designed. Another drawback of human-based approach is that the designer's line of imagination and design habits reflects in the performance of the designed circuits. These constraints can be eliminated in automated design techniques. Evolutionary Design (ED) is the most popular automated design technique for digital circuits. It uses evolutionary algorithms (search algorithms) to realise functions that are not achievable by the conventional design techniques.

Genetic Algorithm (GA), an efficient search technique based on the principle of genetics and natural selection, is used as the optimisation tool for evolving the circuits. The problem of implementing the design using GA can be considered as being equivalent to designing a black box with user defined inputs and outputs as shown in

Fig. 1.3 in chapter 1. The system should generate a digital circuit that satisfies the specified truth table with minimum number of gates in lesser time. This black box is encoded into chromosomes (circuits) which are generated randomly. On applying genetic operators such as selection, crossover and mutation, new and better individuals are created. The flow chart for the GA is shown in Fig. 3.1.

GA for a particular problem should have the following components:

1. A representation for potential solutions (encoding)
2. An initial population of the potential solutions (Usually generated at random)
3. An evaluation function to measure the fitness of individuals (rating of the solutions)
4. Genetic operators that alter the composition of children (crossover and mutation)
5. Suitable values of parameters of GA like size of population, probabilities of the genetic operators and number of generations.

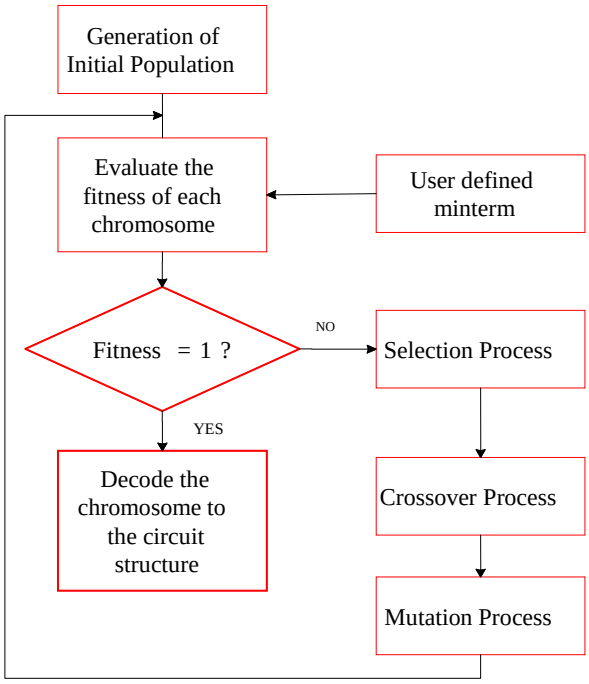


Fig. 3.1 Flow chart of GA

The most important aspect of GA is the encoding of solutions. i.e., the problem of representing the circuits as chromosomes. It affects the population size and hence the convergence time of GA.

3.2 CHROMOSOMAL REPRESENTATION (ENCODING)

The chromosomal representation proposed by (Louis, 1993) is still popular and is generally used for the evolution of digital circuits. He used binary representation for the genes. Later on, Coello *et al*. (1996) suggested a GA based approach in which the chromosomes (individuals) can be represented in any number system such as octal, decimal, hexadecimal etc. which was proved to be effective in larger circuits. Till date chromosomes are represented in a linear fashion as a string of characters using any number system. A brief overview of the linear chromosomal representation is explained in next section.

3.2.1 Linear (Conventional) Chromosomal Representation

To realise a function of *n* variables, it was assumed (Coello *et al*. 2000b) that the circuit had *n* levels and each level had *n* gates as shown in Fig. 3.2, where, G_{11}, G_{12} , …G_{nn} are the various two input gates generated at random. The inputs to the gates in the first column (first set of cells) were obtained from the truth table and all other gates receive their inputs from the previous level. The circuit was encoded into a string of genes where each gene corresponds to a gate and its corresponding inputs. For example, a three variable function can be represented using 9 genes (3 gates / level) as shown in Fig. 3.3. If 2 bits were used to represent the type of gate / input, then the number of bits needed to represent a single gate would be 6 and

the length of chromosome would be 54, whereas for a six variable function it goes up to 216. If the number of bits used was 3, it would become 81 and 324 respectively. Thus, with linear chromosomal representation the chromosomal length increases proportional to the number of variables, which leads to increase in computational time which is one of the issues taken for investigation in this thesis. To address this issue, a 2D chromosomal representation is proposed.

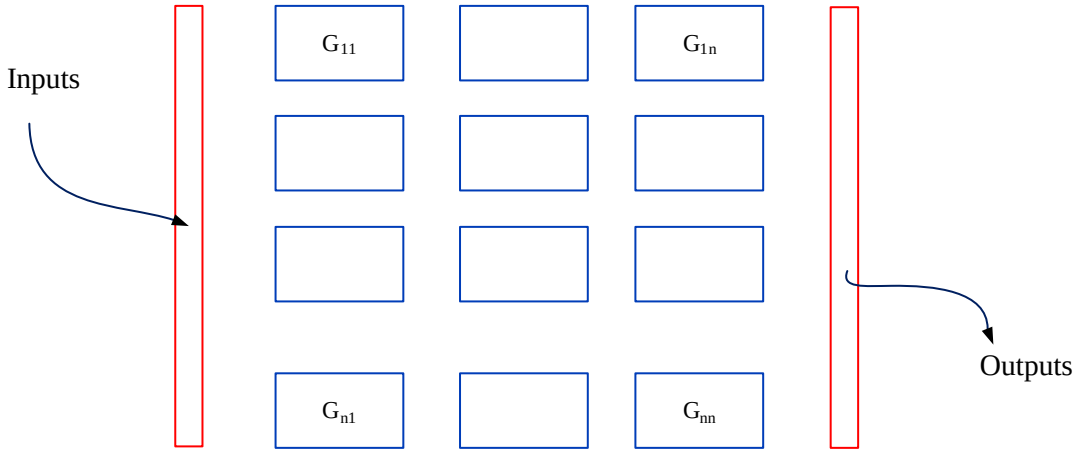


Fig. 3.2 Array of gates for the realisation of a CLC

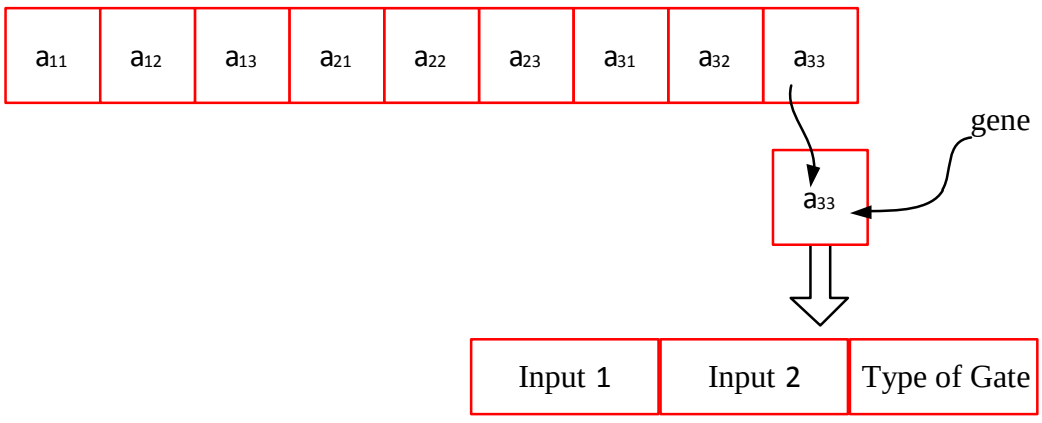


Fig. 3.3 Representation used for encoding of the circuit

3.2.2 2D Chromosomal Representation (Encoding of the circuit)

The author proposes an approach in which the circuit is retained as an array without converting it to linear form, provided suitable crossover and mutation techniques are formulated. Encoding of a three input circuit into its corresponding 2D chromosome is illustrated in this section. If *m* bits are used to represent the gates / inputs, an *n* input circuit is represented by an *mn* x *mn* matrix where the first column represents the inputs to the gates of first level, the second column represents any two input gate / wire in that level and the subsequent columns follow the same pattern. A three input circuit 'A' shown in Fig. 3.4 is encoded as shown in Table 3.1.

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}	A_{36}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}	A_{46}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}	A_{56}
A_{61}	A_{62}	A_{63}	A_{64}	A_{65}	A_{66}

Fig. 3.4 A three input circuit with three levels and three gates / level

Table 3.1 Encoding of gates and corresponding inputs for 2D chromosomal representation

Inputs		**Gates**		
Inputs to gate1	A_{11} A_{21}	Gate1	A_{12} A_{22}:	
Inputs to gate2	A_{31} A_{41}	Gate2	A_{32} A_{42}:	Level 1
Inputs to gate3	A_{51} A_{61}	Gate3	A_{52} A_{62}	
Inputs to gate 4	A_{13} A_{23}	Gate4	$A_{14}A_{24}$	
Inputs to gate 5	A_{33} A_{43}	Gate5	A_{34} A_{44}	Level 2
Inputs to gate 6	A_{53} A_{63}	Gate6	A_{54} A_{64}	
Inputs to gate 7	A_{15} A_{25}	Gate7	A_{16} A_{26}	
Inputs to gate 8	A_{35} A_{45}	Gate8	A_{36} A_{46}	Level 3
Inputs to gate 9	A_{55} A_{65}	Gate9	A_{56} A_{66}	

Usually the chromosomes are represented in binary. In this thesis, gates are represented in binary and inputs are represented by integers so as to obtain a square matrix for the chromosomal representation. The square representation is adopted to obtain better and easy visualisation of the circuit.

As an example, consider two individuals (chromosomes) A and B generated randomly for a three input function as shown in Figs. 3.5 (a) and (b).

0	0	0	1	0	1
1	0	2	0	0	0
2	0	1	0	1	0
1	1	2	1	2	1
2	1	2	0	2	1
1	0	0	1	0	1

(a) Individual A

0	0	0	1	1	1
1	0	1	1	2	0
1	1	1	0	0	1
2	0	0	0	1	0
0	0	2	1	2	1
1	1	1	0	1	1

(b) Individual B

Fig. 3.5 Randomly generated individuals

The encoding of inputs and gates are shown as in Table 3.2.

Table 3.2 Encoding of gates and the corresponding inputs

(a) Encoding of inputs

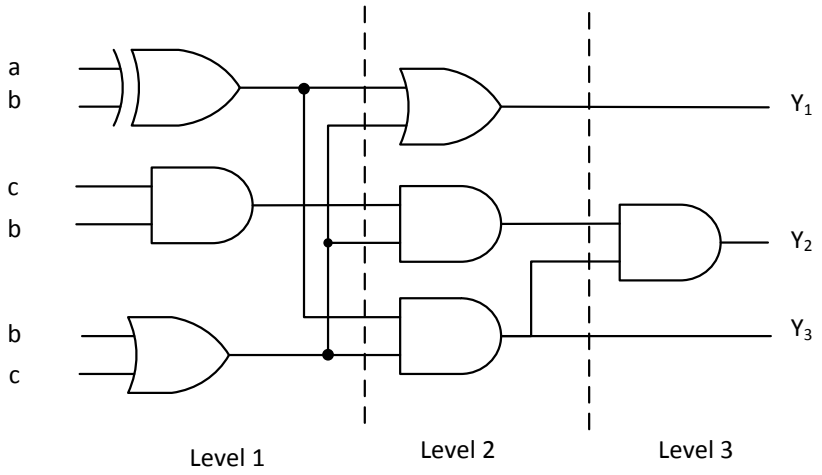
Inputs	
Assignment	Corresponding Inputs
0	a
1	b
2	c
3	d
4	e
5	f

(b) Encoding of gates

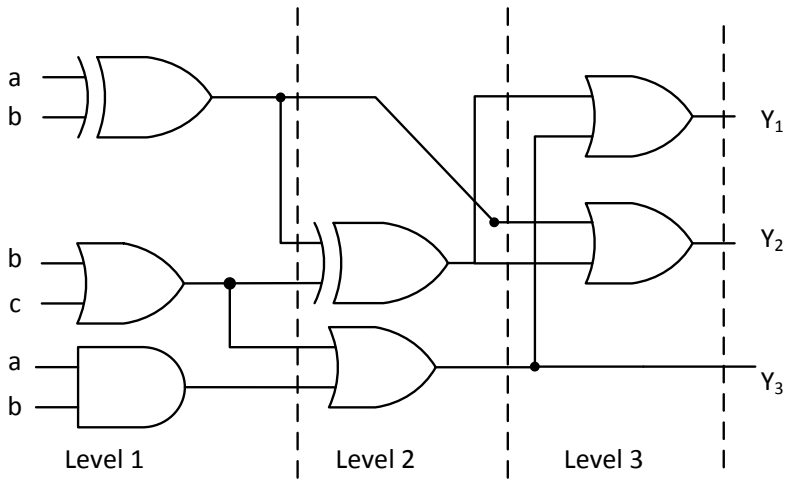
Gates	
Binary code	Corresponding gates
00	XOR
01	AND
10	OR
11	WIRE

If the gate type is a WIRE, then its input is considered to be the top element of the chromosome corresponding to the input combination of that gate and if both the inputs of an AND / OR gate are same, then that gate is replaced by a WIRE.

As per the above encoding, the corresponding circuits for individuals A and B are shown in Figs. 3.6 (a) and (b) respectively.



(a) Circuit A



(b) Circuit B

Fig. 3.6 Chromosomes for a 3 input function generated randomly

For this 2D representation, suitable 2D crossover and mutation techniques are to be developed for the optimisation.

3.3 OPTIMISATION USING GA

The genetic operators such as selection, crossover and mutation have to be applied on the chromosomes for optimisation using GA. As mentioned in Section 1.3.4, using any selection technique, individuals with more fitness are selected for crossover and the weaker ones are discarded. RWS technique is used in this thesis. Crossover operation is the process by which some chromosomal patterns / genes are exchanged between two parents so as to create the offsprings for next generation. Mutation is an unexpected change in the chromosomal pattern which occurs rarely so as to evolve better individuals.

Since 2D chromosomal representation has been proposed for the individuals, 2D crossover and mutation techniques are also developed.

3.3.1 2D Crossover Technique

2D crossover technique is illustrated in this section assuming that the possible solutions (circuits) are encoded in to an array of size $2n \times 2n$, where n is the number of variables involved in the function. Consider two parent individuals A and B. Parents A and B are three input circuits and the corresponding chromosomes are shown in Fig. 3.7. To perform the crossover operation, a portion of the chromosome of the parent individuals must be swapped. To achieve this, the region of crossover is to be identified. This can be

done using suitable mask matrices which when multiplied by the original matrix identifies the region to be swapped (Vijayakumari and Mythili, 2012).

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}	A_{36}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}	A_{46}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}	A_{56}
A_{61}	A_{62}	A_{63}	A_{64}	A_{65}	A_{66}

B_{11}	B_{12}	B_{13}	B_{14}	B_{15}	B_{16}
B_{21}	B_{22}	B_{23}	B_{24}	B_{25}	B_{26}
B_{31}	B_{32}	B_{33}	B_{34}	B_{35}	B_{36}
B_{41}	B_{42}	B_{43}	B_{44}	B_{45}	B_{46}
B_{51}	B_{52}	B_{53}	B_{54}	B_{55}	B_{56}
B_{61}	B_{62}	B_{63}	B_{64}	B_{65}	B_{66}

Fig. 3.7 Parents A and B selected for crossover

To obtain the mask matrices, a set of 4 random numbers R1, R2, C1, C2 are generated where R1 and R2, C1 and C2 are numbers between '1' and '2*n*' and they specify the start and end rows / columns of a sub matrix in the parents respectively. The genes within this sub matrix will be swapped between the parents. The sub matrices to be swapped are shown in Fig. 3.8. (Corresponding to R1 = 2, R2 = 6, C1 =2 and C2 =5 generated randomly).

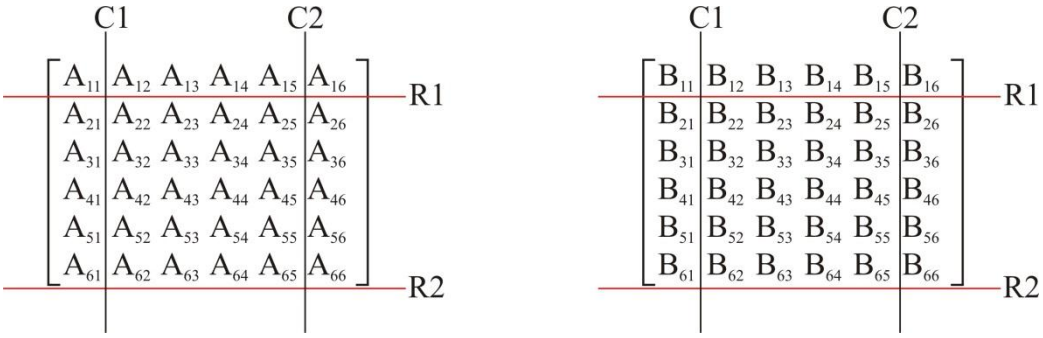


Fig. 3.8 Selection of sub matrices

The complementary mask matrices M1 and M2 are generated using the following procedure. The rows and columns outside the sub matrix are filled with '1's and the rows and columns inside the sub matrix are randomly filled with 1s and 0s as shown in Fig. 3.9. The mask matrices generated randomly for the parents A and B are shown in Fig. 3.9.

1	1	1	1	1	1
1	0	1	1	0	1
1	0	0	1	0	1
1	1	0	0	1	1
1	0	1	0	0	1
1	0	0	1	1	1

M1

0	0	0	0	0	0
0	1	0	0	1	0
0	1	1	0	1	0
0	0	1	1	0	0
0	1	0	1	1	0
0	1	1	0	0	0

M2

Fig. 3.9 Mask matrices M1 and M2

The offsprings can be obtained by multiplying the mask with the corresponding parents and adding suitably as given by equations 3.1 and 3.2.

$$Offspring1 = parent1 \cdot M1 + parent2 \cdot M2 \quad (3.1)$$

$$Offspring2 = parent1 \cdot M2 + parent2 \cdot M1 \quad (3.2)$$

After crossover, the corresponding offsprings obtained are shown in Fig. 3.10.

A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}
A_{31}	A_{32}	A_{33}	A_{34}	A_{35}	A_{36}
A_{41}	A_{42}	A_{43}	A_{44}	A_{45}	A_{46}
A_{51}	A_{52}	A_{53}	A_{54}	A_{55}	A_{56}
A_{61}	A_{62}	A_{63}	A_{64}	A_{65}	A_{66}

B_{11}	B_{12}	B_{13}	B_{14}	B_{15}	B_{16}
B_{21}	B_{22}	B_{23}	B_{24}	B_{25}	B_{26}
B_{31}	B_{32}	B_{33}	B_{34}	B_{35}	B_{36}
B_{41}	B_{42}	B_{43}	B_{44}	B_{45}	B_{46}
B_{51}	B_{52}	B_{53}	B_{54}	B_{55}	B_{56}
B_{61}	B_{62}	B_{63}	B_{64}	B_{65}	B_{66}

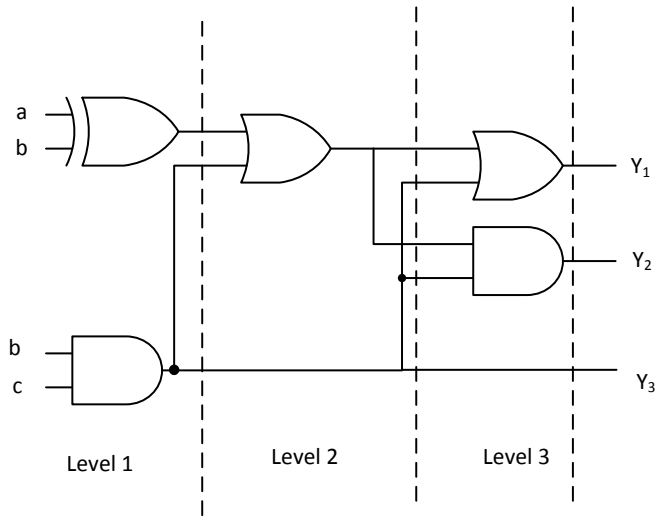
Fig. 3.10 Offsprings of parents A and B

The offsprings of the parents A and B shown in Fig. 3.5 are C_{2D} and D_{2D} respectively and is shown in Fig. 3.11.The corresponding circuits for C_{2D} and D_{2D} are as shown in Fig. 3.12.

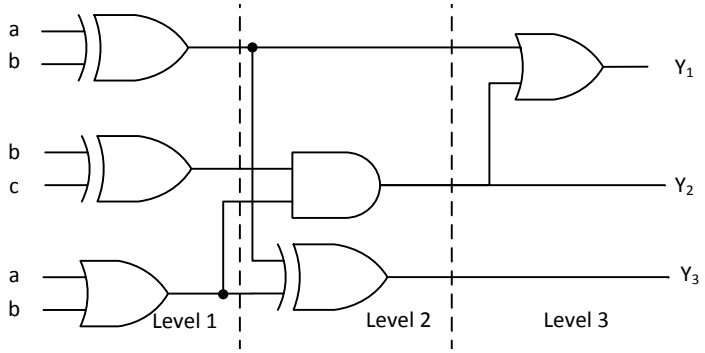
0	0	0	1	0	1
1	0	2	0	2	0
2	1	1	0	0	0
1	1	0	0	2	1
0	0	2	1	2	1
1	1	1	1	0	1

0	0	0	1	1	1
1	0	1	1	0	0
1	0	1	0	1	1
2	0	2	1	1	0
0	1	2	0	2	1
1	0	0	0	1	1

Fig. 3.11 Offsprings (C_{2D} and D_{2D}) after 2D crossover



(a) Offspring C_{2D}



(b) Offspring D_{2D}

Fig. 3.12 Offsprings (Circuits) for parents A and B after 2D crossover

A comparison has been made on the offsprings generated for the same parents using the proposed 2D crossover and the existing linear crossover in the next section.

Comparison between the proposed technique and the existing linear crossover

The linear chromosomal representations for the same individuals A and B are shown in Fig. 3.13.

0100	0210	0010	2101	1201	1201	2110	2001	2011 A
0100	0111	1210	1210	1000	0110	0101	2110	2111 B

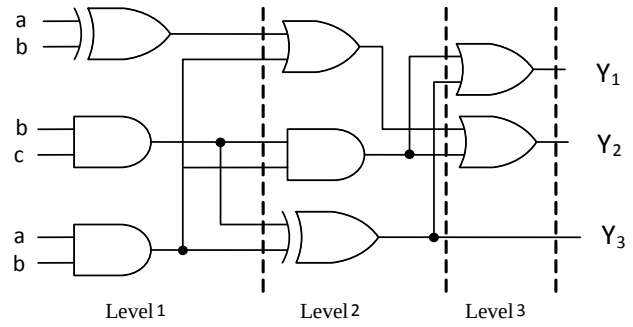
Fig. 3.13 Linear chromosomal representation of parents A and B

If the crossover point is selected randomly as shown in Fig. 3.13, the offsprings C_{linear} and D_{linear} after crossover are shown in Fig. 3.14.

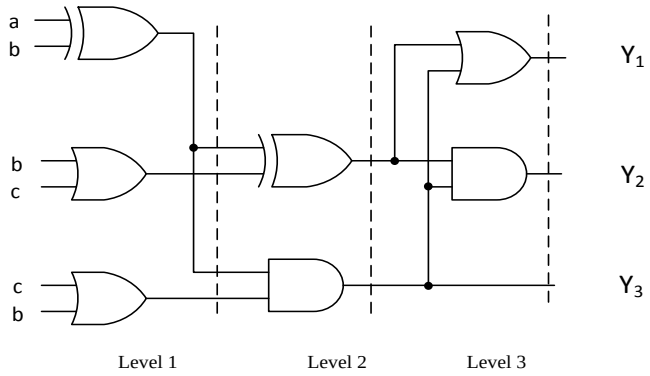
0100	0210	0010	2101	1201	0110	0101	2110	2111 C
0100	0111	1210	1210	1000	1201	2110	2001	2011 D

Fig. 3.14 Offsprings C_{linear} and D_{linear} after linear crossover

The circuits corresponding to the offsprings C_{linear} and D_{linear} are shown in Fig. 3.15.



(a) Offspring C_{linear}



(b) Offspring D_{linear}

Fig. 3.15 Circuits corresponding to offsprings after linear crossover

From Fig. 3.12 and Fig. 3.15, it can be observed that in linear crossover, the changes occur only in the levels after the crossover point, whereas in 2 D crossover, since the sub matrices are swapped, changes can occur in any level either in connections / type of gates. Thus, generated offsprings can have many variations from their parents in 2D crossover, which leads to faster convergence.

After crossover, a small proportion of the offsprings thus evolved is allowed to undergo mutation so as to obtain better circuits. The procedure for mutation is discussed in the next section.

3.3.2 2D Mutation

In mutation, a single bit in the chromosome is selected and it is altered. If it comes in any of the even columns, it is flipped from either 0 to 1 or from 1 to 0 so that gates are changed. For odd columns, that particular digit is set to a random number other than the one which is selected for mutation. This can be achieved by a single mask. To prepare the mask matrix, two random numbers R and C are generated to fix the row and column of the bit to be mutated. A mask matrix M is generated with all elements 0 except the selected bit and this mask operator is superposed over the offspring to be mutated. The selected bit is flipped and the rest are unaltered.

Let the randomly generated numbers R and C be 1, 4. Then a mask matrix (M) is formed with all elements 0 except the element in first row and fourth column, M_{14} which is set to 1. The mask matrix formed is shown in Fig. 3.16 (a). Here, the presence of a ‘1’ in the mask matrix indicates a change in the characteristic of the offspring and a ‘0’ indicates no change. Thus it modifies the gate type / input

combinations, which implies that a completely new circuit can be generated. The offsprings before and after mutation are shown in Figs. 3.16 (b) and (c).

0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

(a) Mutation mask

1	0	0	1	0	1
2	1	2	0	2	0
2	1	1	0	0	0
1	1	2	0	2	1
0	0	2	1	1	1
1	1	1	1	0	1

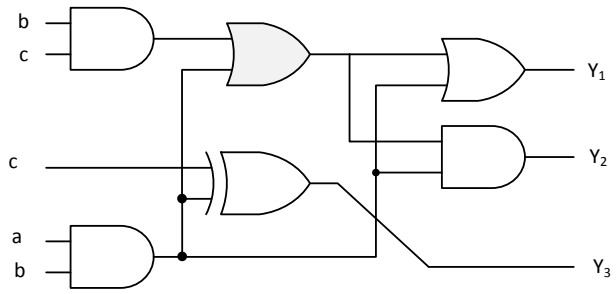
(b) Offspring before mutation

1	0	0	0	0	1
2	1	2	0	2	0
2	1	1	0	0	0
1	1	2	0	2	1
0	0	2	1	1	1
1	1	1	1	0	1

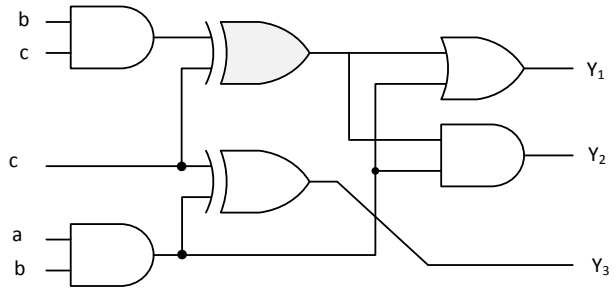
(c) Offspring after mutation

Fig. 3.16 2D Mutation Process

The new offsprings (circuits) before and after mutation are shown in Fig. 3.17. It can be observed that before mutation, the gate in the first row of second level was OR whereas after mutation it has been changed to an XOR gate.



(a) Before mutation



(b) After mutation

Fig. 3.17 Offsprings before and after mutation

After crossover and mutation, the individuals have to be evaluated for fitness. This can be done with the help of formulating a suitable fitness function as discussed in the next section.

3.3.3 Fitness Function

The quality of the evolved circuits depends upon the strength of fitness function used. To ensure optimal / efficient circuits a fitness function with two parts has been proposed. This is done for two reasons. The first part (F_1) is to achieve 100 % functional circuit which matches with the truth table and the second part (F_2) is an additional fitness to minimise the number of gates used. i.e., when more wires appear in the generated circuit, this part of the fitness function increases. If both the inputs are same for AND / OR gates, then those gates are replaced by wires.

Thus, the Fitness function $F = F_1 + F_2$ (3.3)

$$F_1 = \frac{N - \sum_{i=1}^{i=N} (O_{1i} \oplus O_{2i})}{N} \quad (3.4)$$

Where N = Possible number of input combinations (2^n)

O_1 - Evolved output

O_2 - Desired output

$$F_2 = \frac{WIRE}{n^2} \times 100 \quad (3.5)$$

WIRE - Number of wires in the circuit

n - Number of input variables

3.3.4 GA Parameters

The parameters chosen for the GA are a crossover rate of 0.7 and mutation rate 0.3%. A population size of 1000 was used to evolve complex functions also. The number of generations chosen was 200 for 2D technique and 500 for linear technique. Roulette Wheel Selection technique has been used for selecting the individuals for crossover.

3.4 RESULTS

Using this proposed 2D technique, circuits of 2 to 6 inputs with varying complexities have been evolved. Benchmark functions specified in Section 1.7 are used to validate the results. In addition to this, circuits have been evolved for some functions in the literature for comparison.

On various runs, the number of generations needed for convergence varies. Hence the number of generations required is considered to be the average of all the runs. Computational time considered is the average of the convergence time required in all the runs.

Benchmark Functions

1. Majority3

This is a benchmark function of 3 variables which produces an output one when the number of ‘1’s in the function is two or more.

The evolved circuit by the proposed method is shown in Fig. 3.18. The circuit needs only 4 gates in 3 levels. Whereas with conventional design based on K map, 5 units embedded in 3 levels are needed. The average number of generations required is only 32 compared to 59 with linear (existing) chromosomal representation and genetic operations.

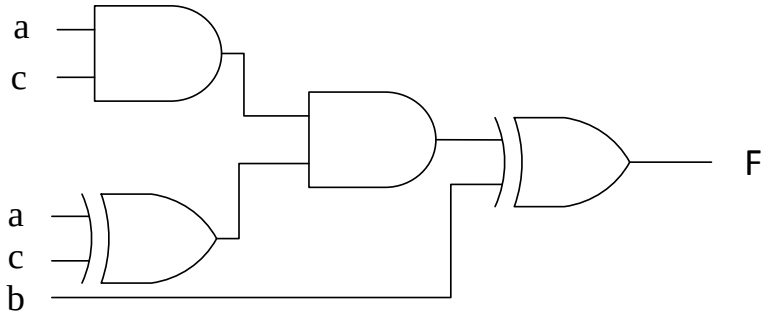


Fig. 3.18 Circuit evolved for Majority3 using gates

2. Four Bit odd parity checker

This is a four input function which produces a high output for odd number of '1's in the input combinations. The circuit evolved is shown in Fig. 3.19 with 3 gates in 2 levels, where as conventional technique needs 22 units in 6 levels.

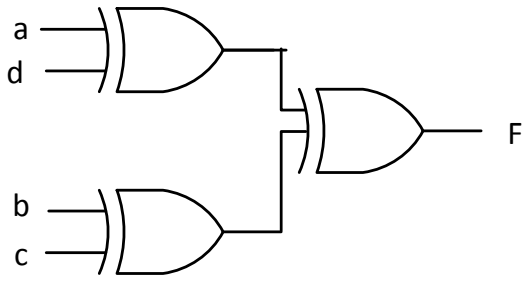


Fig. 3.19 Four bit odd parity checker using gates

3. **xor5 –** A 5 bit XOR function

With the use of XOR gates in automated design, the function can be realised easily with lesser number of gates. The evolved circuit shown in Fig. 3.20 needs only 4 gates in 3 levels. The circuit was evolved in 75 generations. Its design using conventional techniques is very hard.

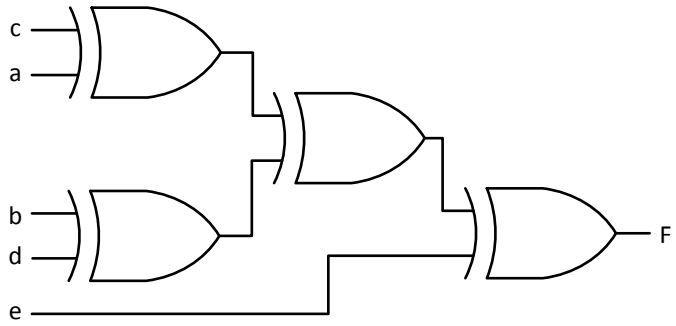


Fig. 3.20 Circuit generated for xor5 using gates

4. **6one135 –** A 6 bit benchmark function which produces an output 1 when the number of 1's in the input combinations are 1, 3 or 5.

Though design using K map is possible for functions up to 6 variables, design of more than 4 variable functions is not easy. On mapping this function into K map, there are no two adjacent ones to group together. Hence it is not possible to minimise the function by conventional techniques. Reduction by applying algebraic rules is very complex. By the proposed technique, the circuit was evolved in lesser time with minimum number of gates.

The circuit generated by evolution is shown in Fig. 3.21. The average number of generations needed is 85 compared to 150 with the existing technique.

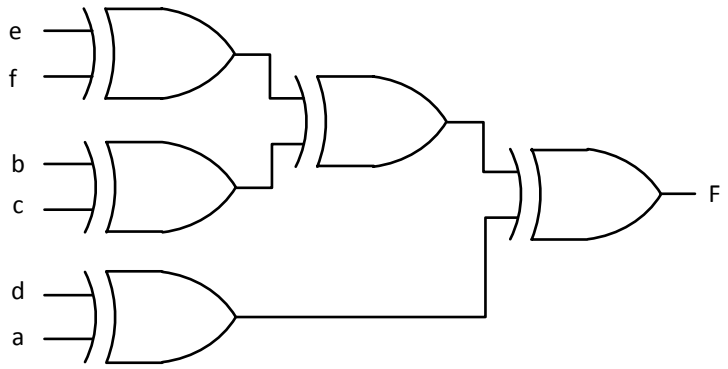


Fig. 3.21 Circuit evolved for 6one135 using gates

5. **6one0246** A 6 bit benchmark function which produces an output '1' when the number of 1s in the input combinations is 0, 2, 4 or 6.

The evolved circuit is shown in Fig. 3.22 with 6 gates in 4 levels in 90 generations and the computational time is 122.6345 sec, whereas with linear representation, the time was 239.362sec.

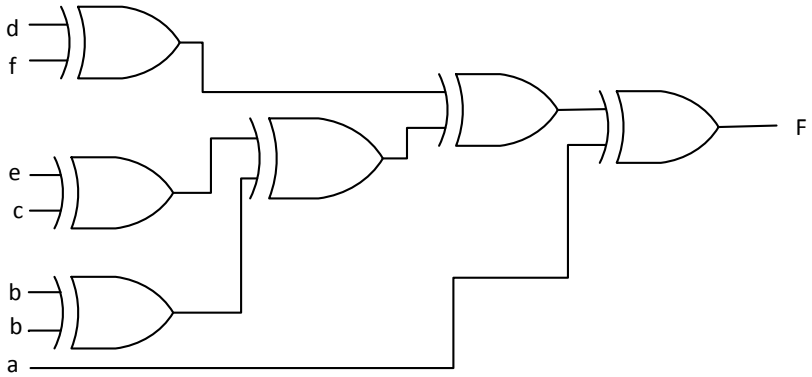


Fig. 3.22 Circuit evolved for 6one0246 using gates

Other Functions

1. Four bit even parity checker
2. Four bit binary to gray code converter
3. Full adder

4. 2-1 mux
5. F (a, b, c) = Σ m (3, 5, 6)
6. F (a, b, c, d) = Σ m (4, 5, 6, 7, 8, 9, 10, 13)
7. F (a, b, c, d) = Σ m (1, 2, 4, 5, 7, 8, 10, 11, 13, 14)

1. **Four Bit even parity checker -** The circuit evolved by the proposed method is shown in Fig. 3.23. It can be observed that the circuit requires only 4 gates compared to 27 gates in manual design. The number of generations required for convergence is 72 compared to 130 in linear representation.

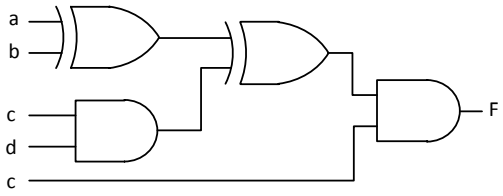


Fig. 3.23 Realisation of Four bit even parity checker using gates

2. **Full Adder -** It is a 3 input 2 output function

Fig. 3.24 shows the evolved circuit for a full adder circuit which requires only five gates to realise the sum and carry outputs. It has been observed that the convergence speed of the full adder circuit using 2D technique is much more than the existing technique. The circuit is evolved within 20 generations using the proposed technique compared to 110 generations in the existing linear technique.

It has been mentioned in Section 1.2.1, that conventional techniques like Quine - McCluskey method and K map do not support the use of XOR / XNOR gates. On using XOR gates in automated design, the sum could be obtained in two levels with 2 gates and

carry in three levels with 3 gates. The circuit for Full adder using 2 input gates is shown in Fig. 3.25 which uses 18 gates in 5 levels.

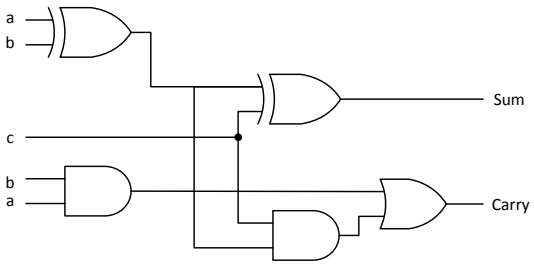


Fig. 3.24 Circuit evolved for Full Adder using automated technique

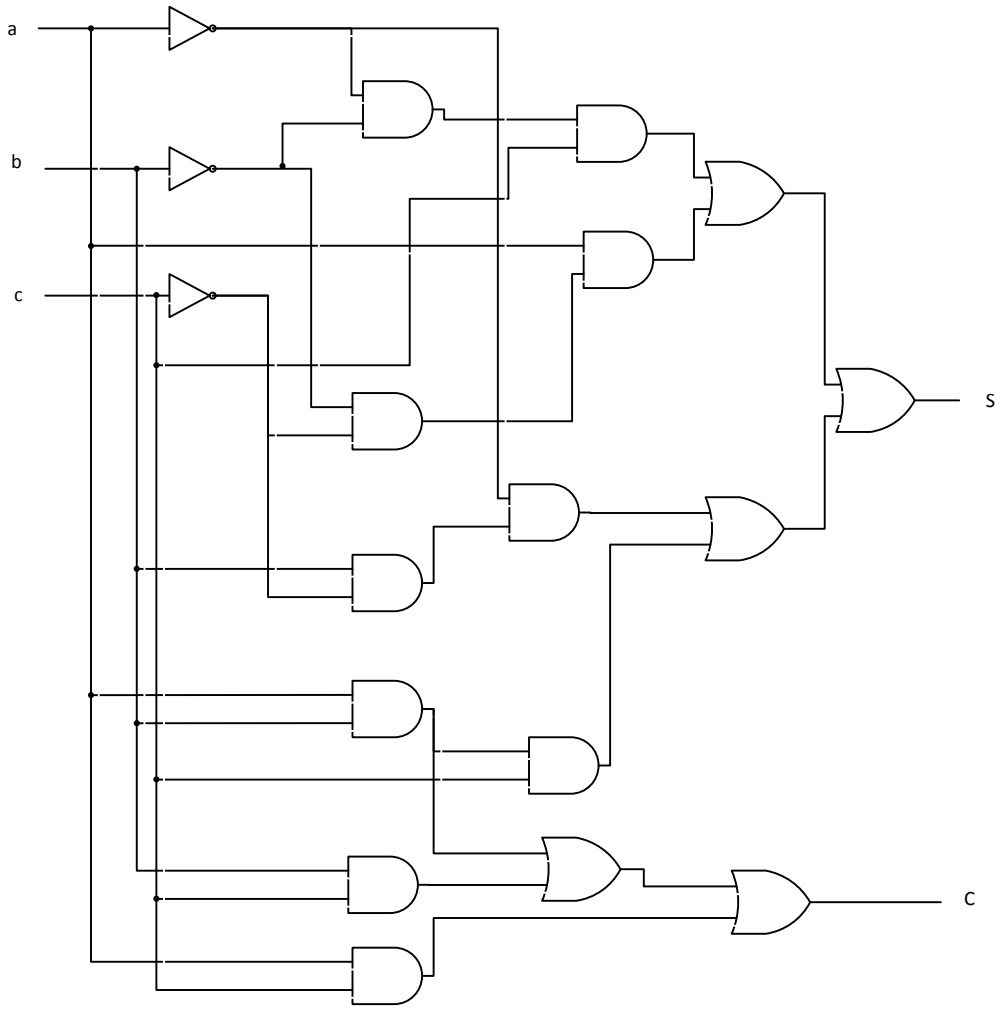


Fig. 3.25 Circuit for Full adder using basic gates

3. **2-1 multiplexer** - The evolved circuit of a 2-1 Multiplexer is shown in Fig. 3.26(a). Conventional technique which employs only basic gates needs 4 gates in 3 levels as shown in Fig. 3.26(b). It is obvious that the circuit evolved by automated design is more expensive in terms of area and power by the use of XOR gates. In this thesis, the gates used are XOR, AND, OR and WIRE. This example has been cited here only to demonstrate the faster convergence of 2D crossover and mutation compared to existing linear crossover technique. The proposed technique requires only 40 generations while the existing technique requires 320 generations to evolve the desired functionality. Thus the computation time has been reduced significantly.

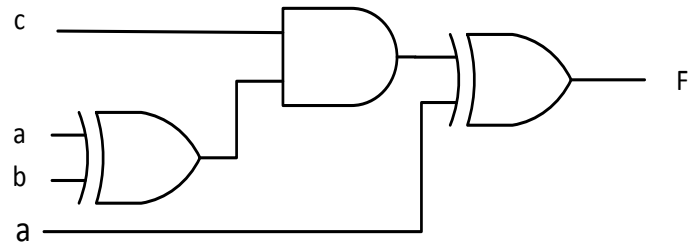


Fig. 3.26 (a) Circuit generated for 2-1 multiplexer by automated design

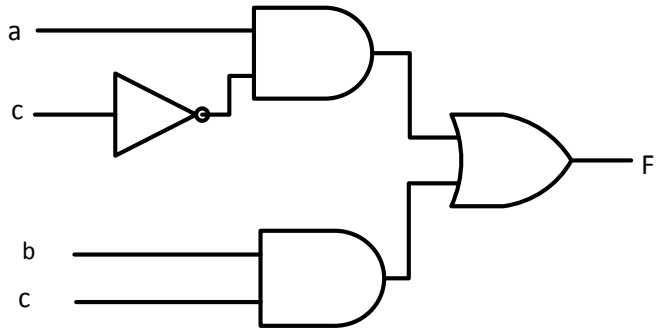


Fig. 3.26 (b) Circuit for 2-1 multiplexer by conventional design

4. **Four bit Binary to Gray code converter** - a four input four output function which converts the binary code into its equivalent gray code. The circuit generated for the function is shown in Fig. 3.27 which uses only 3 gates.

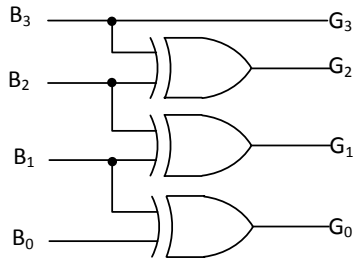


Fig. 3.27 Evolved circuit for a Four bit Binary to Gray code converter

5. **F (a, b, c, d) = Σ m (4, 5, 6, 7, 8, 9, 10, 13)** - An example taken from (Slowik and Bialko, 2008). The authors used linear chromosomal representation and the circuit was realised in 4 gates. The generated circuit with the proposed technique is at par with the literature using 4 gates as shown in Fig. 3.28.

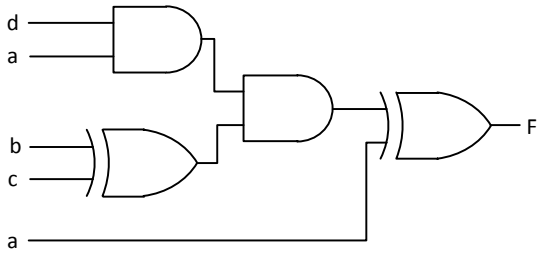


Fig. 3.28 Circuit evolved for F (a, b, c, d) = Σ m (4, 5, 6, 7, 8, 9, 10, 13)

6. **F (a b, c) = Σ m (3, 5, 6)** - a function which produces a high output when the number of '1's in the input combinations is two.

   The circuit generated for this function is shown in Fig. 3. 29. It requires only 4 gates, whereas conventional technique needs 11 gates for realising the function.

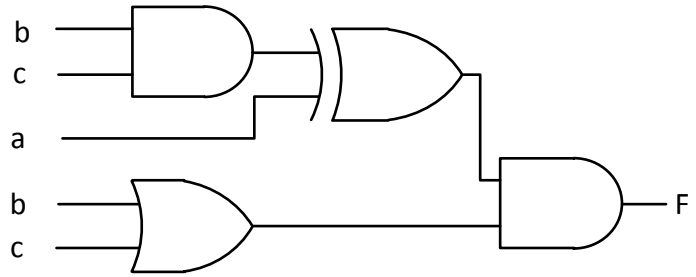


Fig. 3.29 Evolved circuit for F (a, b, c) = Σ m (3, 5, 6)

7. **F (a, b, c, d) = Σ m (1, 2, 4, 5, 7, 8, 10, 11, 13, 14)** - This is an example taken from (Coello, 2004) in which the automation was done using PSO and the circuit was evolved with 6 gates and no mention was made regarding the convergence time. With the proposed 2D technique too, it was realised with 6 gates as shown in Fig. 3.30.

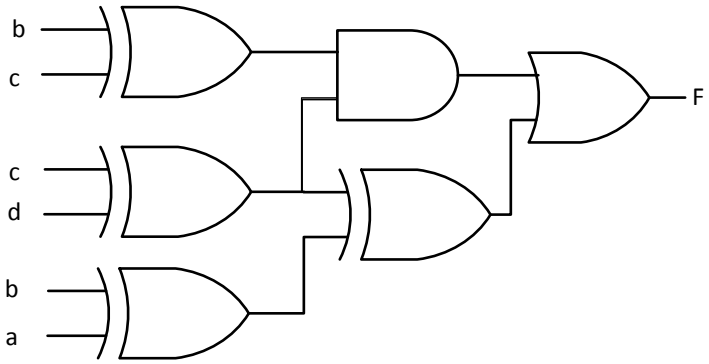


Fig. 3.30 Circuit of F (a, b, c, d) = Σ m (1, 2, 4, 5, 7, 8, 10, 11, 13, 14)

Table 3.3 shows a comparison of automated design with conventional design in terms of number of gates / levels used. Only a few of the evolved circuits have been considered for tabulation. It can be observed that there is a considerable saving in the number of gates and number of levels. Reduction in number of levels is a clear indication of reduced delay.

Table 3.3 Comparison between conventional and automated techniques in terms of number of gates / levels used

Sl No.	Function	Conventional (gates/levels)	Automated (gates/levels)
1	Half adder	6/3	2/1
2	Σ m (3, 5, 6)	11/5	4/3
3	Σ m (1, 2, 4)	11/5	4/3
4	Majority3	5/3	4/3
5	Full adder	18/5	5/3
6	2-1 multiplexer	4/3	3/3
7	3 bit binary to gray code convertor	9/3	2/1
8	3 bit odd parity checker	14/5	2/2
9	4bit odd parity checker	27/6	3/3
10	4 bit even parity	27/6	4/3
11	4 bit binary to gray code convertor	13/3	3/1
12	Σ m(4, 5, 6, 7, 8, 9, 10, 13)	10/5	4/3
13	Σ m (1, 2, 4, 5, 7, 8, 10, 11, 13, 14)	19/5	6/3
14	Σ m (0, 2, 3, 8, 9, 11, 12, 13, 14)	14/5	5/3
15	xor5	60/7	4/3
16	6one135	83/8	5/3
17	6one0246	83/8	6/4

From Table 3.3, it can be observed that the number of gates / levels is reduced significantly in automated design compared to the conventional design. Moreover, complex functions like xor5, 6one135, 6one0246 cannot be simplified by conventional techniques.

Table 3.4 gives a comparison of the convergence time required for realising the benchmark functions using the proposed 2D technique and linear technique. The program was run on INTEL core i5 processor @ 2.5 GHz, 32 bit processor with 2 GB RAM. It can be observed that the time needed to arrive at 100% functional and

optimal solution is reduced significantly with the proposed technique compared to linear crossover and mutation techniques.

Table 3.4 Comparison of the proposed technique with the existing technique in terms of convergence time

Sl No.	Function	Convergence time in sec (2D)	Convergence time in sec (Linear)
1	Majority3	5.3989	33.12621
2	4bit odd parity checker	68.3177	154.7651
3	xor5	122.230098	190.24014
4	6one135	116.89353	201.49251
5	6one0246	122.6345	239.36210

From Table 3.4, it is obvious that the convergence time required to realise the functions with the proposed technique is reduced considerably.

Fig. 3.31 shows the comparison between the two approaches in terms of the number of generations needed for the circuits to converge. It is evident that the proposed 2D technique evolves circuits in lesser number of generations compared to the existing linear technique.

The functions used for the analysis are:

- F1: 2 bit circuit (Half Adder)
- F2: majority3
- F3: 4 bit Circuit (4 bit binary to gray code converter)
- F4: xor5
- F5: 6one135
- F6: 6one0246

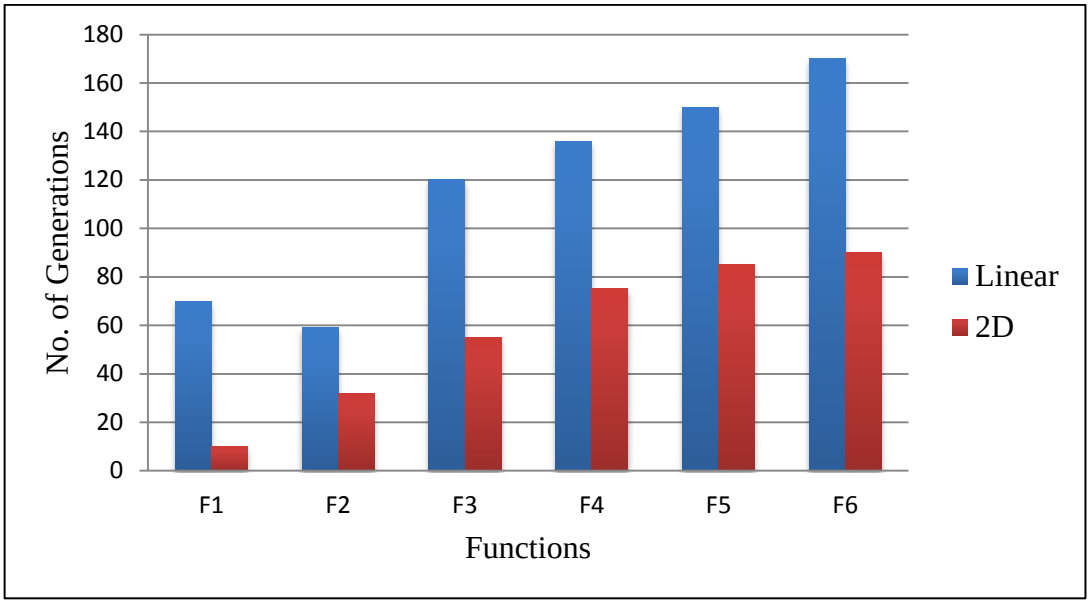


Fig. 3.31 Comparison of the proposed technique with the existing technique in terms of number of generations

From the above examples and discussions it can be concluded that the proposed technique is faster compared to the existing technique. The optimum circuits were always generated in less than 100 generations with the proposed technique.

3.5 SUMMARY

A new 2D representation for the design of CLCs is proposed. Suitable crossover and mutation techniques are developed for this 2D representation. The fittest circuit evolved is in the form of '1's and '0's in a matrix form so that decoding is made easy. Various circuits including benchmark functions with inputs up to 6 variables have been evolved. A comparison of the convergence time between the proposed technique and the conventional method has been made. In linear crossover, some of the levels may remain unaltered during crossover operation, whereas with the proposed 2D technique, since the sub matrices are swapped for crossover operation, variations from the parent circuits can occur at any level either in connections / type of gates. Thus, the convergence speed of GA has been significantly increased compared to the conventional method, which in turn reduces the computational time for evolving optimal circuits.

CHAPTER 4

COMBINATIONAL LOGIC CIRCUIT DESIGN USING UNIVERSAL LOGIC MODULES

4.1 INTRODUCTION

Over the last few decades, several researches were carried out to automate the design of digital circuits using evolutionary algorithms. The main motivation behind this comes from the need to produce power efficient and cost effective optimal circuits. Since the use of integrated circuits in the areas of high performance computing, telecommunication, consumer electronics etc. are growing at a faster pace, a cost effective design is very significant. Evolutionary design helps to obtain an optimised circuit in terms of area / delay / power.

One of the advantages of evolutionary design is the use of universal building blocks such as multiplexers / RM blocks as the basic design element. The repeated use of a single design element reduces the cost of VLSI implementation and hence the exclusive use of ULMs is recommended for the design. Earlier, the researchers focused on generating circuits using minimum number of gates. However, in VLSI design, reduction in manufacturing cost is more significant compared to the reduction in number of components used (Aguirre *et al*. 1999 and 2000). This is possible by replicating the same element so as to evolve fully functional circuits.

In this chapter, design of CLCs based on the exclusive use of 2-1 mux / 2-1 RM ULM is discussed. The objective is to generate fully functional circuits with minimum hardware using suitable evolutionary algorithms. The quality of the evolved circuits

solely depends on the algorithm used for evolution. Being an efficient evolutionary algorithm, GA is used as the optimisation tool to evolve the circuits.

4.2 METHODOLOGY

Any Boolean expression can be implemented using ULMs such as multiplexers or RM ULMs. In this work, 2-1 mux / 2-1 RM ULM has been used for the realisation of Boolean functions. By SI technique, a function of *n* variables can be realised by a tree network using 2^n-1 binary ULMs in *n* levels. The existing reduction techniques such as Shannon's / Davio decomposition technique do not help in arriving at an optimised circuit. Sub functions are useful for implementing a Boolean function. To implement a hardware circuit for a function of *n* variables, the following identities based on Shannon's decomposition and Davio decomposition techniques can be used.

$$F = F'(A_j)' + F''A_j \qquad \text{For 2-1 mux implementation} \tag{4.1}$$

$$F = F'\left(A_j\right)' \oplus F'''Aj \qquad \text{For 2-1 RM ULM implementation} \tag{4.2}$$

where F′, F″ and F‴ are functions of n-1 variables and $(A_j)'$ is the complement of A_j.

Both identities are used to reduce the original design problem into two simpler problems (Wang, 2000). Applying such decomposition repeatedly with another variable A_j allows the problem under synthesis to be reduced further and further to either literals or constants as mentioned in Section 1.4.2. Fig. 1.5 in Section 1.4.2 shows the circuit realisation of a three variable function using binary multiplexers in SI.

The characteristic features of the tree network of ULMs in SI are:

- All the units in the same level share the same control signal
- A control signal selected in one level cannot be used in other levels
- All the variables of the function should be used as control (select) signal
- Only the variables of the function can be used as control signal
- Inputs to the first level (bottom most) should be 0s or 1s
- Regular / uniform interconnections

Since the Shannon's / Davio decomposition is not helpful in reduction of hardware, an attempt has been made to evolve better circuits by making some alterations in the conventional tree network. GA is used for this purpose. Design of CLCs using binary multiplexers / RM ULMs is discussed in the subsequent sections.

4.2.1 Binary Multiplexer (2-1 mux)

The conventional use of multiplexer is to route data from one of the *n* sources to a common destination. Besides, it can be used for the realisation of combinational circuits. It is known that, any function can be realised by applying Shannon's decomposition technique that uses only binary multiplexers. The decomposition technique has been explained in Section 1.4.2.

A multiplexer with *n* selection lines is a combinational circuit that selects data from 2^n input lines and directs it to a single output line. A binary multiplexer has 2 input lines and one control line. They are of "active low" or "active high" denoted as class A or class B multiplexers as mentioned in Section 1.4.2. Only class A mux is used in this work and its logic symbol is shown in Fig. 4.1.

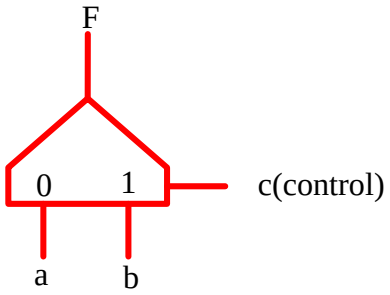


Fig. 4.1 Logic symbol of a Class A mux

The output of class A mux is given by

$$F = [(c'.a) + (c.b)] \tag{4.3}$$

where c′ is the complement of c.

4.2.2 Binary RM ULM

RM ULM is based on AND-XOR logic while mux is based on AND-OR logic. XOR gates are slow and require a large area when realised in comparison with OR gates. But with the advancement of new technologies and advent of various FPGA devices, XOR / XNOR implementation of circuits has become easy.

The logic symbol of a 2-1 RM ULM is shown in Fig. 4. 2 whose output is given by,

$$F = a \oplus b \cdot c \tag{4.4}$$

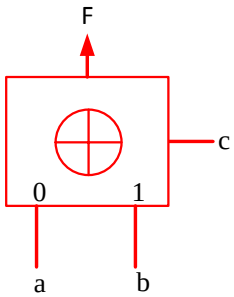


Fig. 4.2 Logic symbol of a 2-1 RM ULM

Any Boolean function can be expressed in the form of Reed-Muller (RM) expression using AND and XOR operators. This representation has various advantages such as ease of complementing and testing. It requires lesser number of gates and interconnections in applications which involve XOR operations.

Basic theorems involved in the design using RM ULM are

$$X \oplus X = 0; \tag{4.5}$$

$$X \oplus 0 = X; \tag{4.6}$$

$$X \oplus X' = 1; \tag{4.7}$$

$$X \oplus 1 = X'; \tag{4.8}$$

where X is any variable and X' is its complement.

This thesis aims at the design of circuits with minimum number of units and levels compared to SI which uses 2^n-1 units in n levels. E.g., a three variable function needs seven units distributed in three levels with conventional method. An attempt has been made to reduce the number of units / levels by making certain modifications on the SI technique using GA.

4.3 PROPOSED METHODS

In this work, few modifications have been made on the well known Shannon's / Davio decomposition technique, so as to evolve better circuits with minimum hardware. The most important property of conventional design is that the units in the same level share the same control / select signal as discussed in the previous section. The proposed method allows any variable to be used as select signal to any unit in

any level. In addition to the use of variables as select signals, this method uses functions derived from previous level as select signal which is not permitted in SI. With this approach, circuits can be evolved with lesser number of modules and levels than standard implementation, thereby reducing the cost and complexity of the circuit.

Two different GA based techniques have been proposed to obtain optimal circuits with binary ULMs. These techniques are referred to as ***Constant Input Method*** **(CIM)** and ***Variable Input Method*** **(VIM)** in this thesis and are discussed in the subsequent sections.

4.3.1 Constant Input Method (CIM)

As in Shannon's / Davio decomposition techniques, the inputs to the ULM units in the first level (where inputs are fed) are only constants, i.e., 0s and 1s. Inputs to the subsequent levels can include 0, 1, or outputs of immediate preceding level. In the case of control signals, Shannon's / Davio decomposition technique used fixed control signals for each level whereas the proposed method uses randomly generated control signals and the circuits are evolved using GA.

In CIM, the modifications made to the existing Shannon's / Davio decomposition technique are consolidated as follows.

1. The control signals for all the units in a particular level need not be the same as they are selected at random using GA.
2. The signal selected at one level can be used for other levels too.

3. The control signal need not be a variable, it can also be a function derived from the previous level.
4. All the variables need not be used as control signal.

Thus, the control signals for the ULMs can be variables of the Boolean function, their complements, or the outputs of units from the previous level. Using the above modifications, circuits can be evolved with lesser number of units / levels.

The flow chart for the proposed method is shown in Fig. 4.3. The maximum number of levels required to realise a function of *n* variables is *n* and the number of units required is 2^n-1. The number of units in a level is 2^{n-l}, where *l* is counted from the level where data inputs are fed to the circuit. Incorporating the modifications suggested above, a random population is generated. The circuits are encoded into chromosomes which can accommodate the worst possibility of having the number of levels and number of units as in SI. Output of the n^{th} level unit is the final output of the circuit. It is evaluated for 100% fitness with minimum number of levels / units. The fitter individuals are allowed to undergo selection, crossover and mutation and the process is continued till optimal circuits are evolved.

Once the circuits are evolved, the ULMs are checked for redundancies and are eliminated. For E.g., if multiple units have the same data inputs and the control signals in the same level, then one of them will be retained and the rest will be discarded. Moreover, muxes with the same data inputs can be replaced by a wire. Hence the numbers of units are further reduced thereby reducing the power and area. The advantage of CIM is that the inputs to the tree are only constants.

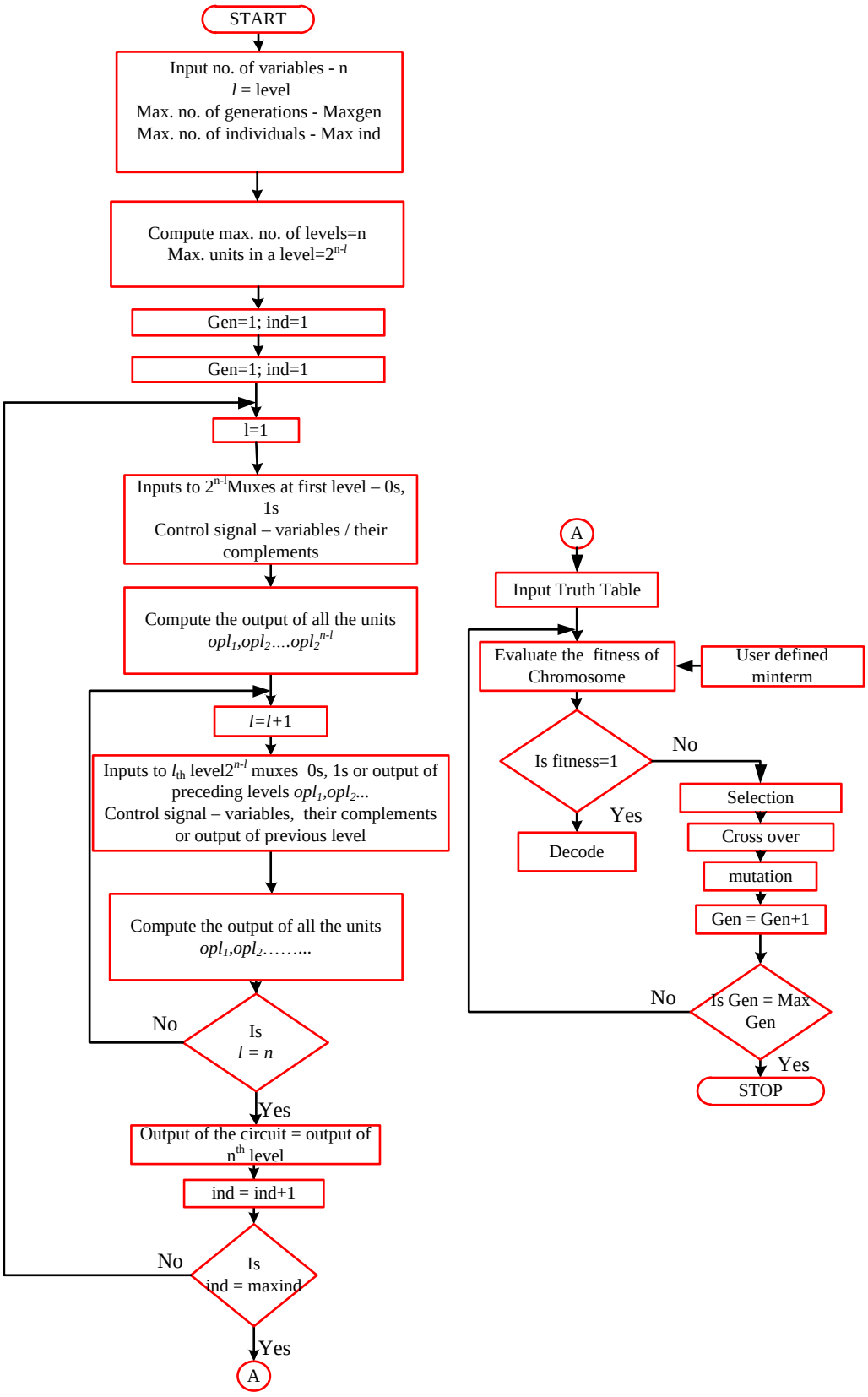


Fig. 4.3 Flow chart for CIM

4.3.2 Variable Input Method (VIM)

VIM is similar to the CIM except that the inputs to the units in first level can be 0s, 1s, variables or their complements. With this added modification, the number of units / levels can be still reduced. GA is used to select both the inputs and the control signals for optimisation.

4.4 IMPLEMENTATION OF GA

In the case of design using gates, it was assumed that an *n* variable circuit had *n* levels and each level had *n* gates so that the chromosomal representation for the circuit was in the form of a square matrix and 2D crossover and mutation were applied over these square matrices. While in the case of a tree structure, the number of units in all the levels is not equal. The number of units in the i^{th} level of a tree is 2^{n-i} where the level is counted from bottom to top. E.g., for a three bit function, the number of units in the three levels is 4, 2 and 1 respectively. Due to the non-uniformity in the number of units in each level of the tree, 2D crossover technique cannot be used here. Hence linear crossover and mutation are adopted in this work.

4.4.1 Chromosomal Representation for the Circuit

The chromosomal representation of circuits using mux and RM ULM are similar. Hence the encoding of the tree network using RM ULM units in both the methods are discussed. The chromosome contains the information about the inputs and control signals for the ULM units.

Constant Input Method

The encoding of the units are similar except for the first level. Encoding of a single RM ULM in the first level is as follows. A three input function is considered as an example. As mentioned earlier, in CIM, the inputs to the first level are 0, or 1 and can be any one of the combinations - 0, 0; 0, 1; 1, 0; or 1, 1. To represent this, two bits are needed. The control signal can be variables or their complements. For a three input function, 6 possibilities are there and hence 3 bits are required to represent the control signal. Fig. 4.4 shows the typical representation of a chromosome for an RM unit in the first level (level where the inputs are fed) of the tree for a three variable circuit. The same is repeated for all the units in first level.

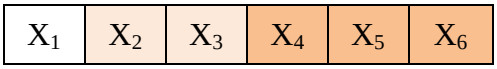


Fig. 4.4 Chromosomal representation of an RM ULM in the first level

Here, X_1 indicates the presence / absence of the unit, X_2X_3 represents the inputs to the unit, and the combination $X_4X_5X_6$ corresponds to the control signal used.

Table 4.1 shows the encoding of the above unit. Tables 4.1 (a) and (b) shows the encoding of the inputs and the control signal of that unit respectively.

Table 4.1 Encoding of a single RM ULM in the first level of a tree implemented in CIM

(a) Encoding of inputs

X		Inputs
X_2X_3	00	0, 0
	01	0, 1
	10	1, 0
	11	1, 1

(b) Encoding of control signals

X		Control signal
$X_4X_5X_6$	000	a
	001	b
	010	c
	011	a′ *
	100	b′
	101	c′
	110	a
	111	b

* ′ represents the complement

For E.g., let the randomly generated chromosome be "101010". Since the first bit (X_1) is 1, it implies that an RM ULM exists in that position. The next two bits "01" indicate that the inputs are 0, 1 and the last three bits "010" imply that the control signal is "c". The generated unit corresponding to this chromosome is shown in Fig. 4.5 and its output is *c*.

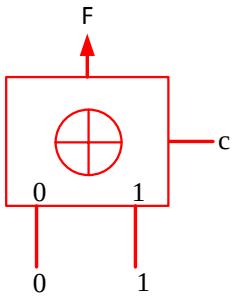


Fig. 4.5 Unit evolved for the chromosome "101010"

For the second level, in addition to 0s and 1s, the inputs can be outputs from the immediate preceding layer. Control signals can be variables, their complements or outputs from the preceding level. Hence encoding is to be made incorporating all

these possible combinations. Fig. 4.6 shows the chromosomal representation for a unit in second level.

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}

Fig. 4.6 Chromosomal representation of an RM ULM in the second level

As mentioned earlier, bit X_1 is to indicate the presence of a unit, X_2 to X_7 represents the possible inputs and the last four bits X_8 to X_{11} are used to represent the possible control signals. Similarly the other units in the tree can be encoded.

Variable Input Method

Here, in addition to 0, 1, the inputs can be variables or their complements and hence the number of bits required for encoding is more compared to CIM. Fig. 4.7 shows the chromosomal representation of an RM ULM in first level. Bit X_1 indicates the presence of a unit as in CIM. Combination of bits X_2 to X_7 represents the possible input signals to the RM ULM at the first level which can be any of the possible combinations of input variables a, b, c, their complements, 1s or 0s. $X_8X_9X_{10}$ determines the control signal of the unit.

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}

Fig. 4.7 Chromosomal representation of an RM ULM in the first level of VIM

For E.g., consider a string “1011100010”. Here the first bit X_1 implies that an RM ULM exists at that position of the tree. The string “011100” corresponds to the input combination a' and b. The possible control signals in the first level are variables and their complements. In the above string, last 3 bits “010” implies that the control signal

is c. The unit corresponding to this string is shown in Fig. 4.8 whose output is given by ***a' ⊕ (b. c).***

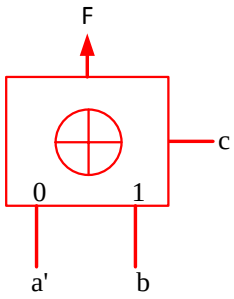


Fig. 4.8 Unit evolved for the chromosome "1011100010"

For the second level, in addition to the variables and their complements, 1s and 0s, the outputs of previous level can be given as inputs. As mentioned above, control signal can be any of the variables, their complements, or functions derived from the previous level. So encoding is to be done incorporating all the possible combinations of inputs and control signals. Coding is being done for all other units in a similar manner to make the complete chromosome for the entire circuit.

Chromosomes are encoded in a similar fashion for circuits using multiplexers too.

4.4.2 Optimisation Using GA

Individuals (Chromosomes) are generated at random which holds the particulars regarding the presence of a unit, corresponding inputs, control signals and outputs as genes. To start the GA, an initial population of circuits (strings) is generated at random. Search is then carried out among this population and the genetic operations such as reproduction, crossover and mutation are carried out as explained in chapter 1. The total number of correct outputs in response to the corresponding inputs gives the measure of fitness. Successive generations of new strings are reproduced on the basis of their fitness.

In this work, RWS technique is used to select the strings for crossover. Single point crossover is performed. The mutation operation occurs rarely, usually 0.2% to 0.3% of the population undergo mutation which changes the characteristics of a given gene in the chromosome. The number of individuals in the initial population is fixed and is maintained across the generations until the GA converges. A suitable fitness function is formulated to help in the convergence of GA.

4.4.3 Fitness Function

The fitness function (F) used for evaluating the CLC evolved is assumed to consist of three parts F_1, F_2 and F_3 and is given by

$$F = F_1 + F_2 + F_3 \tag{4.7}$$

where F_1 ensures 100% functionality of the circuit, F_2 reduces the number of units and F_3 ensures minimum number of levels.

The proximity of the outputs of the evolved circuit to the desired outputs in the truth table is evaluated for each individual in the population and the fitness is calculated as

$$F_1 = \left(\frac{N - \sum_{i=1}^{N} (O_{1i} \oplus O_2 i)}{N}\right) \times 100 \tag{4.8}$$

where O_{1i} is the evolved output corresponding to the i^{th} row, O_{2i} is the corresponding desired output and N is the number of possible combinations of inputs. ($N = 2^n$)

$$F_2 = \frac{1}{count} \times 100 \tag{4.9}$$

where count is the no. of units in the generated circuit.

$$F_3 = \frac{1}{level} \times 100 \tag{4.10}$$

where level is the number of levels of the circuit.

Based on F_1, a fit circuit is evolved. Once a 100% fit circuit appears, extra fitness inversely proportional to the number of blocks and levels in the corresponding circuit referred to as F_2 and F_3 are added to the actual fitness. As mentioned in Section 4.3.1, at the end of the prescribed number of generations, idle units if any are eliminated from the best fit circuit, ensuring an optimal solution for the circuit.

4.5 GA PARAMETERS

The parameters selected for GA are single point crossover with a crossover rate of 0.7 and mutation rate of 0.3 %. Population size of 20,000 is chosen so that even complex circuits can be evolved. For most of the functions, the number of generations was chosen as 100, and for complex functions, it was chosen as 500. The simulation was done in MATLAB R2012a.

4.6 RESULTS

The results on the design of CLCs are grouped into two categories i) using mux and ii) using RM ULM. In this work, design using ULMs have been limited to circuits with one output only. A comparison has been made between the SI technique and the proposed methods, CIM and VIM. The results are validated using benchmark

functions listed in Table 1.1 of chapter 1. Besides, circuits have been generated for several other functions available in the literature using the proposed techniques.

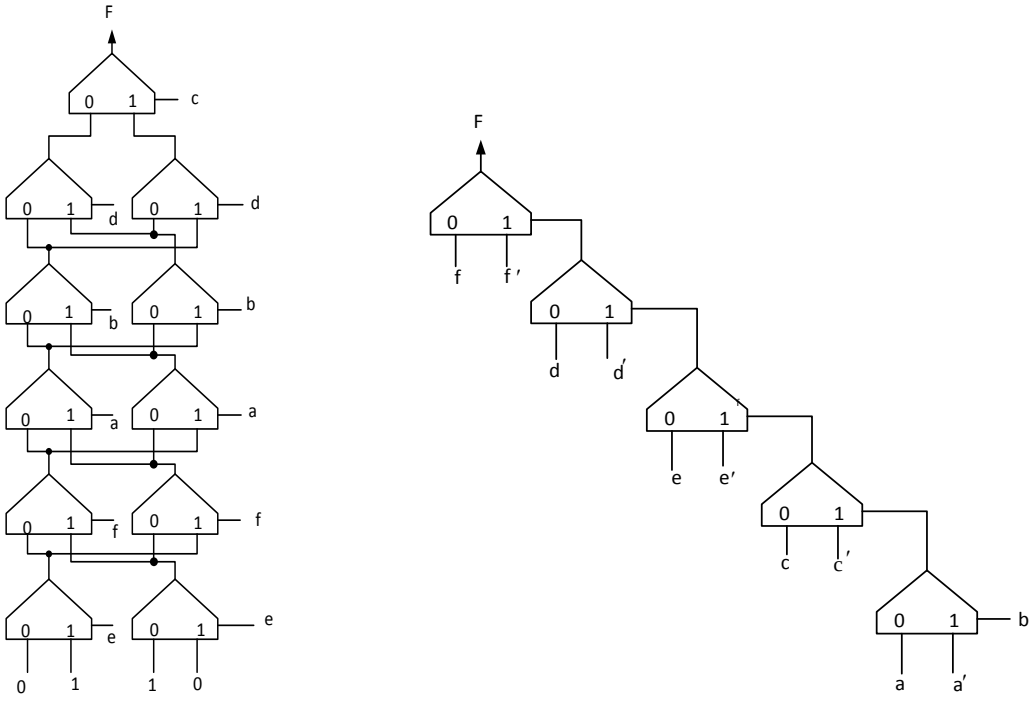
4.6.1 Realisation of Circuits Using mux

1. 6one135

The circuits evolved with the proposed methods CIM and VIM are shown in Figs. 4.9 (a) and 4.9 (b) respectively. From the figures, it can be seen that the circuit needs 11 units in 6 levels with CIM and only 5 units in 5 levels with VIM compared to 63 units in 6 levels with SI technique. Thus there is a saving of 82.53% and 92.06% respectively in the number of units compared to SI. The number of levels is also reduced in VIM so that the delay can be reduced.

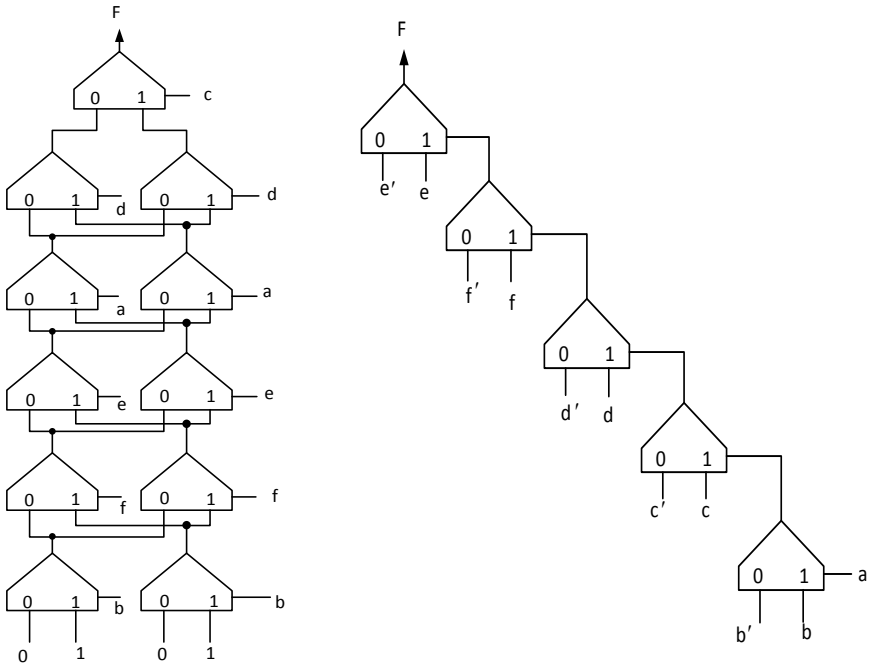
2. 6one0246

The circuits evolved by the proposed methods CIM and VIM need 11 units / 6 levels and 5 units / 5 levels respectively as shown in Figs. 4.10 (a) and (b) respectively, whereas the same circuit needs 63 units / 6 levels in SI.



(a) CIM (b) VIM

Fig. 4.9 Circuit generated for 6one135 using mux

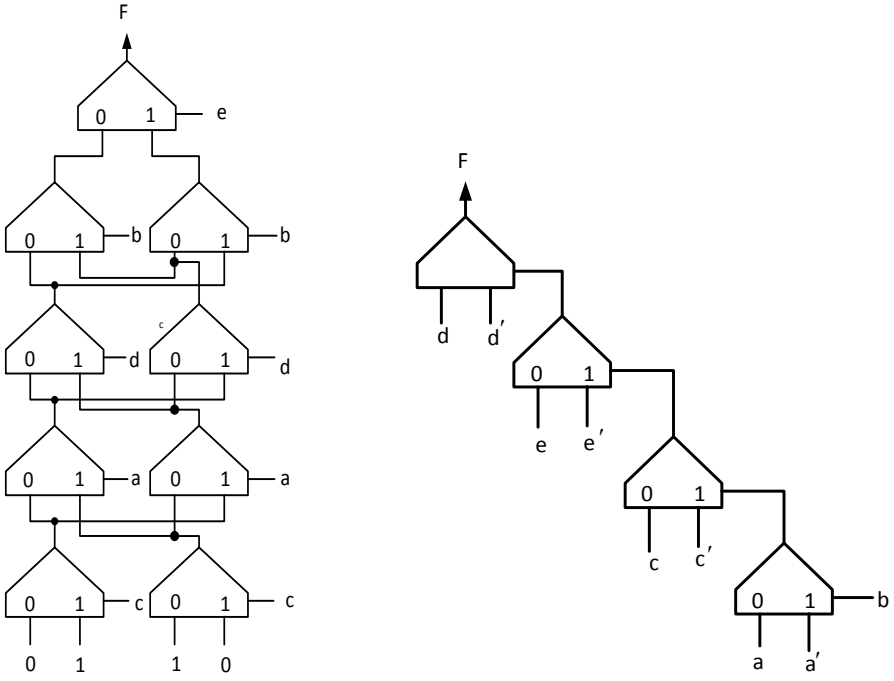


(a) CIM (b) VIM

Fig. 4.10 Realisation of 6one0246 using mux

3. xor5 – 5 Bit XOR Operation

It is very hard to realise XOR functions with more than four variables using mux (Aguirre *et al*. 2004). With the proposed techniques, it could be implemented with 9 units in 5 levels for CIM and 4 units in 4 levels for VIM instead of 31 units and 5 levels with SI. Thus the reduction in number of units with the proposed techniques CIM and VIM are 71% and 87% respectively compared to SI. The evolved circuits are shown in Figs. 4.11 (a) and (b) respectively.



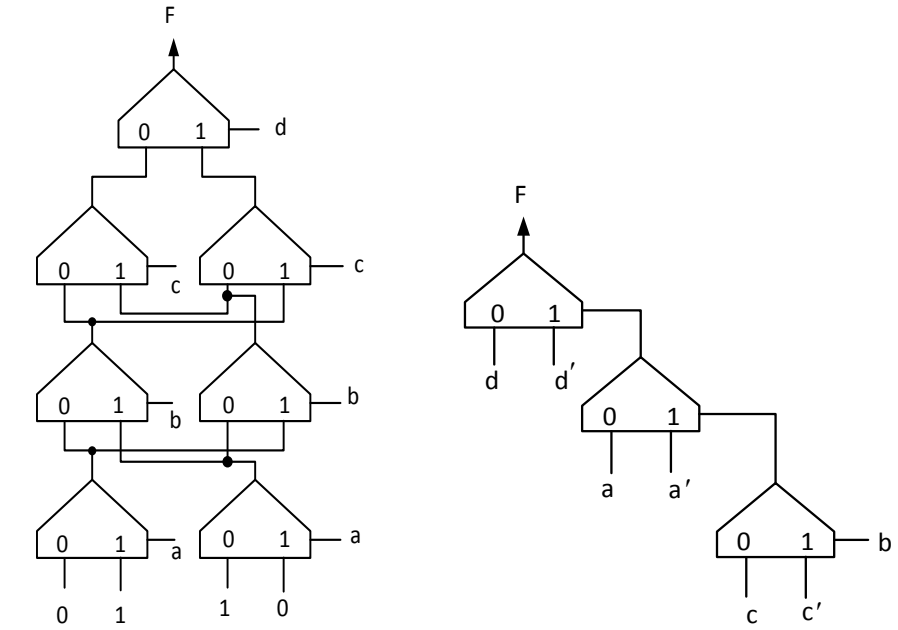
(a) CIM (b) VIM

Fig. 4.11 mux implementation of xor5

4. Four Bit Odd Parity Checker

The circuits generated are shown in Figs. 4.12 (a) and (b) for CIM and VIM respectively. For SI, the circuit needs 15 muxes in 4 levels. With the proposed

techniques CIM and VIM, the circuits needed 7 muxes in 4 levels and 3 muxes in 3 levels respectively. Thus, the saving in number of units is 53.3% in CIM and 80% in VIM. In the literature (Aguirre and Coello, 2004), it has been mentioned that parity circuits with 4 or more variables are difficult to realise using multiplexers but it can be seen that with the proposed methods, it is relatively simpler.

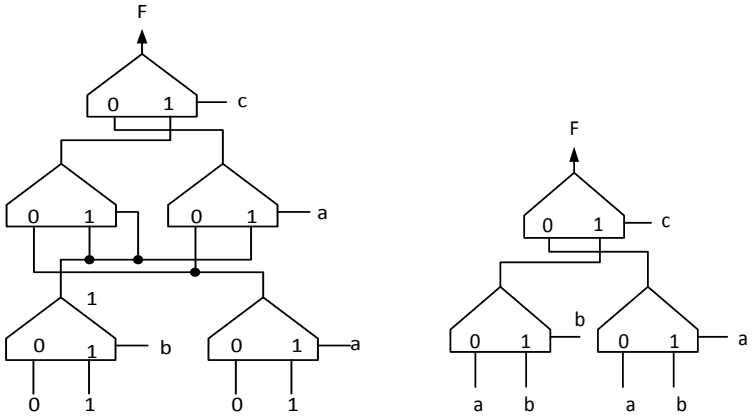


(a) CIM (b) VIM

Fig. 4.12 Circuit evolved for 4 bit odd parity checker using mux

5. Majority 3

Figs. 4.13 (a) and (b) shows its CIM and VIM implementations respectively with 5 units in 3 levels and 3 units in 2 levels respectively. In SI, the circuit requires 7 units in 3 levels.



(a) CIM (b) VIM

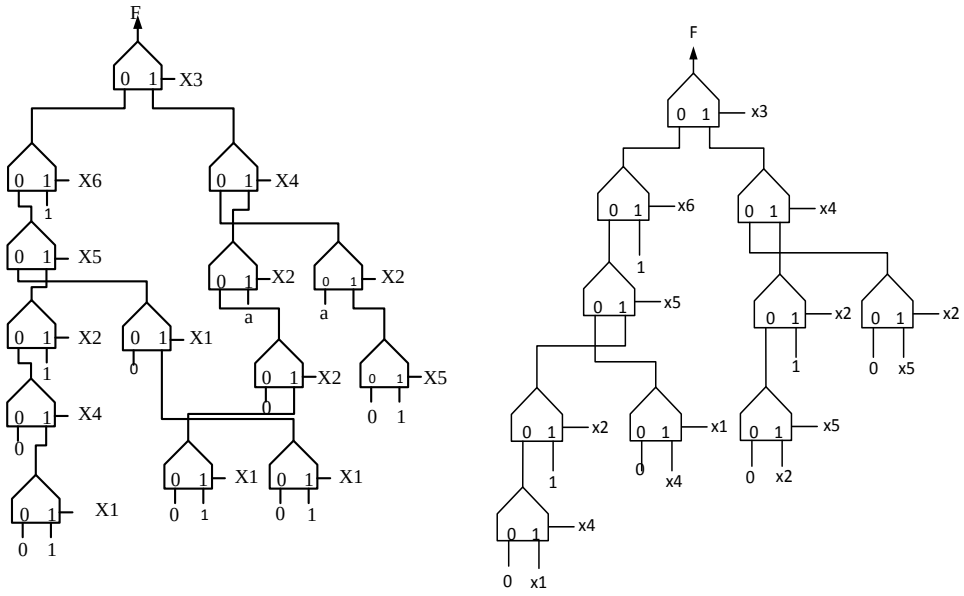
Fig. 4.13 Circuit generated for Majority 3 using mux

Other Functions

1. F = $X_1X_4 + X_2X_5 + X_3X_6$

F= Σ m (9, 11, 13, 15, 18, 19, 22, 23, 25, 26, 27, 29, 30, 31, 36, 37, 38, 39, 41, 43, 44, 45, 46, 47, 50, 51, 52, 53, 54, 55, 57, 58, 59, 60, 61, 62, 63)

This is a complex function and is very hard to evolve. The circuits evolved using CIM and VIM are shown in Figs. 4.14 (a) and (b) respectively. The circuit required 14 units / 6 levels in CIM and 10 units / 5 levels in VIM compared to 63 units / 6 levels in SI.

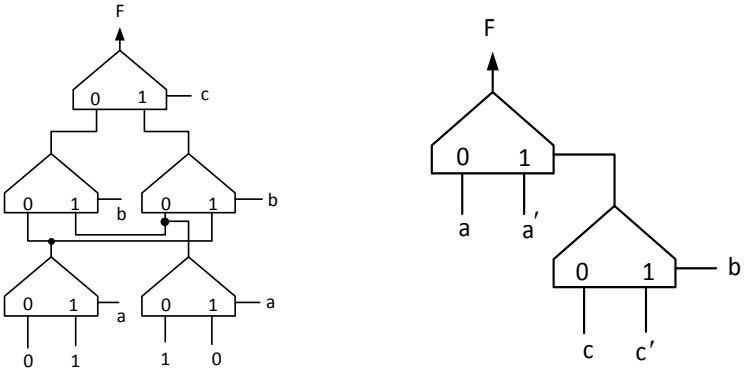


(a) CIM (b) VIM

Fig. 4.14 mux implementation of the function $F(X_1, X_2, X_3, X_4, X_5, X_6) = X_1X_4 + X_2X_5 + X_3X_6$

2. Three Bit Odd Parity Checker

The circuit evolved with CIM needs 5 units and 3 levels which is at par with the method proposed in literature (Aguirre and Coello, 2004) as shown in Fig. 4.15 (a). The circuit evolved by VIM shown in Fig. 4.15 (b) has 2 units in 2 levels compared to 5 units in 3 levels in the literature and 7 units in 3 levels with the conventional technique.

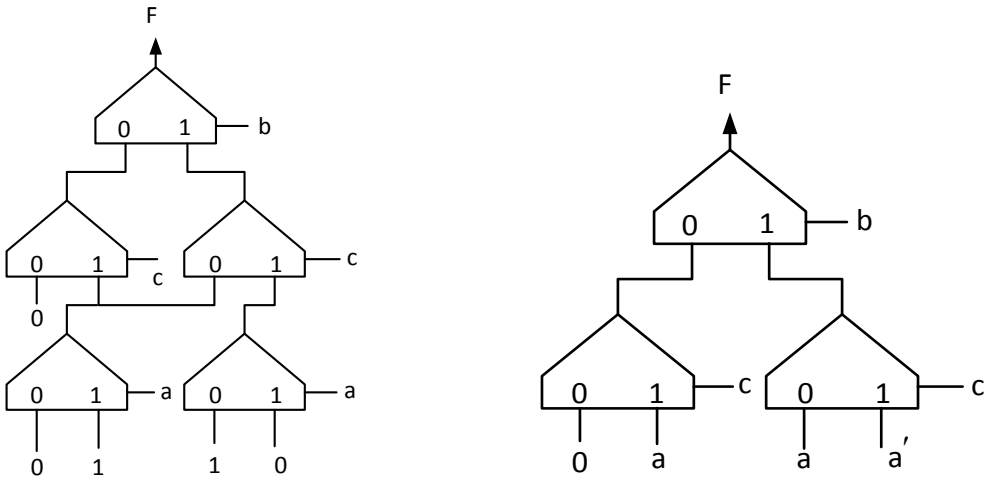


(a) CIM (b) VIM

Fig. 4.15 mux implementation of 3 bit odd parity checker

3. F (a, b, c) = Σ m **(3, 5, 6) -** a function which produces an output '1' when the number of '1's in the input combinations is two.

With genetic programming, (Aguirre *et al*. 1999 and 2004) needed 6 units in 3 levels with 0s and 1s as inputs. The proposed technique, CIM needs 5 units / 3levels and VIM requires 3 units / 2 levels as shown in Figs. 4.16 (a) and (b) respectively. Thus, compared to the method proposed in (Aguirre *et al*. 1999) there is a saving of 3 units and one level with VIM, and there is a saving of one unit in CIM. The implementation of the same circuit using standard technique is shown in Fig. 4.17 which requires 7 units in 3 levels.



(a) CIM (b) VIM

Fig. 4.16 Circuit generated for F (a, b, c) = Σ m (3, 5, 6) using mux

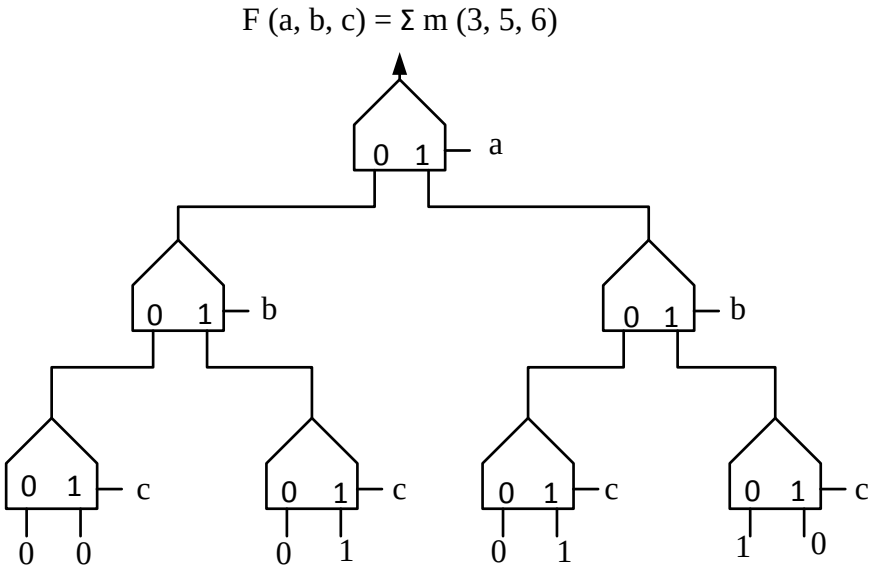


Fig. 4.17 Circuit for F (a, b, c) = Σ m (3, 5, 6) using mux with SI technique

Comparison between the proposed and existing methods

Circuits evolved by the proposed methods CIM and VIM are compared with SI and methods available in the literature in terms of number of units / levels. Table 4.2 shows the comparison of results between the conventional method and proposed methods in terms of units / levels. It is obvious that the number of units / levels needed is reduced significantly with the proposed methods. CIM involves only 0s and 1s as inputs to the units in first level. Hence the number of input buses needed is less compared to VIM. Implementation of circuits by VIM needs lesser number of units and levels compared to CIM and SI. As the number of modules is less, cost, area and power consumption is reduced which ensures better performance.

Table 4.2 Comparison of the proposed techniques with the existing techniques in terms of number of units/levels in mux implementation

Sl No.	Function	SI Units/ Levels	Aguirre *et al.* (2004) Units/ Levels	CIM Units/ Levels	VIM Units/ Levels
1	F=Σ m (3, 5 ,6)	7/3	6/3	5/3	3/2
2	F=Σ m (1, 2, 4)	7/3	5/3	5/3	3/2
3	Majority3 F=Σ m (3, 5, 6, 7)	7/3	-	5/3	3/2
4	3bit odd parity	7/3	5/3	5/3	2/2
5	F = Σ m (5, 6)	7/3		5/3	3/2
6	F = Σ m (0, 2, 3 ,4, 6)	7/3	-	4/3	2/2
7	F = Σ m (0, 2, 4, 6)	7/3		2/2	1/1
8	4bit odd parity	15/4	-	7/4	3/3
9	F = Σ m (4, 5, 6, 7, 8, 9, 10, 13)	15/4		5/3	2/2
10	F = Σ m (5, 6, 9, 10)	15/4	-	6/4	3/3
11	F = Σ m (1, 2, 3, 5, 7, 8, 12)	15/4	-	7/4	4/4
12	F = Σ m (13, 14)	15/4		6/3	3/2
13	xor5	31/5	-	9/5	4/4
14	F = Σ m (0, 4, 6, 7, 8, 12, 14, 15)	15/4		3/2	1/1
15	F = Σ m (0, 1, 9, 22, 23, 25, 30, 31)	31/5	-	10/5	6/4
16	F = Σ m (9, 11, 25, 27, 29, 31)	31/5	-	7/5	5/5
17	F = Σ m (3, 7, 8, 15, 19, 23, 24, 26, 27, 31)	31/5	-	8/3	4/2
18	F = Σ m (0, 1, 2, 3, 16, 17, 18, 19, 32, 33, 34, 35, 48, 49, 50, 51, 60, 61, 62, 63)	63/6	-	5/4	3/3
19	6one 135	63/6	-	11/6	5/5
20	6 one 0246	63/6	-	11/6	5/5
21	F = X1X4 + X2X5 + X3X6	63/6	-	14/6	10/5

The evolved circuits were synthesised on FPGA Spartan3 (device XC3S400) using Xilinx ISE 14.2 and hence the delay, device utilisation and power are estimated. Table 4.3 shows the delay associated with each circuit in standard implementation and the proposed techniques. The delay includes both the delay of logic units and the delay

caused by the RC component of the interconnecting wires. It can be observed that there is a considerable reduction in delay with VIM compared to the other two methods.

Table 4.3 Comparison of delay in various methods of mux implementation

Function	SI		CIM		VIM	
	No. of Levels	Delay (nsec)	No. of Levels	Delay (nsec)	No. of Levels	Delay (nsec)
Majority 3	3	8.138	3	7.760	2	6.305
4 bit odd parity checker	4	9.215	4	7.760	3	6.816
xor5	5	9.615	5	9.09	4	8.138
6one135	6	10.22	6	9.357	5	8.943
6one0246	6	10.22	6	9.412	5	8.943

Table 4.4 shows the area in terms of device utilisation of the circuits generated by the proposed techniques and the circuits by conventional method. It can be seen that the number of Look-Up-Tables (LUTs) / slices / Input Output Blocks (IOBs) have been reduced considerably for the circuits with the proposed techniques.

Table 4.4 Comparison of device utilisation in various methods of mux implementation

Function	Device utilization	SI	CIM	VIM
Majority 3	No.of slices	1	1	1
	LUTs	2	1	1
	IOBs	6	6	4
4 bit odd parity checker	No.of slices	3	1	1
	LUTs	4	2	1
	IOBs	7	7	5
Xor5	No.of slices	3	1	1
	LUTs	4	2	2
	IOBs	8	8	6
6one135	No.of slices	3	2	1
	LUTs	5	3	2
	IOBs	9	9	7
6one0246	No.of slices	3	2	1
	LUTs	5	3	2
	IOBs	9	9	7

The power consumed by the circuits generated in CIM, VIM and Standard Implementation technique are shown in Table 4.5. It can be observed that the power consumption has been reduced in the proposed techniques even in FPGA implementation, which indicates that power reduction can be ensured in ASIC implementation.

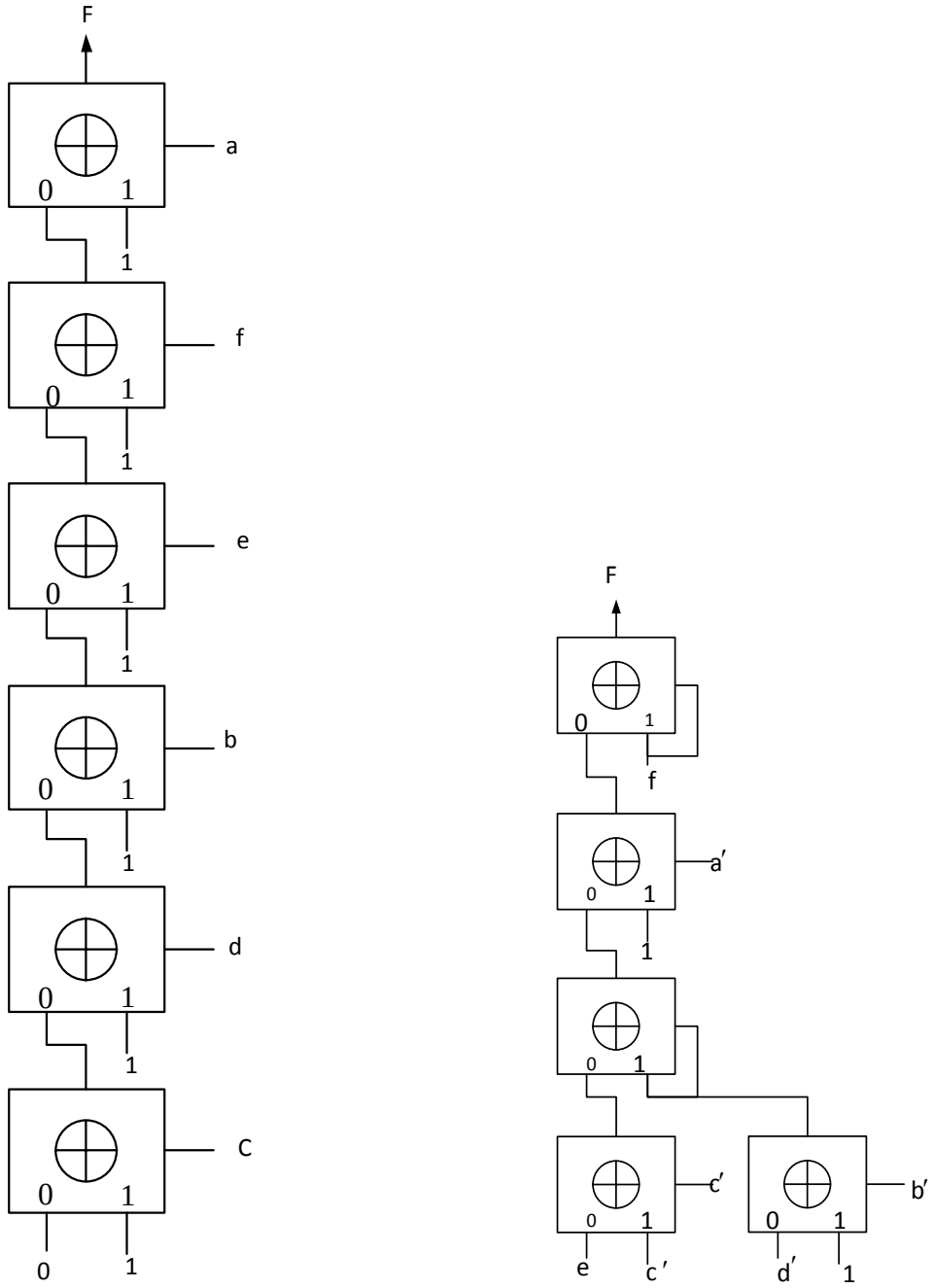
Table 4.5 Comparison of power consumption in various methods of mux implementation

Function	**Power in mw**		
	SI	**CIM**	**VIM**
Majority3	0.149	0.147	0.141
4 bit odd parity checker	0.153	0.147	0.143
Xor5	0.154	0.151	0.149
6one135	0.158	0.153	0.152
6one0246	0.158	0.154	0.150

4.6.2 Realisation of Circuits Using RM ULM

The benchmark functions listed in Table 1.1 are used for the validation of the proposed techniques using RM ULM.

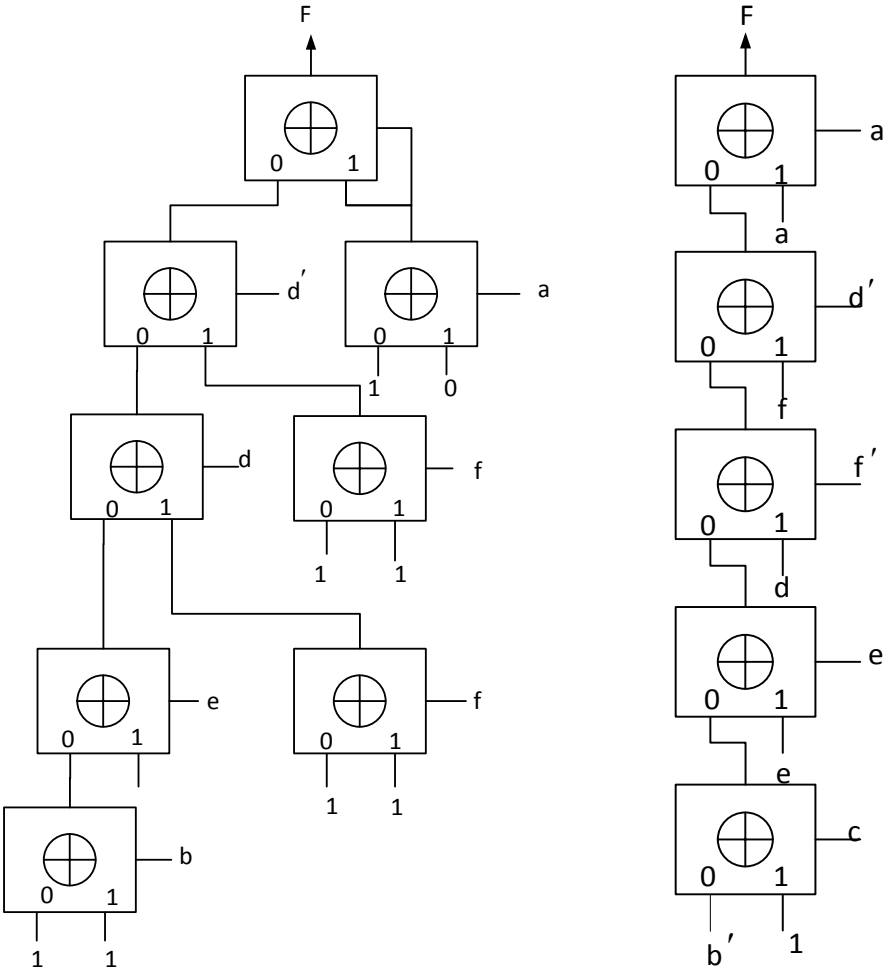
1. **6one135**- The circuit was realised using 6 units in 6 levels with CIM and 5 units in 4 levels with VIM compared to 63 units in 6 levels with the conventional technique as shown in Figs. 4.18 (a) and (b) respectively.



(a) CIM (b) VIM

Fig. 4.18 Circuit evolved for 6one135 using RM ULM

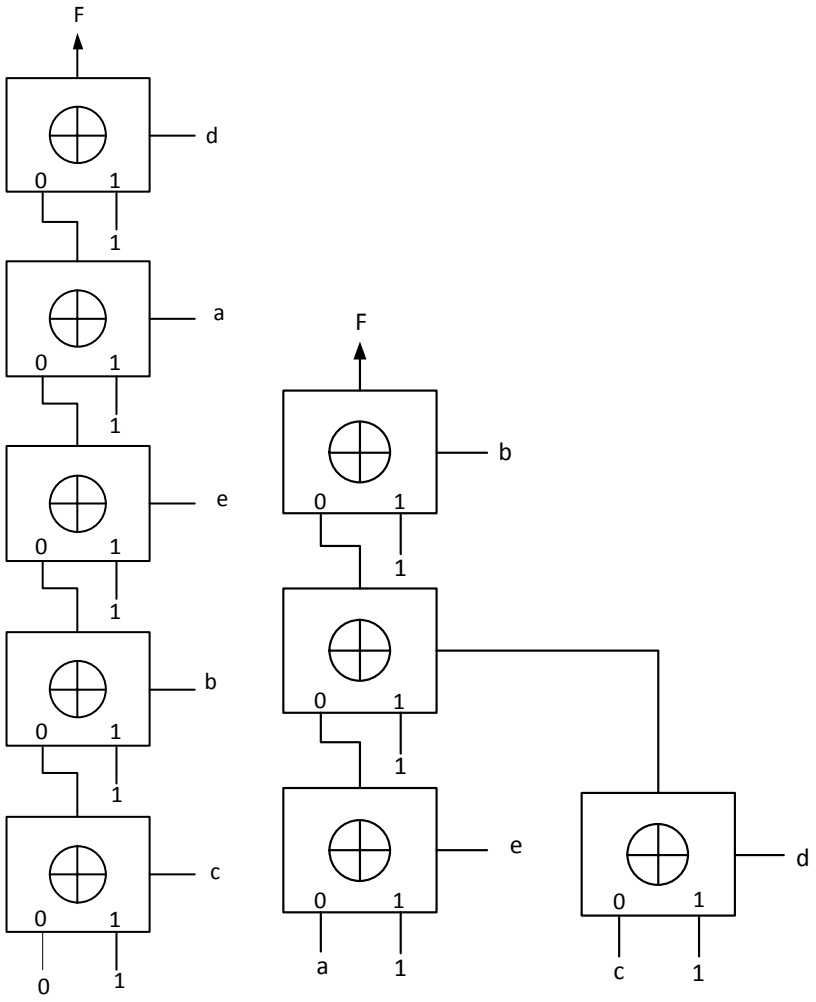
2. **6one0246** - The circuit needs only 8 units in 5 levels with CIM and 5 units in 5 levels with VIM as shown in Fig. 4.19, whereas SI technique requires 63 units and 6 levels.



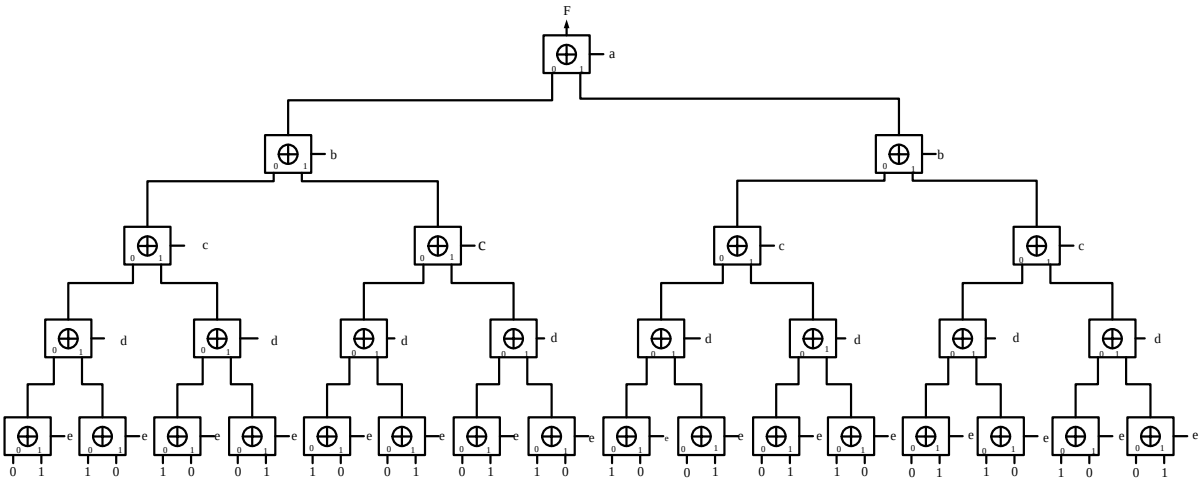
(a) CIM (b) VIM

Fig. 4.19 Realisation of 6one0246 using RM ULM

3. **xor5** - The circuit needs only 5 units in 5 levels with CIM, whereas with VIM the function could be realised with 4 units in 3 levels as shown in Fig. 4.20 (a) and (b) respectively. Fig. 4.20 (c) shows the RM realisation of xor5 by standard implementation technique which requires 31 units in 5 levels. Thus there is a saving of 26 units in CIM and 27 units in VIM respectively. Though the number of levels in CIM remains the same as in SI, VIM shows a reduction in one level.



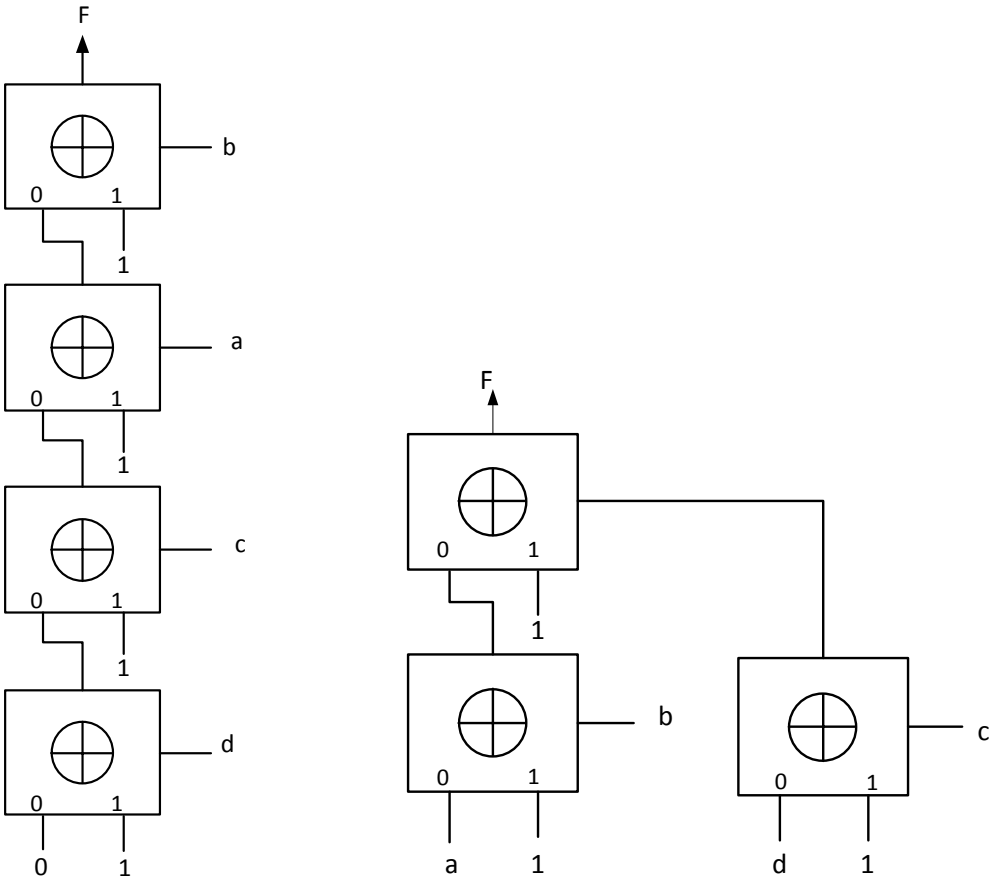
(a) CIM (b) VIM



(c) Standard implementation

Fig. 4.20 RM implementation of xor5

4. **4 bit odd parity checker -** The circuit generated (Fig.4.21 (a)) has only 3 units in 2 levels in VIM as compared to 15 units and 4 levels in SI. With CIM, the circuit can be realised using 4 units and 4 levels, which is far better than SI. Thus on reducing the number of units, the cost, area and power can be reduced and on getting the number of levels reduced from 4 to 2, the delay involved is reduced. The same function using mux needs 3 units and 3 levels whereas its RM implementation accommodates the 3 units in 2 levels. Thus RM implementation is found to be better for XOR based operations. At the same time GA converges very fast with RM logic for parity checker circuits.



(a) CIM (b) VIM

Fig. 4.21 Evolved circuit for a Four bit odd parity checker using RM ULM

5. ***Majority3*** –Fig. 4.22 shows the circuits evolved for this function using the proposed methods. It can be seen that the circuits require 5 units in 3 levels and 3 units in 2 levels in CIM and VIM respectively.

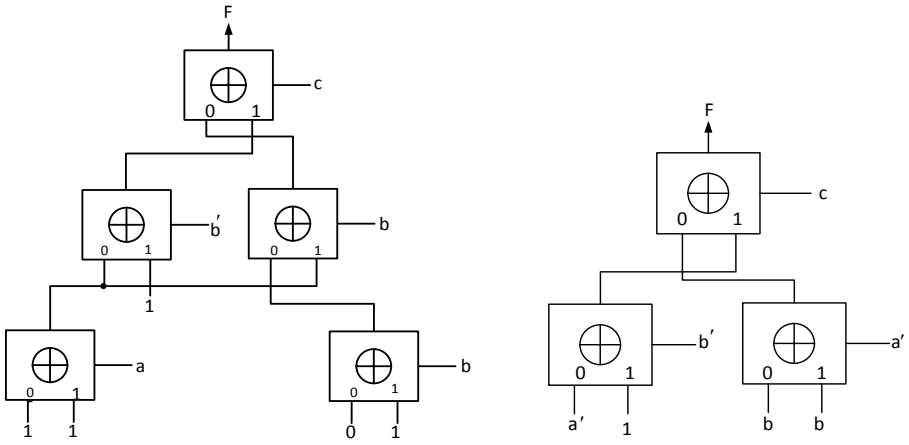
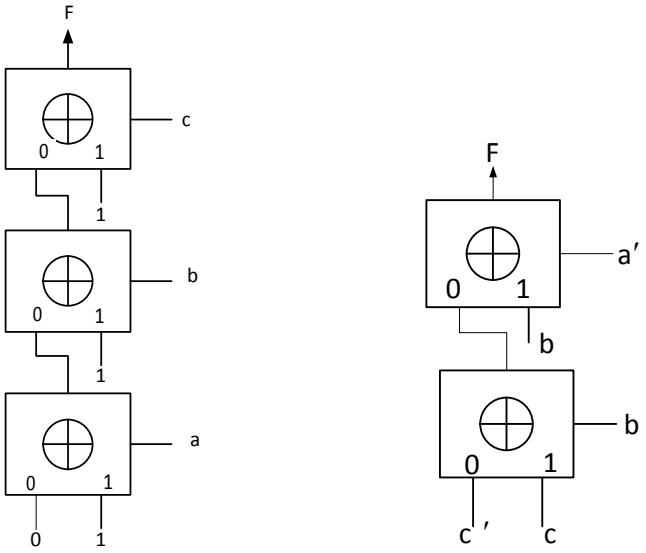


Fig. 4.22 Circuit evolved for Majority3 using RM ULM

Other functions

1. **3 bit odd parity checker-** The circuit evolved with CIM needs 3 units and 3 levels, but with VIM, the circuit needs only 2 units in 2 levels as shown in Fig. 4.23. Thus in VIM, a saving of 5 units is achieved in comparison with the conventional method and a saving of 1 unit and 1 level in comparison with CIM.

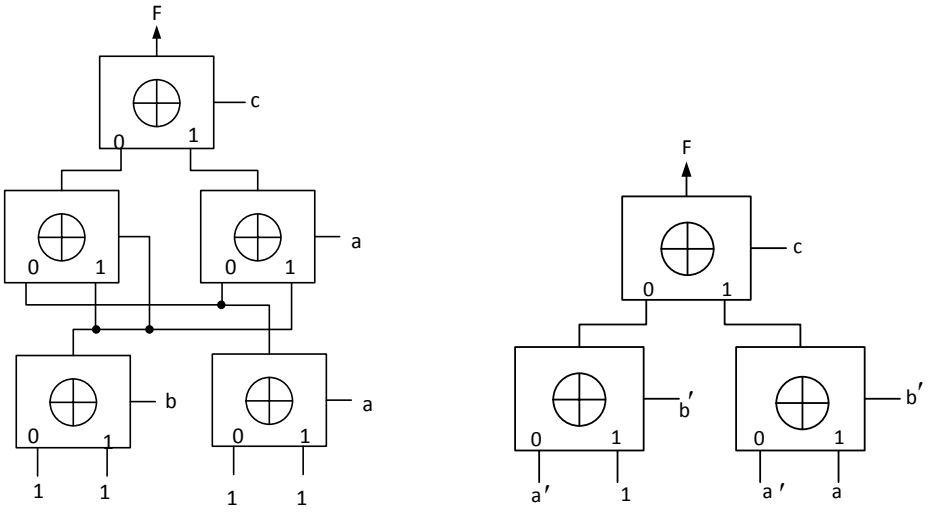


(a) CIM (b) VIM

Fig. 4.23 Three bit odd parity checker using RM ULM

2. **F (a, b, c) = ∑ m (1, 2, 4)**

Figs. 4.24 (a) and (b) shows the circuits evolved by CIM and VIM for the RM implementation of this function.

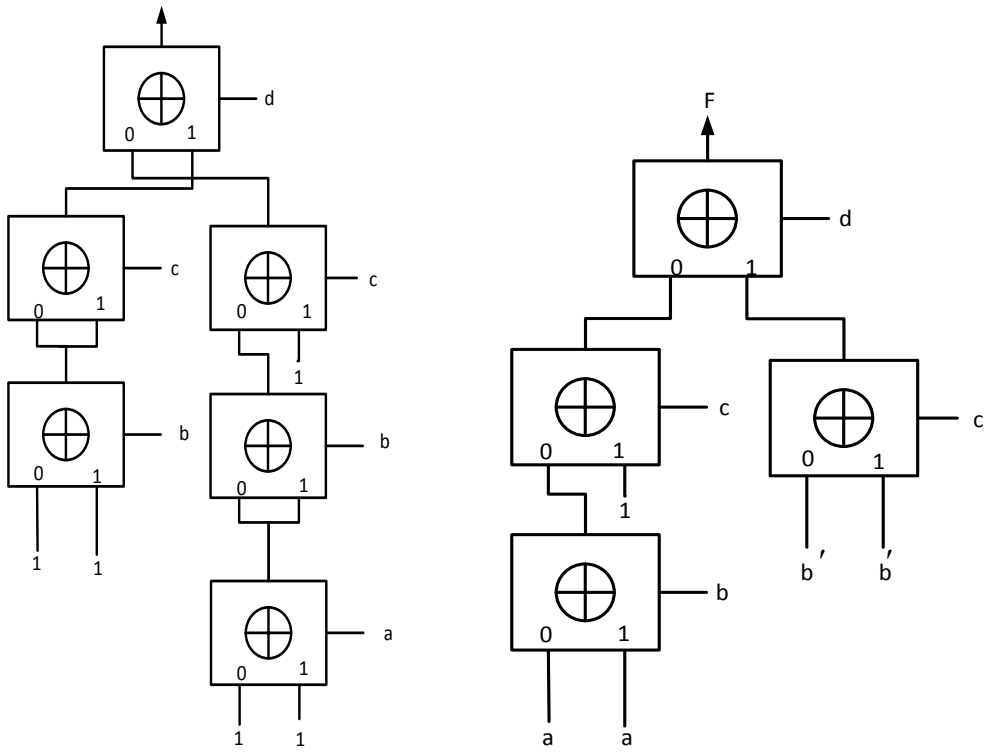


(a) CIM (b) VIM

Fig. 4.24 Circuit generated for F (a, b, c) = ∑ m (1, 2, 4) using RM ULM

3. **F (a, b, c, d) = ∑ m (1, 2, 3, 5, 7, 8, 12)**

This is an example taken from (Oh and Almaini, 2007) based on 2VROBDD which required 6 units in 4 levels. But with the proposed techniques, the function could be implemented with lesser number of units / levels. With CIM, the circuit needs 6 units and 4 levels whereas with VIM, it needs only 4 units in 3 levels as shown in Figs. 4.25 (a) and (b) respectively. Thus there is a saving of 2 units and 1 level compared to the literature in VIM.



(a) CIM (b) VIM

Fig. 4.25 RM implementation of F (a, b, c, d) = Σ m (1, 2, 3, 5, 7, 8, 12)

Table 4.6 shows a comparison of the proposed methods with SI technique in terms of number of units / levels. It is observed that the proposed methods, CIM and VIM use minimum hardware in lesser number of levels compared to SI.

Table 4.6 Comparison of the circuits evolved by various methods in terms of number of units / levels in RM implementation

Sl. No.	Function	SI Units/ Levels	CIM Units / Levels	VIM Units / Levels
1	F=Σ m (0, 1, 2, 4)	7/3	5/3	3/2
2	F=Σ m (3, 5, 6)	7/3	4/4	3/3
3	F=Σ m (1, 2, 4)	7/3	5/3	3/2
4	Majority3 F = Σ m (3, 5, 6, 7)	7/3	5/3	3/2
5	3bit odd parity	7/3	3/3	2/2
6	F (a, b, c) = Σ m (5, 6)	7/3	5/3	2/2
7	F (a, b, c) = Σ m (0, 1, 2, 4, 6)	7/3	2/2	1/1
8	4bit odd parity	15/4	4/4	3/2
9	F (a, b, c, d) = Σ m (4, 5, 6, 7, 8, 9, 10, 13)	15/4	6/4	3/3
10	F (a, b, c, d) = Σ m (5, 6, 9, 10)	15/4	4/3	3/2
11	F(a, b, c, d) = Σ m (1, 2 ,3, 5, 7, 8, 12)	15/4	6/4	4/3
12	F(a, b, c, d) = Σ m (13, 14)	15/4	5/3	3/2
13	F(a, b, c, d) = Σ m (0, 4, 6, 7, 8, 12, 14, 15)	15/4	5/3	2/2
14	xor5	31/5	5/5	4/3
15	F (a, b, c, d, e) = Σ m (0, 1, 9, 22, 23, 25, 30, 31)	31/5	10/5	6/4
16	F (a, b, c, d, e) = Σ m (9, 11, 25, 27, 29, 31)	31/5	7/5	5/5
17	6one 135	63/6	6/6	5/4
18	6 one 0246	63/6	8/5	5/5
19	F = X1X4 + X2X5 + X3X6	63/6	14/6	9/5

The circuits generated by the proposed techniques are synthesised on FPGA Spartan3 (device XC3S400) using Xilinx ISE 14.2 and hence the delay, device utilisation and power are estimated. Table 4.7 shows the delay associated with each circuit in standard implementation and the proposed techniques using RM realisation. It can be observed that there is a considerable reduction in delay with VIM compared to the other two methods.

Table 4.7 Delay in RM implementation by various methods

Function	SI		CIM		VIM	
	No. of Levels	Delay (nsec)	No. of Levels	Delay (nsec)	No. of Levels	Delay (nsec)
Majority3	3	8.138	3	8.074	2	7.760
4 bit odd parity	4	9.215	4	8.979	3	8.138
Xor5	5	10.566	5	9.215	4	8.190
6one135	6	10.443	6	9.893	5	9.09
6one0246	6	10.443	6	9.103	5	8.943

The area occupied by the circuits is compared by considering the number of LUTs, slices and IOBs utilised by each circuit. Table 4.8 shows the comparison of the device utilisation of the circuits in proposed techniques and in SI technique. It is obvious that the circuits generated by VIM occupy lesser area compared to the other two methods.

Table 4.8 Device utilisation of various circuits in SI and proposed techniques.

Function	Device utilisation	SI	CIM	VIM
Majority3	No.of slices	1	1	1
	LUTs	2	1	1
	IOBs	6	6	4
4 bit odd parity checker	No.of slices	3	1	1
	LUTs	6	2	1
	IOBs	7	7	6
xor5	No.of slices	4	1	1
	LUTs	7	2	1
	IOBs	8	8	7
6one135	No.of slices	4	1	1
	LUTs	7	2	2
	IOBs	9	9	8
6one0246	No.of slices	4	1	1
	LUTs	7	2	2
	IOBs	9	9	8

The power consumption of circuits generated by the proposed techniques are compared with the power consumed by circuits in SI and the results are tabulated as in Table 4.9. It can be observed that power consumption is less in the proposed techniques than the standard technique even in FPGA implementation. Hence power consumption will be less in circuits with the proposed techniques in ASIC implementation.

Table 4.9 Comparison of Power consumption in various methods by RM implementation

Function	**Power in mw**		
	SI	**CIM**	**VIM**
Majority3	0.151	0.148	0.147
4 bit odd parity checker	0.150	0.146	0.116
xor5	0.152	0.150	0.136
6one135	0.153	0.152	0.142
6one0246	0.154	0.153	0.151

Comparison between the circuits generated using RM ULM and 2-1 mux

A detailed analysis has been made on the performance of circuits evolved using the two ULMs (binary mux and binary RM ULM) by the proposed methods. A comparison has been made on the power consumption, number of units and the number of levels required for the realisation of various functions. Table 4.10 shows the comparison of power consumption in various methods of circuit implementation. It can be observed that the power required has been reduced in both CIM and VIM compared to SI. For XOR based circuits, RM implementation in VIM has the least power requirement. For other circuits, mux implementation in VIM consumes least power.

Table 4.10 Power consumption in various methods of circuit implementation

Function	**Power in SI (mw)**		**Power in CIM (mw)**		**Power in VIM (mw)**	
	mux	**RM**	**mux**	**RM**	**mux**	**RM**
Majority3	0.149	0.151	0.147	0.148	0.141	0.147
4 bit odd parity checker	0.153	0.150	0.147	0.146	0.143	0.116
xor5	0.154	0.152	0.151	0.150	0.149	0.136
6one135	0.158	0.153	0.153	0.152	0.152	0.142
6one0246	0.158	0.154	0.154	0.153	0.150	0.151

Fig. 4.26 shows a comparison between the circuits based on mux and RM ULM using CIM in terms of the number of units. The functions F1 to F4 are parity functions and it can be observed that RM based implementation involves lesser number of units for these functions. For the other functions, implementation using mux involves equal / lesser number of units.

The functions used are:

F1: 4 bit odd parity

F2: xor5

F3: 6one135

F4: 6one0246

F5: Majority3

F6: $F(a, b, c, d) = \Sigma m (4, 5, 6, 7, 8, 9, 10, 13)$

F7: $F(a, b, c, d, e) = \Sigma m (3, 7, 8, 15, 19, 23, 24, 26, 27, 31)$

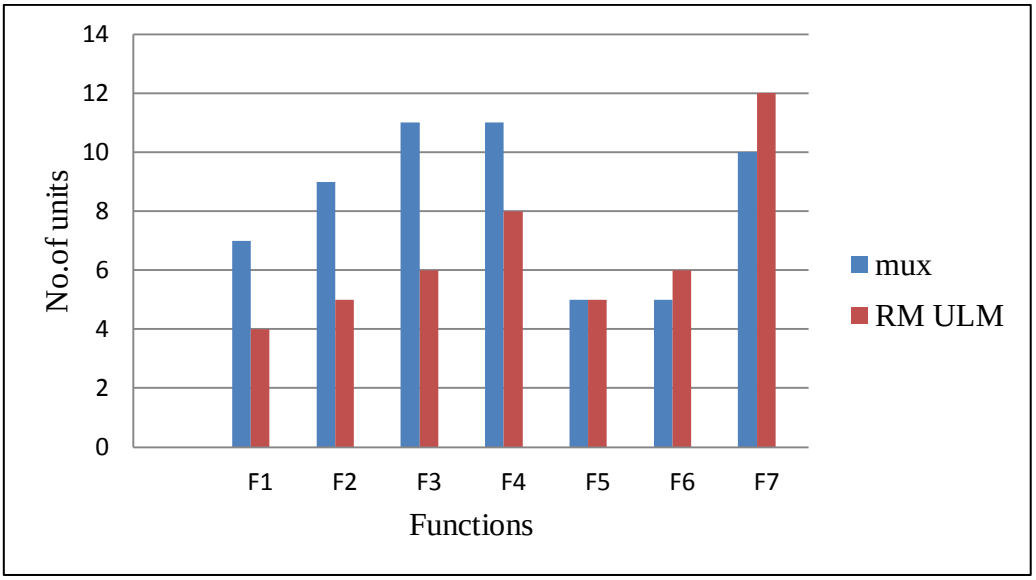


Fig. 4.26 Comparison of CIM in terms of number of units in mux and RM implementations

Fig. 4.27 depicts the comparison of mux and RM implementations in VIM in terms of number of units. It can be seen that these methods perform equally good in both the implementations for XOR based circuits, while for other circuits, mux implementation is better if number of units required is to be minimum.

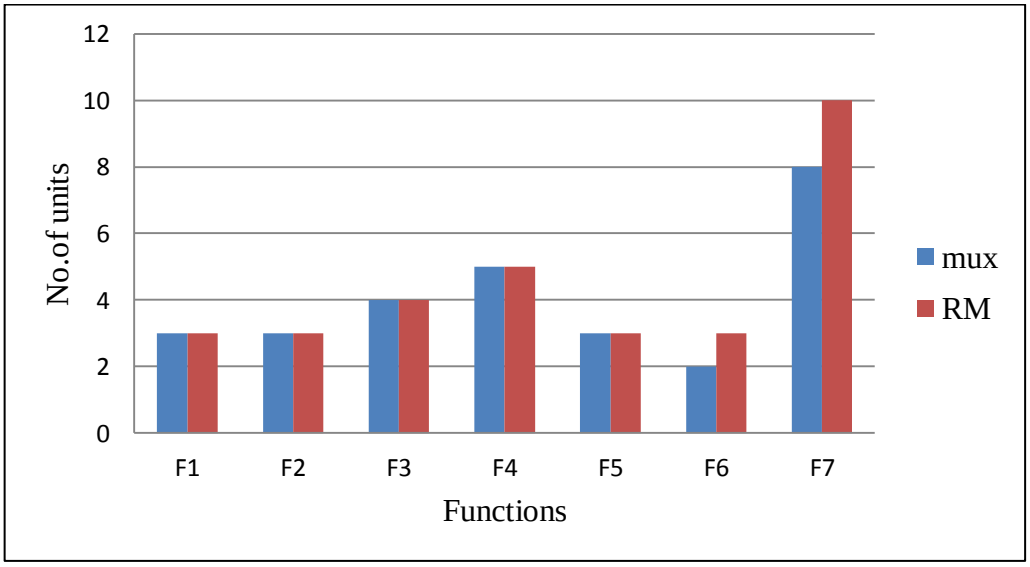


Fig.4.27 Comparison of VIM in terms of number of units in mux and RM implementations

Fig. 4.28 illustrates a comparison on the number of levels for various circuits using mux / RM ULM in the proposed methods. It can be observed that in both the implementations, variable input method requires lesser number of levels. For parity based circuits, RM implementation in VIM uses lesser number of levels. For other circuits, number of levels with mux implementation is either less or equal to RM implementation.

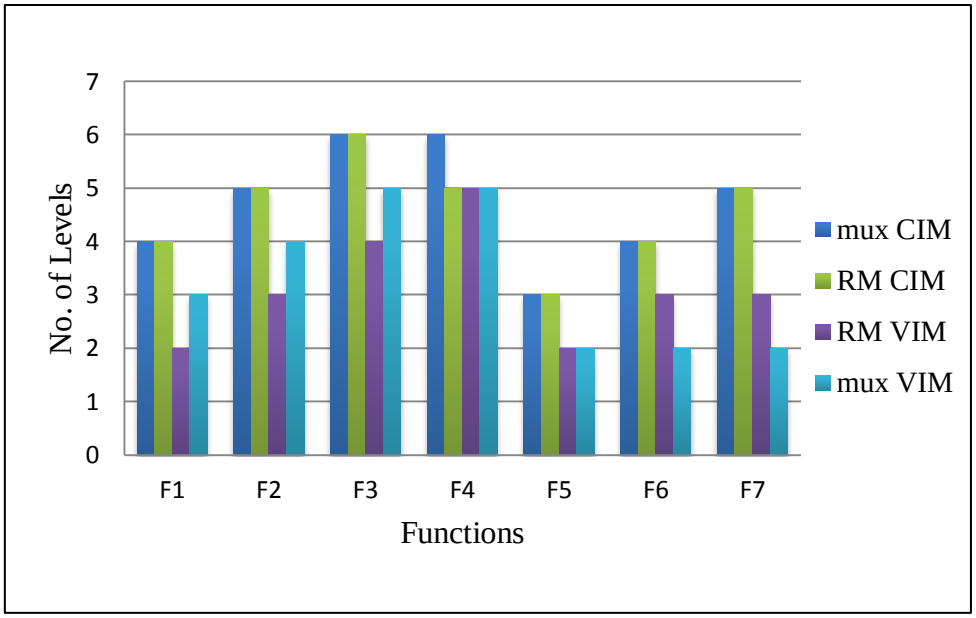


Fig. 4.28 Comparison of proposed methods in terms of number of levels

To conclude, from Figs. 4.26, 4.27, and 4.28, it is observed that the number of units / levels required is minimum for XOR based circuits in VIM using RM ULM, whereas for other circuits, circuits with minimum hardware / levels are evolved for VIM using mux.

4.7 SUMMARY

Two simplified and efficient techniques referred to as CIM and VIM have been proposed for the design of combinational circuits which uses lesser number of units, interconnections and levels. The design based on the Universal building blocks,

2-1 mux / 2-1 RM ULM uses GA as the optimisation tool. The evolved circuits were synthesised on FPGA Spartan 3 -XC3S400 device using Xilinx ISE 14.2. VHDL is used to describe the developed design.

Functions up to 6 variables have been generated and detailed analysis of the evolved circuits for number of units, area and delay has been made. The results were compared with the methods available in the literature. Based on the analysis, it was found that the CIM is superior to SI technique. But VIM outperforms CIM also in terms of units / levels. It was observed that the VIM using RMULM evolves circuits with minimum hardware / delay for XOR based circuits and for other circuits, mux implementation is at par or better with reduced number of units / levels. The results are validated using benchmark functions.

CHAPTER 5

DESIGN OF SEQUENTIAL CIRCUITS

5. 1 INTRODUCTION

A lot of research has been done in the field of combinational circuits, while the design of sequential circuits is still in the toddler stage. Most real world electronic products use sequential circuits to send, receive, store, retrieve and process information stored in binary fashion. Hence there is a need for its optimised design. Sequential logic circuit (SLC) constitutes a combinational part and memory elements such as flip flops to store the states. SLCs can be classified into Synchronous (SSCs) and Asynchronous Sequential Circuits (ASCs) as mentioned in Section 1.1.

This thesis focuses on the design of SSCs. A modified GA has been proposed to obtain the OSA. The complexity of the combinational part of SSC is determined by the state assignment and hence finding the best state assignment plays a major role in the design of sequential circuits so as to minimise the chip area and hence the cost.

The behavior of an SSC can be represented by an FSM. It is a mathematical model of the sequential circuit with discrete inputs, discrete outputs, and internal states (Ali *et al.* 2004). There are two types of FSM, namely Moore machine and Mealy machine. In Moore machine the output depends on the states alone where as in Mealy machine, the output depends on the inputs as well as the states (Mano, 2002) as shown in Figs. 5.1 and 5.2 respectively. The states of a system refer to the information of the past inputs which in turn determines the behavior of the system on the application of

subsequent inputs. The states are stored in the flip flops. The combinational part is used to produce the next state and the output of SSC. Therefore, an SSC consists of two combinational parts, viz., one to determine the next state and the other to determine the output of the system. The complexity of the circuit is completely determined by the number of logic gates involved in the combinational part.

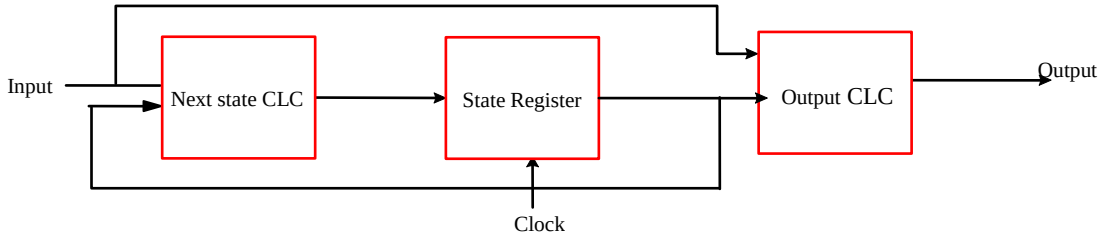


Fig. 5.1 Block diagram of Mealy machine

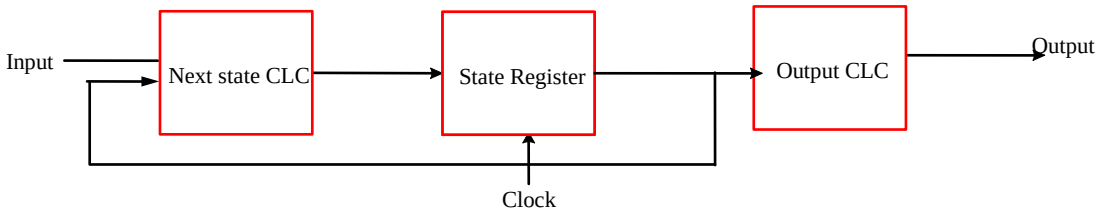


Fig. 5.2 Block diagram of Moore machine

In the case of CLCs, a truth table completely specifies the circuit, where as an SSC requires a STT or state transition diagram for specifying the circuit.

5.2 STATE TRANSITION TABLE

A state transition table or simply state table enumerates the time sequence of inputs, outputs and states of the flip flops. It consists of three sections namely the present state, next state and output. The STT can be prepared only if appropriate state assignment is done and from

the state table, the excitation table is generated corresponding to the type of flip flops chosen. Here D / T flip flops are used. The desired combinational circuits are evolved from the respective excitation tables by using the methods proposed in chapters 3 and 4.

Thus, the first step in the design of SSCs is to obtain the OSA.

5.3 STATE ASSIGNMENT

In an FSM, each state is to be identified by a string of bits. Assigning a unique binary code to each of the states of the FSM is termed as state assignment. The process of obtaining a relationship between the states and the bit strings which result in minimal cost is referred to as the problem of OSA (Amaral *et al*. 1995). The design of OSA is crucial as it determines the complexity of the CLCs to be used. Once OSA is obtained, the corresponding STT is prepared.

If the number of states is N, then the number of state variables **s** is the smallest integer that is equal to or greater than $|\log_2 N|$ and the total number of possible state is equal to 2^s.

The assignment process decides which of these 2^s states must be assigned to any particular state in the FSM. Total number of possible encodings is given by (Ali, 2003)

$$T(n,s) = \frac{2^s!}{(2^s - n)!} \tag{5.1}$$

Thus, for a circuit with 4 states, the possible number of encodings is 24. For a 5 state machine the number of encodings goes up to 6720. Since the number of possible state assignments grows enormously with the number of internal states, it is almost impossible to try all assignment manually in order to select the one which leads to the simplest logic

circuit. Use of evolutionary algorithms is a better alternative for the OSA of FSMs. Here a modified GA is used specifically to cater the needs of SSCs.

5.4 MODIFIED GENETIC ALGORITHM

Much research has been done in this area to get an OSA in FSMs. Prior to the use of GA, several algorithms such as NOVA, Mustang etc. were used to obtain the state assignment. In this work, GA is used to obtain the OSA with the help of Desired Adjacency Graph (DAG). DAG is an undirected, fully connected graph with states as its nodes. In an SSC implementation, the desirability of having two states adjacent to each other is determined by the strength of the arc connecting the two nodes. For an SSC to have minimum cost, the distance between the states that are strongly connected in the DAG is to be minimum.

Thus the steps involved in the problem of state assignment are

i) Obtain the weight of each arc of the DAG

ii) Apply GA to find the minimum hamming distance between the strongly connected states to evolve the OSA.

To determine the strength of a connection in DAG, the following heuristic rules are applied on the DAG (Amaral *et al*. (1995).

State assignment should be done in such a way that

- the distance between states that are in the same set of successors should be minimum
- the distance between states that are in the same set of predecessors of a given state with a given input condition should be minimum

- the distance between the states that are in the same partition should be minimum

A state S_i is called a successor of state S_j if there is a transition from state S_j to S_i. Similarly a state S_i is said to be a predecessor of state S_j if there is a transition from state S_i to S_j. States S_i and S_j are said to be *associated* with each other if both of them are a successor of a given state, if both of them are in the set of predecessors of a state with a given input condition or if both of them are in the same partition of an output. Each output is said to partition the states of an FSM into subsets. The set of partitions of an output Z_k is denoted by O (Z_k). For Moore machines, output is given by Z_k (S_i). In Mealy machines, it is represented as Z_k (S_i, I_a) where I_a is an input condition represented in binary. Distance between two states S_i and S_j is termed as the hamming distance and is denoted as D (S_i, S_j). In an FSM with p input signals, there are $c = 2^p$ input conditions.

The strength of the connection between state i and state j is denoted by DAG_{ij} or DAG_{ji} and is given by (Amaral *et al*. 1995).

$$DAG_{ij} = DAG_{ji} = R_1 \sum_{l=0}^{s-1} \alpha_{li}\alpha_{lj}\delta_{ij} + R_2 \sum_{a=0}^{c-1}\sum_{l=0}^{s-1} \beta_{lis}\beta_{lja}\delta_{ij} + R_3 M \sum_{b=0}^{v-1} \gamma_{ijb}\delta_{ij} + R_3(1-M)\sum_{a=0}^{c-1}\sum_{b-0}^{v-1} \phi_{ijab}\delta_{ij} + R_4(\alpha_{ij}+\alpha_{ji})\delta_{ij} \quad (5.2)$$

Where c is the no. of input conditions, v is the number of output variables, s is the number of states, and

$$\alpha_{lm} = \begin{cases} 1 \text{ if } S_m \epsilon S_{uc}(S_l) \\ 0 \text{ otherwise} \end{cases}$$

$$\beta_{lma} = \begin{cases} 1 \text{ if } S_m \epsilon P_{red}(S_l, I_a) \\ 0 \text{ otherwise} \end{cases}$$

$$\gamma_{ijb} = \begin{cases} 1 \text{ if } Z_b(S_i) = Z_b(S_j) \\ 0 \text{ otherwise} \end{cases}$$

$$\delta_{ij} = \begin{cases} 1 \text{ if } i \neq j \\ 0 \text{ if } i = j \end{cases}$$

$$\phi_{ijab} = \begin{cases} 1 \text{ if } Z_b(S_i, I_a) = Z_b(S_j, I_a) \\ 0 \text{ otherwise} \end{cases}$$

$$M = \begin{cases} 1 \text{ for Moore machines} \\ 0 \text{ for Mealy machines} \end{cases}$$

R_1, R_2, R_3 and R_4 are constant coefficients which are fixed on the significance of each individual rule.

First term of Eq. (5.2) corresponds to the pair of states that are common successors to a particular state (rule1). The second term stands for the pair of states that are common predecessors to a particular state (rule 2). The third and fourth terms correspond to pairs of states that are in the same output partition for a given output (rule 3). The last term indicates transitions between two states. It is used when the relative position of each state is difficult to know from the previous terms.

Since the DAG is an undirected and fully connected graph, the values of its connections can be represented by a symmetric square matrix. Here the coefficients of R are fixed as $R_1 = 3$, $R_2 = 4$, $R_3 = 2$ and $R_4 = 1$ (Comer, 1984). The SSC cost is lowered when the distance between two states with strong connections in the DAG is minimum. Thus, if DAG_{ij} is large, D (S_i,S_k) should be small. For a given FSM specification and a state assignment, *fitness* of this specific assignment can be computed. The *fitness* function for this is given by:

$$Fitness = \sum_{i=0}^{s-1} \sum_{j-0}^{s-1} (k + 1 - D\left(S_i, S_j\right)) DAG_{ij} \quad (5.3)$$

Where k is the number of bits used for the state codification. Eq. (5.3) can be expressed as

$$Fitness = \sum_{i=0}^{s-1} \sum_{j-0}^{s-1} (k+1) DAG_{ij} - \sum_{i=1}^{s-1} \sum_{j-0}^{s-1} D\left(S_i, S_j\right) DAG_{ij} \quad (5.4)$$

Since DAG_{ij} is fixed for a given FSM formulation and *k* is a positive constant, the first sum results in a constant term. Thus fitness is maximum when the second term is minimum. i.e., when sum over *i* and j of the product $D\ (S_i, Sj)\ DAG_{ij}$ is minimum.

5.4.1 Chromosomal Representation

The list of states is represented as chromosomes. The length of the chromosomes is equal to the number of states used for the sequential machines. As mentioned earlier, the number of bits *s* required to represent the states is equal to the smallest integer which is greater than or equal to $|\log_2 n|$. If there are 3 states, then number of bits required is 2. With these two bits, there can be four possible states. i.e., 0, 1, 2, 3. Here, the states (chromosomes) are represented as decimal integers. An initial population of integers is generated randomly without any duplication of integers in every chromosome. The function "randperm" in MATLAB 2012a is used to generate a shuffled decimal number assignment without any duplication of numbers. The individuals / chromosomes are selected for crossover by any selection procedure as mentioned in Section 1.4. In this work, RWS technique has been used.

5.4.2 Crossover

In (Amaral *et al*. 1995), an individual was represented by a binary matrix with *s* rows and *k* columns where s is the number of states in the FSM and k is the number of bits used in the SSC. Crossover was done by randomly selecting columns from the parents in order to create the offspring. If *k* bits are used to represent the states, let *l* bits (columns) be preserved from parent 1 and the remaining *k-l* bits from parent 2. Here, 2^{k-l} assignments can be identical in these *l* columns which are not permitted.

It can be explained by considering an example shown in Table 5.1 with 6 states. If 2 bits are preserved from parent1 ($k = 3$, $l = 2$) then, at most $2^{k-l} = 2^1 = 2$ combinations can be identical in the first two columns and the conflict has to be eliminated by selecting the appropriate bit from the two different combinations (1 or 0) for the third column. It can be observed that the offsprings S_2 and S_5 have the same attribution which is invalid. Hence the offsprings have to be checked for conflicts if any and proper modifications have to be made which makes the job cumbersome.

Table 5.1 Example for illustrating the conflict during conventional crossover

States	Parent1	Parent 2	Offspring
S_0	001	000	000
S_1	000	011	001
S_2	010	100	**010**
S_3	101	010	100
S_4	110	001	111
S_5	011	110	**010**

The checks for conflicts if any, and further corrections can be eliminated in the proposed modified crossover technique where the chance for getting the states invalid is eliminated.

Hence the states have to be checked for duplication, which consumes more computational time. To avoid duplication, a modified crossover technique is proposed which leads to modified GA.

Consider two parents with five states selected for crossover as shown in Fig. 5.3. Before crossover, GA takes a copy of the parents and stores it in a temporary variable and a "Similarity Test" is performed. In this test, the first element of parent B is taken and compared with all elements of parent A. If that element is present in parent A, then the second element of parent B is compared with all elements of parent A. If there are no similar elements in parent A, the corresponding decimal values of the same row are interchanged. This process continues for all the elements in parent B. Now the parent A in the temporary variable becomes offspring C. The process is reversed by taking each element of parent A and comparing it with all elements of parent B in order to produce offspring D (Fig. 5.4). Thus the off springs are protected from duplication.

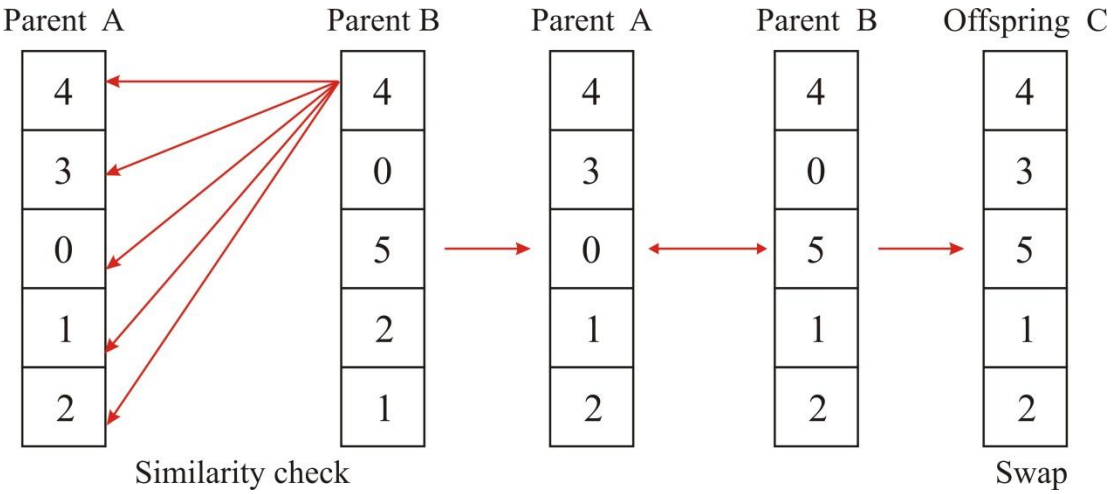


Fig. 5.3 Swapping of individuals by comparing the elements of parent B with the elements of parent A

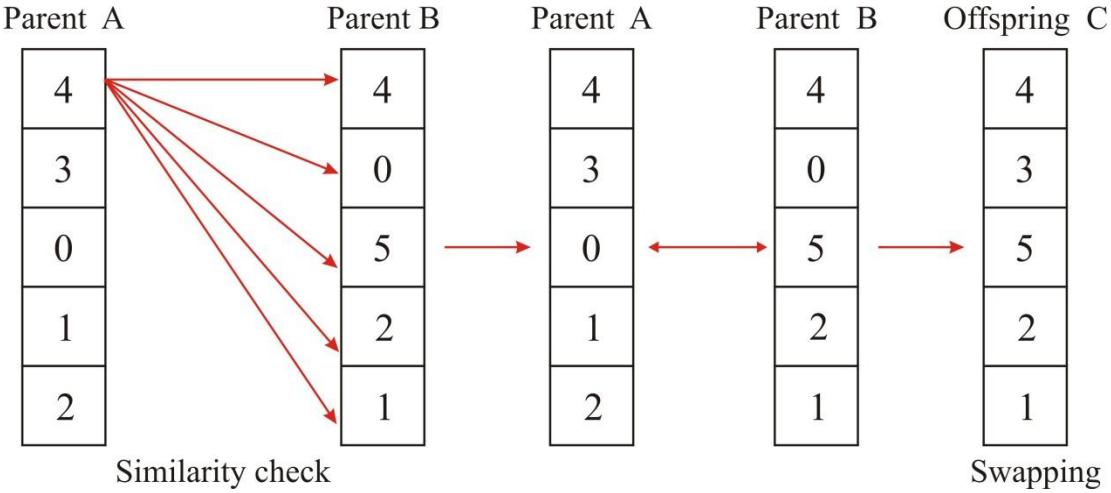
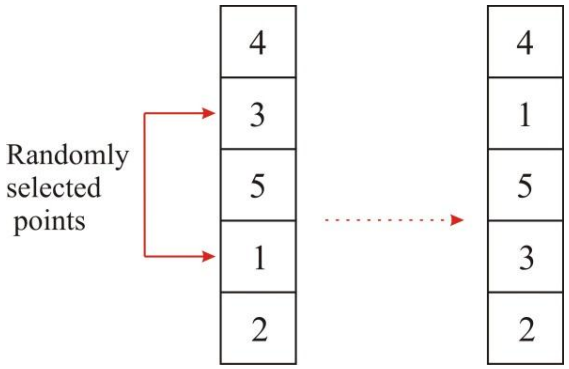


Fig. 5.4 Swapping of individuals by comparing the elements of parent A with the elements of parent B

In the conventional mutation technique, any of the gene in a randomly selected individual is changed. Then there is a chance for two of the states to have the same assignment which is not permissible. Hence the conventional mutation process cannot be applied in this case. To create randomness in the offsprings, instead of changing a selected gene, the genes at two randomly selected positions of the selected offspring are interchanged. For E.g.; consider the offspring shown in Fig. 5.5 (a). If the two randomly selected positions of the individual are, 2^{nd} and 4^{th}, then the corresponding states are interchanged as in fig. 5.5 (b) so as to get the new offspring.



a) Offspring before interchange of genes b) Offspring after interchange of genes

Fig. 5.5 Exchange of genes

The chromosome having the maximum fitness value is considered to be the OSA. Once OSA is obtained from GA, the STT is prepared. Using this STT, the corresponding excitation tables are generated and the circuits are evolved automatically. The combinational circuits have been realised using i) gates alone and ii) using 2-1 mux / 2-1 RM ULM. T / D flip flops are used as state registers.

5.5. DESIGN OF THE COMBINATIONAL PART

For the design of the combinational part using gates, the circuits are evolved using a new GA based technique as mentioned in Section 3.3. For faster implementation of GA, the chromosomes are represented in a bidirectional array (2D) and the suitable crossover and mutation techniques have been applied. With this technique, the CLCs for Moore and Mealy machines have been evolved.

The exclusive use of 2-1mux / 2-1 RM ULM for the implementation of combinational part reduces the manufacturing cost due to the repeated use of the same design element. The circuits based on ULMs were evolved using the VIM proposed in Section 4.3.

5.6 RESULTS

Sequence detectors and modulo-*n* counters have been evolved, the combinational part of which has been realised using gates and ULMs.

5.6.1 Implementation of Mealy Machines

Sequence detectors are the best examples for Mealy machines. “011” sequence detector and “1010” sequence detector circuits have been evolved using the proposed techniques.

Example 1. **“011” Sequence Detector** - the circuit produces a logic high output whenever the sequence “011” is detected in the input stream.

The state transition graph of “011” sequence detector is shown in Fig. 5.6. Since there are only 3 states, number of state variables required is the smallest integer greater than or equal to $s = |\log_2 3|$, which is equal to 2. With two state variables, there can be four possible states 0, 1, 2, 3. GA evolves the OSA using 3 states out of the available 4 states based on the fitness function mentioned in Section 5.4.

With the proposed GA technique, the OSA obtained is 0, 2, 1. With this assignment, STT is constructed as shown in Table 5.2. Using this STT, the necessary excitation table for the state register using T flip flop is generated as shown in Table 5.3 and the system automatically evolves the combinational circuit to obtain the inputs for next stage as well as the output of FSM.

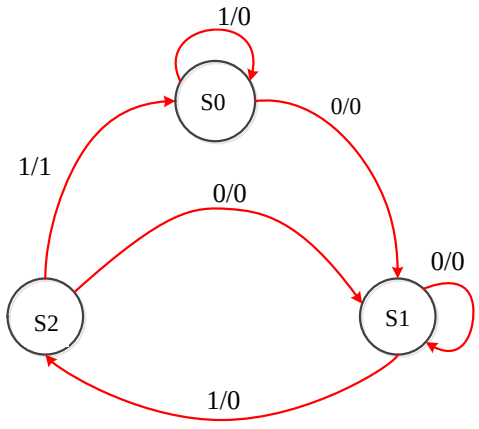


Fig. 5.6 State Transition Graph of “011” sequence detector

Table 5.2 State Transition Table for “011” sequence detector

Present state	**Next State**		**Output**	
	X=0	X=1	X=0	X=1
00	10	00	0	0
10	10	01	0	0
01	10	00	0	1

Table 5.3 Excitation table for “011” sequence detector

Present state			**Next state**		**Output**	**Desired inputs**	
Q_a	Q_b	X	Q_a^*	Q_b^*	Y	T_a	T_b
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	0	1	0	0	1	1
0	1	1	0	0	1	0	1
1	0	0	1	0	0	0	0
1	0	1	0	1	0	1	1
1	1	0	0	0	0	1	1
1	1	1	0	0	0	1	1

Q_a*, Q_b*- states of flip flops A and B after the application of clock pulse.

The combinational part of the circuit is realised with gates using GA. 2D Crossover and mutation techniques have been used (method proposed in Section 3.3) for the implementation which reduces the computation time significantly. The circuit was evolved using 8 gates as shown in Fig. 5.7. The average number of generations required was 98 with a population size of 200 individuals.

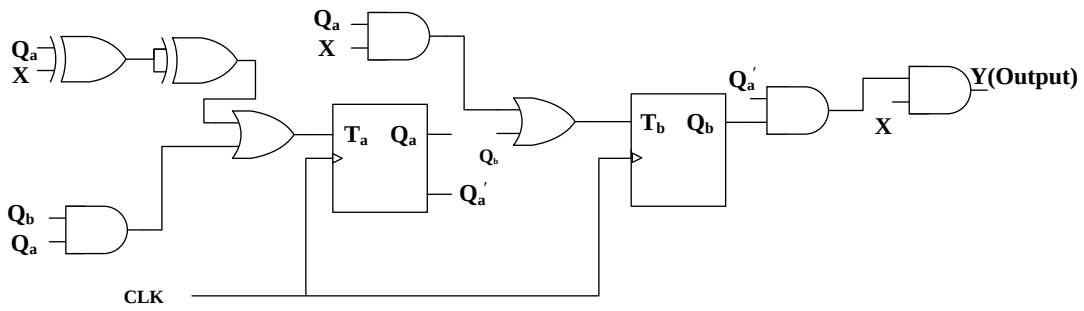


Fig. 5.7 Circuit generated for “011” sequence detector using gates

The design of “011” sequence detector using 2-1 mux is shown in Fig. 5.8. Here, the circuits for T_a, T_b, and Y are evolved separately. To optimise the generated circuits, the algorithm checks for redundancies and are eliminated so that the generated circuits are optimal. The inputs to the state registers as well as the output of the FSM are implemented using 7 binary multiplexers.

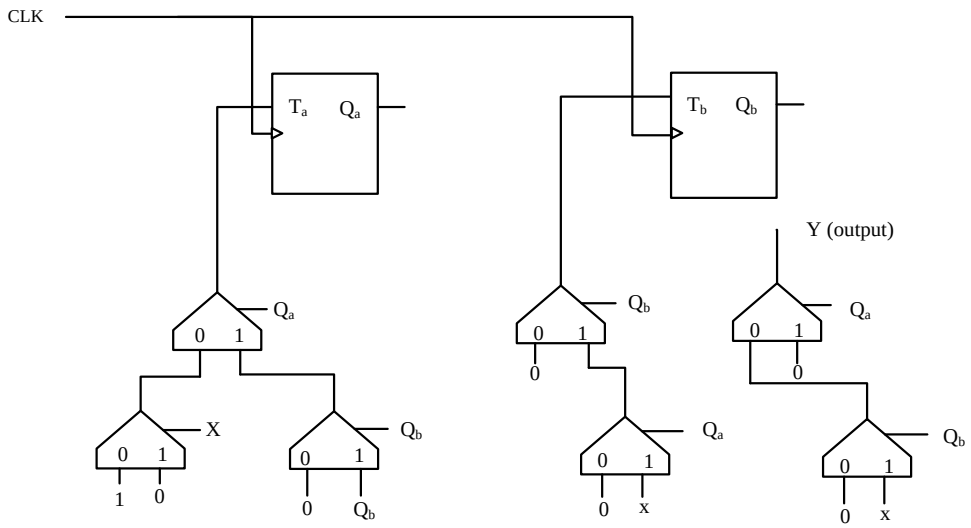


Fig. 5.8 Implementation of “011” sequence detector using 2-1 mux

The generated circuit for “011” sequence detector using RM ULM is shown in Fig. 5.9. It uses 6 binary Reed Muller blocks to realise the FSM.

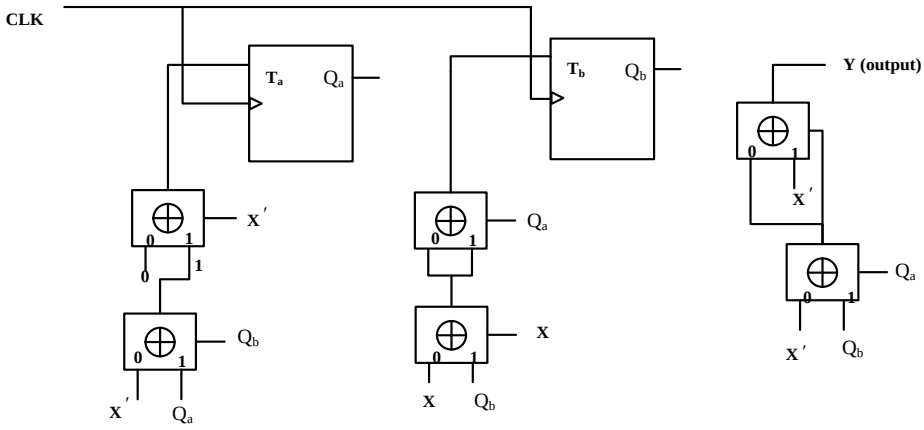


Fig. 5.9 Circuit evolved for “011” sequence detector using RM ULM

Example 2. **“1010” sequence detector** (overlapping) - the circuit raises the output to logic ‘1’ whenever the sequence “1010” is detected in the input stream.

This is an example taken from Ali *et al*. (2004), the state diagram for which is shown in Fig. 5.10. The OSA evolved is (0, 3, 1, 2). With this assignment, STT is prepared as shown in Table 5.4. The combinational circuits at the input of state register as well as the combinational circuit for the output are evolved as per the excitation table shown in Table 5.5.

The CLCs have been evolved using i) gates ii) mux and iii) RM ULM and the corresponding circuits are shown in Figs. 5.11, 5.12 and 5.13 respectively.

It can be observed from Fig. 5.11 that the CLC needs only 4 gates for generating the next state and output, compared to 5 gates in the literature (Ali *et al*. 2004). Here D flip flops have been used as state registers so as to have a comparison with the method in the above literature.

From Figs. 5.11 and 5.12, it is seen that 4 mux and 4 RM units respectively are needed for the implementation of this FSM using the ULMs.

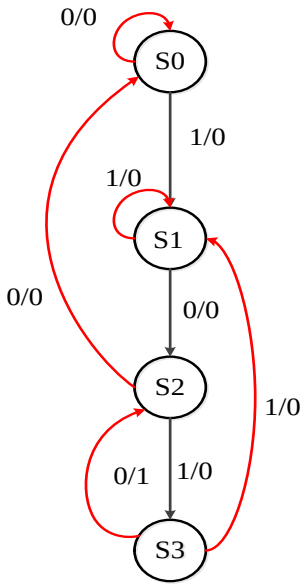


Fig. 5.10 State Transition Graph of "1010" sequence detector

Table 5.4 State Table for "1010" sequence detector

Present state	**Next State**		**Output**	
	X=0	X=1	X=0	X=1
00	00	11	0	0
11	01	11	0	0
01	00	10	0	0
10	01	11	1	0

Table 5.5 Excitation table for "1010" sequence detector for the state assignment 0, 3, 1, 2

Present state			**Next state**		**Output**	**Desired input**	
Q_a	Q_b	X	Q_a^*	Q_b^*	Y	D_a	D_b
0	0	0	0	0	0	0	0
0	0	1	1	1	0	1	1
0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0
1	0	0	0	1	1	0	1
1	0	1	1	1	0	1	1
1	1	0	0	1	0	0	1
1	1	1	1	1	0	1	1

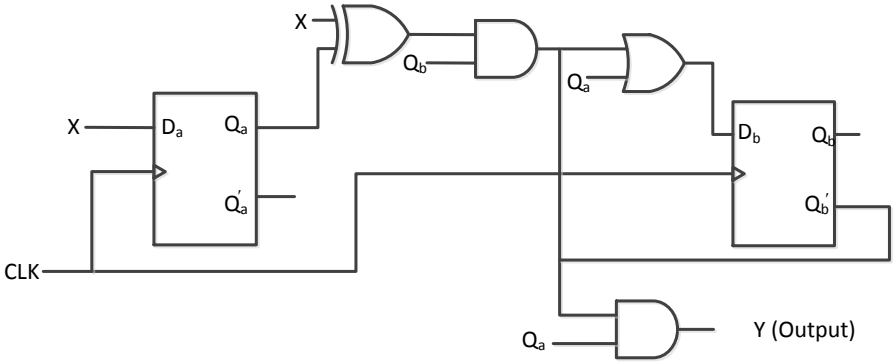


Fig. 5.11 Circuit for "1010" sequence detector using gates

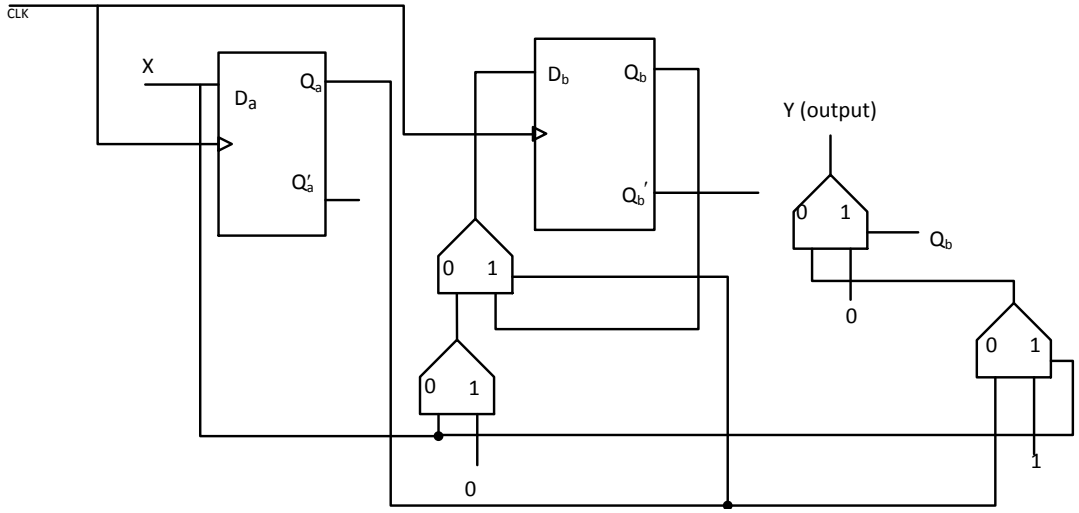


Fig. 5.12 Circuit for "1010" sequence detector using mux

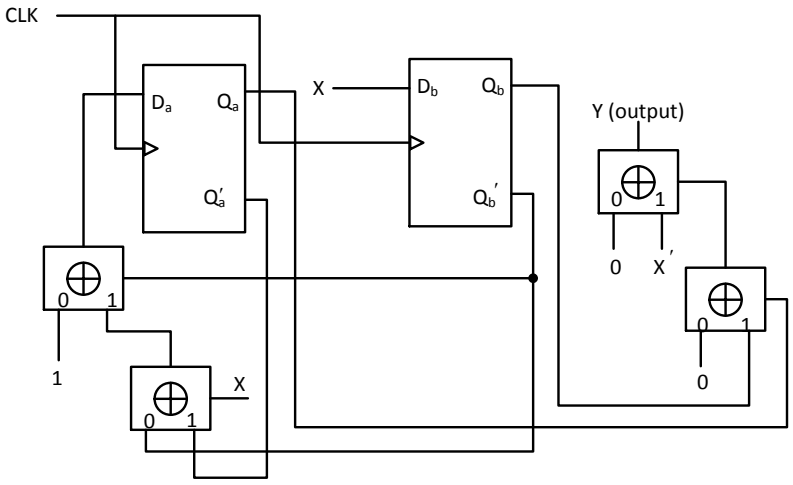


Fig. 5.13 Circuit for "1010" sequence detector using RM ULM

From the results, it is obvious that the proposed technique evolves circuits with hardware which is better or as good as the results available in the literature.

5.6.2 Implementation of Moore machines

Counters such as mod 3, mod 6, and mod 8 have been generated as examples of Moore machines. This can be extended to any number of bits and for any counting pattern.

Example 1. **Mod 3 counter**

The excitation table corresponding to Mod 3 counter with the sequence 0, 1, 2 is shown in Table 5.6 and the CLC to generate the next state are evolved using this excitation table. The inputs to the flip flops T_a and T_b are functions of Q_a and Q_b and the circuit generated using gates is shown in Fig. 5.14. It can be seen that the CLC to generate the next state requires 2 gates.

Table 5.6 Excitation table for mod 3 counter

Present state		**Next state**		**Desired inputs**	
Q_a	Q_b	Q_a*	Q_b*	T_a	T_b
0	0	0	1	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	1	0	0	1	1

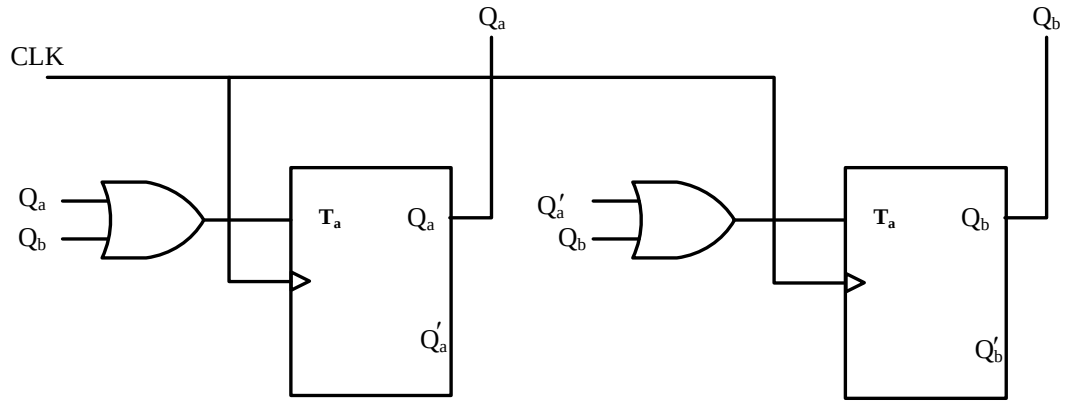


Fig. 5.14 Generated circuit for mod 3 counter using gates

The FSM using T flip flops and mux or RM ULM are shown in Fig. 5.15 and 5.16 respectively. It can be observed that the circuits require 2 ULMs in both the cases.

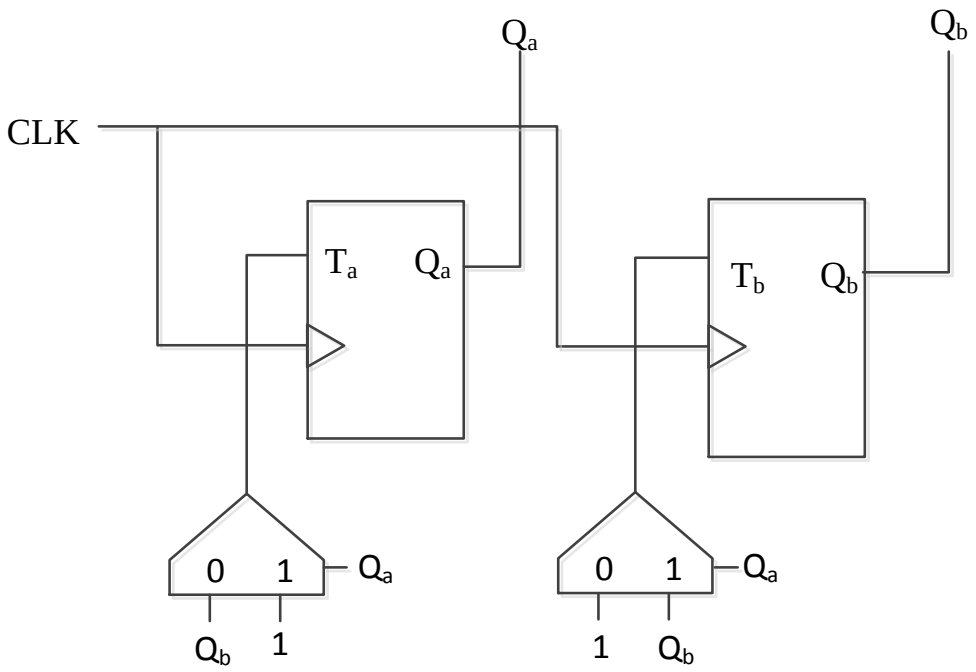


Fig. 5.15 Circuit for mod 3 Counter using mux

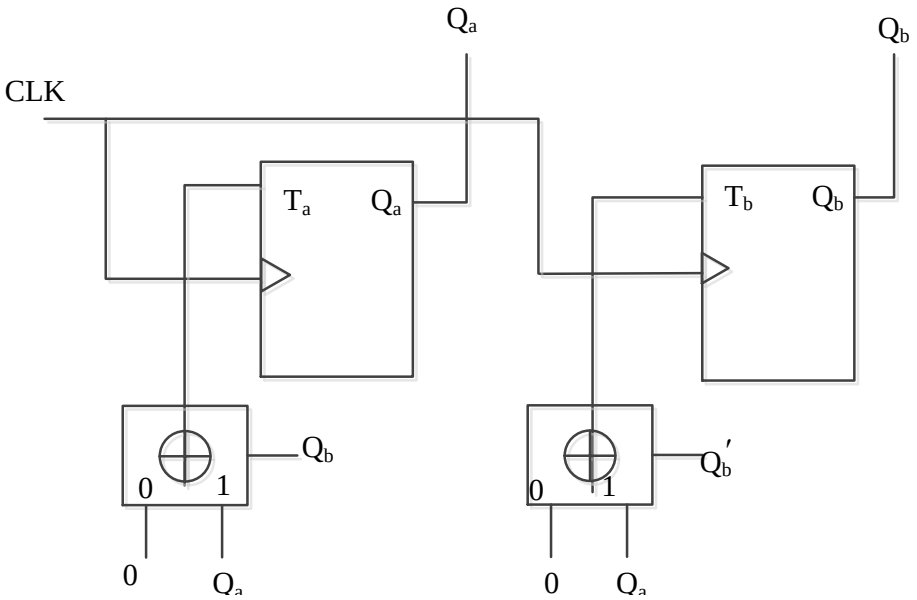


Fig. 5.16 mod 3 Counter using RM ULM

Example 2. **mod 6 counter**

Table 5.7 shows the excitation table corresponding to Mod 6 counter with the counting sequence 0, 1, 2, 3, 4, 5. The corresponding CLCs using gates and the ULMs are shown in Figs. 5.17, 5.18 and 5.19 respectively.

Table 5.7 Excitation table for mod 6 counter

Present State			**Next state**			**Desired inputs**		
Q_a	Q_b	Q_c	Q_a*	Q_b*	Q_c*	T_a	T_b	T_c
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	0	0	0	1	0	1
1	1	0	1	0	0	0	1	0
1	1	1	1	0	0	0	1	1

From Fig. 5.17, it can be seen that the circuit is evolved using 7 gates. Since 2 D crossover technique has been adopted, the computational time is reduced. From Figs. 5.18 and 5.19, it can be seen that the circuit using mux needs 6 units whereas the circuit with RM ULM needs 8 units.

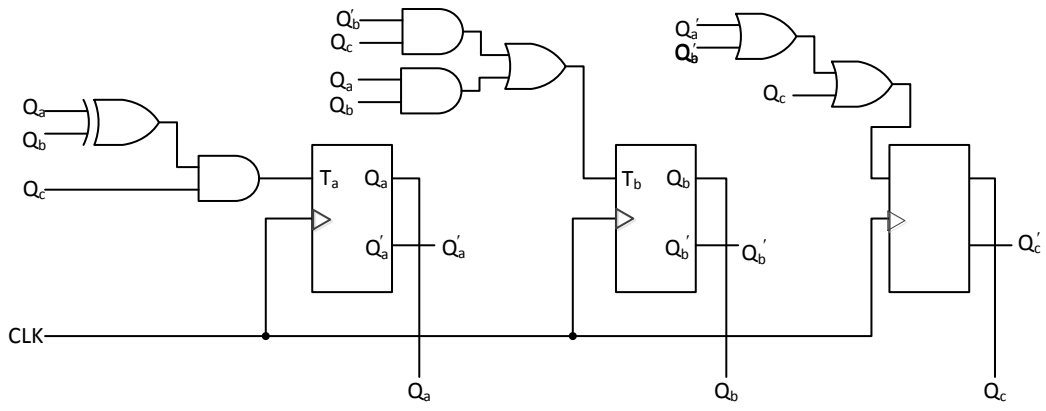


Fig. 5.17 mod 6 Counter using gates

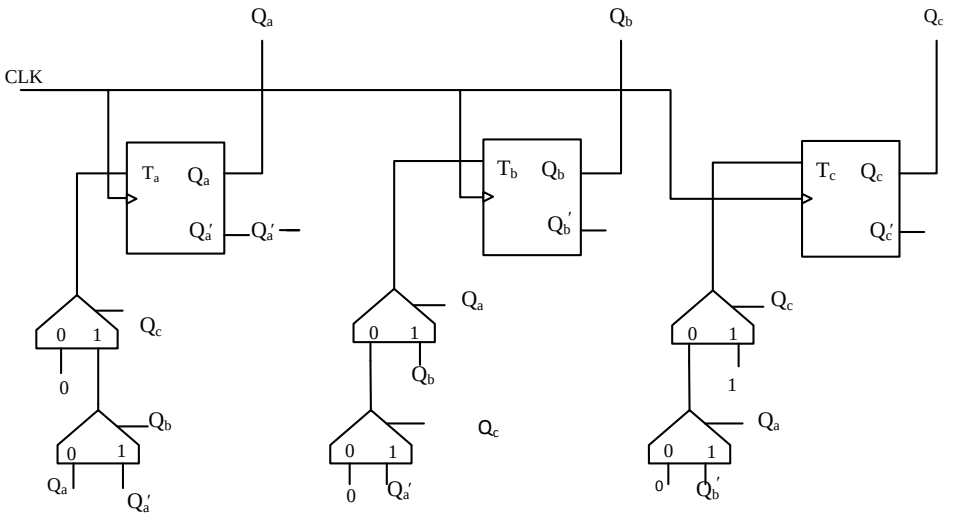


Fig. 5.18 mod 6 Counter using mux

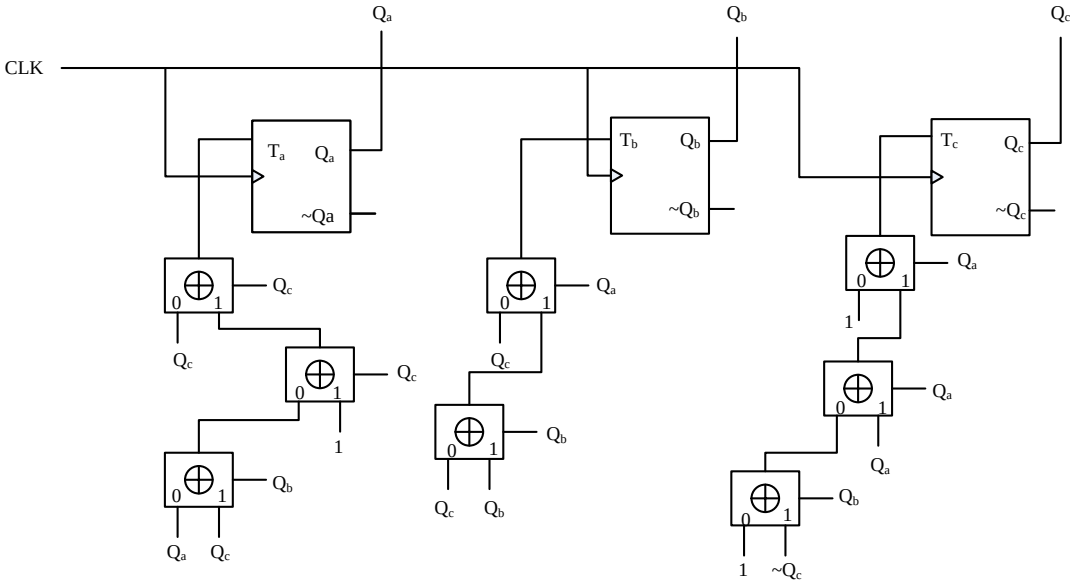


Fig. 5.19 Mod 6 Counter using RM ULM

Example 3. **Mod 8 Counter**

The excitation table for mod 8 counter is shown in Table 5.8 and the corresponding circuits using gates, mux and RM ULM are shown in Figs. 5.20, 5.21 and 5.22 respectively. T flip flop has been used for the implementation.

Table 5.8 Excitation table for mod 8 counter

Present state			**Next state**			**Desired inputs**		
Q_a	Q_b	Q_c	Q_a*	Q_b*	Q_c*	T_a	T_b	T_c
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

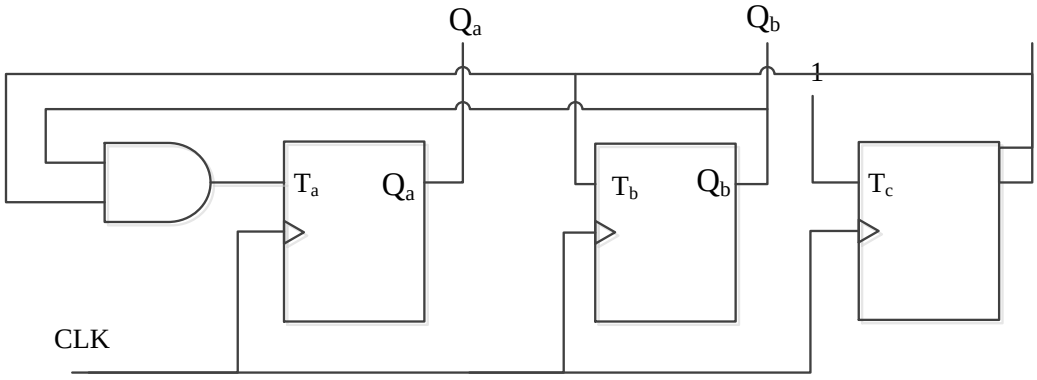


Fig. 5.20 Circuit for mod 8 counter using gates

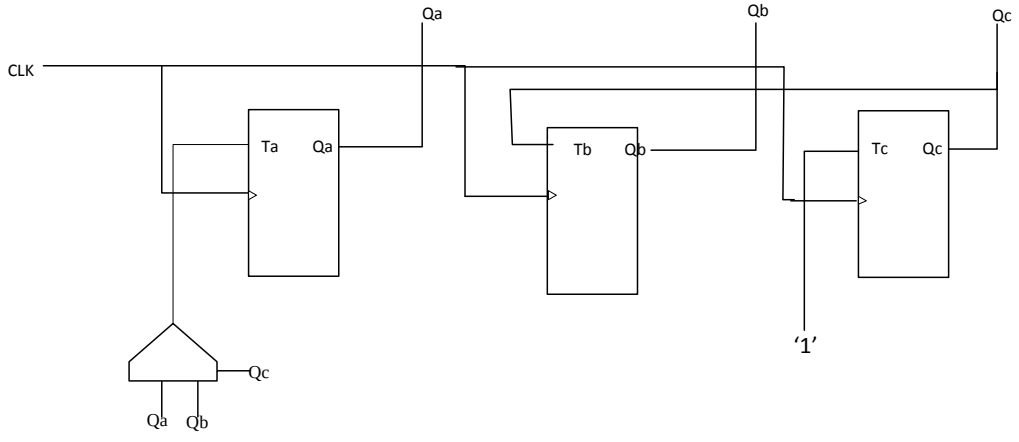


Fig. 5.21 Circuit for mod 8 counter using mux

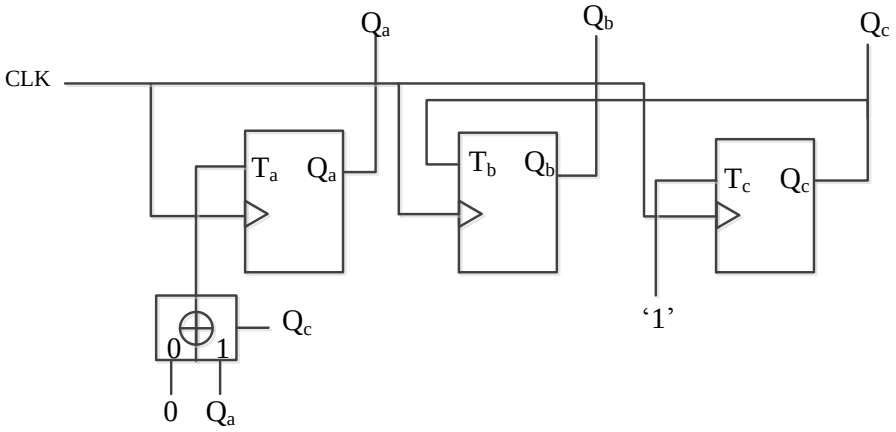


Fig. 5.22 Circuit for mod 8 counter using RM ULM

5.7 SUMMARY

This chapter explores the possibility of using GA for the design automation of SSCs. Conventional GA is not suitable for the state assignment in sequential circuits. Hence a modified GA has been proposed to obtain the OSA. The OSA determines the complexity of the combinational circuits required to generate states as well as the outputs. The CLCs are generated using i) gates and ii) ULMs such as 2-1 mux / 2-1 RM ULM as mentioned in chapters 3 and 4. The circuits based on ULMs were evolved using the VIM proposed in this thesis as this method produced more optimal circuits.

Sequence detectors and modulo *n* counters were used to validate the proposed techniques. The OSA evolved in this work produced circuits better than those in the available literature.

For the CLCs evolved using gates, a new technique proposed in this thesis helped to reduce the computational time involved. The circuits with the exclusive use of ULMs have the advantage that repeated use of a single design element reduces the manufacturing cost.

CHAPTER 6

CONCLUSION AND SCOPE FOR FURTHER WORK

6.1 THESIS SUMMARY AND CONCLUSION

In conventional design, the quality of the designed circuit depends solely on the designer's capability and it varies from designer to designer. Evolutionary design as an alternative method for logic design has become more attractive because of its algebraindependent techniques for generating efficient circuits. The work presented in this thesis concentrates on developing automated techniques for the design of digital circuits using GA. Synthesis of both combinational and sequential logic circuits have been done.

A CLC can be realised by using an interconnection of logic gates, or ULMs such as multiplexers or RM logic modules. The ability to realise logic functions using 2 input ULMs is of growing importance in the design of CLCs and hence only binary ULMs are used in this work. The functions have been implemented using i) gates alone and ii) 2-1 mux / 2-1 RM ULM. The synthesis of SSCs is done with D / T flip flops and the combinational part is being automated using GA with gates / ULMs as design elements. This automated approach for the generation of circuits has the added advantage of reduced dependency on the designer's knowledge and experience.

In this thesis, a faster 2D technique using GA has been proposed for the design of CLCs using gates. A new 2D chromosomal representation and its suitable crossover and mutation techniques are developed. With this technique, since the chromosomes are represented as matrices, crossover operation involves swapping of sub matrices

instead of a portion of a linear array in the case of the existing method of crossover. Hence, in linear crossover some of the levels / gates may remain unaltered during crossover, whereas with the proposed approach, since the sub matrices are swapped for crossover operation, variations from the parent circuits can occur at any level either in connections / type of gates. Furthermore, 2D chromosomal representation offers better visualisation of the circuit. The quality of the evolved circuits is determined by the strength of the fitness function used. XOR, AND, OR gates and WIRE have been used for the design. The fitness function has been formulated in such a way that 100 % functional circuits are evolved with minimum hardware by assigning an additional fitness value for every WIRE used. The computational time involved in this technique has been compared with that of linear crossover and mutation and is found to be reduced considerably. Moreover, on comparing the number of units / levels with the conventional method, automated design generates circuits with lesser number of gates / levels. Benchmark functions have been used to validate the results.

For the design of CLCs using binary ULMs, two new techniques referred to as Constant Input Method (CIM) and Variable Input Method (VIM) are proposed. These methods are based on the exclusive use of Universal Logic blocks such as 2-1 mux / 2-1 RM ULM so as to produce fully functional circuits with minimum number of units, interconnections and levels. With SI technique, using Shannon's / Davio decomposition method, a function of *n* variables can be realised using 2^n-1 binary ULMs in *n* levels. In this thesis, an attempt has been made to reduce the number of units and / levels by suggesting certain modifications on these techniques.

The two techniques differ only in the inputs to the circuit. In CIM, the inputs to the circuit are only 0s and 1s whereas with VIM, the inputs can be 0, 1, variables or their complements. The control signals are selected at random from among the variables, their complements or functions derived from the immediate preceding level, whereas SI used fixed control signals for each level. Moreover, unlike SI technique, control signal selected at one level can be used for other levels too and all the variables need not be used as control signal in the proposed techniques. For CIM, only 0s and 1s are given as inputs so that no variable inputs are needed which is an added advantage.

The proposed techniques have been validated using benchmark functions. The evolved circuits are synthesised using Xilinx ISE 14.2 on Spartan 3 device (XC3S400). Functions up to 6 variables have been implemented and detailed analysis of the evolved circuits for number of units / levels is made. The evolved circuits have been compared with SI and with the methods available in the literature. It was observed that circuits obtained by both the methods are more efficient than the conventional methods. In both mux and RM implementations, VIM produced circuits with minimum hardware / delay. For XOR based circuits, minimum hardware / delay were obtained for VIM using RM ULM.

Design of Synchronous Sequential Circuits (SSCs) which involve memory/storage elements is also investigated in this thesis. Every SSC can be defined as a Finite State Machine and its design involves two stages. First stage involves the OSA which solely determines the complexity of the combinational part of SSCs. A modified GA has been proposed to obtain the OSA with a view to minimise the hardware. Second stage involves the design of combinational part to generate the next state and the output of the state machine.

The combinational part of SSC has been evolved using i) gates and ii) the binary ULMs such as 2-1 mux and 2-1 RM ULM. For the CLCs using gates, the technique proposed in this thesis helped to reduce the computational time involved. The circuits with the exclusive use of ULMs have the advantage that repeated use of a single design element reduces the cost of implementation. Moreover, the circuits evolved by the techniques presented in this thesis for the design using ULMs are more efficient compared to the conventional methods in terms of hardware / delay. Here, VIM mentioned in Section 4.3 has been used for the realisation of the combinational part. A few Moore and Mealy machines have been implemented using these techniques. D / T flip flops are used as state registers and XOR, AND, OR gates or WIRE are used for the gate level design of the combinational part.

It has been observed that the OSA evolved in this work produced circuits better than or at least as comparable to those in the available literature.

6.2 SCOPE FOR FURTHER WORK

The present work used only four types of gates for the realisation of circuits. As a future expansion, more number of gates could be used so that still better circuits could be evolved. This thesis dealt with completely specified functions. Future enhancement is to extend the work to incompletely specified functions. In this work, circuits are evolved with the exclusive use of a single ULM. But it could be extended to design the circuits using a combination of mux and RM ULM. The methods presented in this work based on ULMs used functions derived from the immediate preceding level as control signals to the next level. One possible future extension is to use the functions derived from the lowest level onwards to be transferred to any level.

The number of units can still be reduced by eliminating the units having the same outputs in any of the levels. So an important future direction is to eliminate such modules. In this work, design of CLCs using ULMs is limited to single output functions and as a future work, it can be extended for multiple output functions. Another possible extension for this work is applying 2D crossover and mutation techniques for the design of CLCs using ULMs.

In this thesis, design automation of SSCs has been made and it could be extended to the design of ASCs. As a future scope, GA can be applied effectively by modifying the fitness functions to obtain optimal circuits by inexact computing. Other biologically inspired algorithms may also be tried for the evolution of circuits.

REFERENCES

1. **Aguirre, A.H., C.A.C. Coello, and B.P. Buckles** (1999) A genetic programming approach to logic function synthesis by means of multiplexers. Proceedings of *First NASA/DoD Workshop on Evolvable Hardware,* California, USA, July, 46-53. IEEE Computer Society Press.

2. **Aguirre, A.H., B.P. Buckles, and C.A.C. Coello** (2000). Gate-level synthesis of Boolean functions using binary multiplexers and genetic programming. Proceedings of *Congress on Evolutionary Computation,* California, USA, July, 675-682. IEEE.

3. **Aguirre, A.H., B.P. Buckles, and C.A.C. Coello** (2002) Circuit Design Using Genetic Programming: An Illustrative Study. Proceedings of *10th NASA Symposium on VLSI Design,* New Mexico, March, 4.1.1-4.1.9. IEEE.

4. **Aguirre, A.H. and C.A.C. Coello** (2004) Using genetic programming and multiplexers for the synthesis of logic circuits. *Engineering Optimization*, **36(4)**, 491-511.

5. **Ahmad, I. and M.K. Dhodhi** (2000) State assignment of finite-state machines. *IEE Proceedings - Computers and Digital Techniques*, **147(1)**, 15-22.

6. **Akers, S.B.** (1978) Binary decision diagrams. *IEEE Transactions on computers*, **100(6)**, 509-516.

7. **Al Jassani, B.A., N. Urquhart, and A.E.A. Almaini** (2010) Manipulation and optimization techniques for Boolean logic. *IET Computers & Digital Techniques*, **4(3)**, 227-239.

8. **Al Jassani, B.A., N. Urquhart, and A.E.A. Almaini** (2011) State assignment for sequential circuits using multi-objective genetic algorithm. *IET Computers & Digital Techniques*, **5(4)**, 296-305.

9. **Ali, B.** (2003) *Evolutionary algorithms for synthesis and optimization of sequential logic circuits. PhD Thesis,* Napier University Edinburgh.

10. **Ali, B., A.E.A. Almaini, and T. Kalganova** (2004) Evolutionary algorithms and their use in the design of sequential logic circuits. *Genetic Programming and Evolvable Machines*, **5(1)**, 11-29.

11. **Almaini, A.E.A., J.F. Miller, and L. Xu** (1992) Automated synthesis of digital multiplexer networks. *IEE Proceedings E-Computers and Digital Techniques*, **139(4)**, 329-334.

12. **Almaini, A.E.A., J.F. Miller, P. Thomson, and S. Billina** (1995*a*) State assignment of finite state machines using a genetic algorithm. *IEE Proceedings-Computers and Digital Techniques*, **142(4)**, 279-286.

13. **Almaini, A.E.A., N. Zhuang, and F. Bourset** (1995*b*) Minimization of multi-output Reed-Muller binary decision diagrams using hybrid genetic algorithm. *Electronics Letters*, **31(20)**, 1722-1723.

14. **Almaini, A.E.A. and N. Zhuang** (1995) Using genetic algorithms for the variable ordering of Reed-Muller binary decision diagrams. *Microelectronics Journal*, **26(5)**, 471-480.

15. **Almaini, A.E.A. and N. Zhuang** (1997). Variable ordering of BDDs for multi-output boolean functions using evolutionary techniques. Proceedings of *Fourth IEEE-ICECS97, pp. 1239-1244.*

16. **Amaral, J.N., K. Tumer, and J. Ghosh** (1995) Designing genetic algorithms for the state assignment problem. *IEEE Transactions on Systems, Man, and Cybernetics*, ***25*(4)**, 687-694.

17. **Bryant, R.E.** (1985) Symbolic manipulation of boolean functions using a graphical representation. Proceedings of *22nd ACM/IEEE Design Automation Conference*, 688-694. IEEE Press.

18. **Chaudhury, S. and S. Chattopadhyay** (2008) Fixed polarity Reed-Muller network synthesis and its application in AND-OR/XOR-based circuit realization with area-power trade-off. *IETE Journal of Research*, **54(5)**, 353-363.

19. **Coello, C.A., A.D. Christiansen, and A.H. Aguirre** (1996) Using genetic algorithms to design combinational logic circuits. *Intelligent Engineering through Artificial Neural Networks*, **6(0)**, 391-396.

20. **Coello, C.A.C., A.D. Christiansen, and A.H. Aguirre** (2000*a*) Towards automated evolutionary design of combinational circuits. *Computers & Electrical Engineering*, **27(1)**, 1-28.

21. **Coello, C.A.C., A.D. Christiansen, and A.H. Aguirre** (2000*b*) Use of evolutionary techniques to automate the design of combinational circuits. *International Journal of Smart Engineering System Design*, **2**, 299-314.

22. **Coello, C.A.C., A.H. Aguirre, and B.P. Buckles** (2000*c*) Evolutionary multi-objective design of combinational logic circuits. Proceedings of *Second NASA/DoD Workshop on Evolvable Hardware*, California, USA,161-170. IEEE Computer Society.

23. **Coello, C.A.C., R.L.Z. Gutiérrez, B.M. García, and A.H. Aguirre** (2002). Automated design of combinational logic circuits using the ant system. *Engineering Optimization*, **34(2)**, 109-127.

24. **Comer, D. J.** (1995*) Digital logic and state machine design*, Oxford University Press, 1995

25. **Czerwiński, R. and D. Kania** (2010) Synthesis method of high speed finite state machines. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, **58(4)**, 635-644.

26. **De Micheli, G., R.K. Brayton, and A. Sangiovanni-Vincentelli** (1985) Optimal state assignment for finite state machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **4(3)**, 269-285.

27. **Digalakis, J.G. and K.G. Margaritis** (2001) On benchmarking functions for genetic algorithms. *International journal of computer mathematics*, **77(4)**, 481-506.

28. **Drechsler, R., H. Hengster, H. Schäfer, J. Hartmann, and B. Becker** (1999) Testability of 2-level AND/EXOR circuits. *Journal of Electronic Testing*, **14(3)**, 219-225.

29. **Ercegovac, M.D. and T. Lang** *Digital systems and hardware/firmware algorithms*. John Wiley & Sons, 1985.

30. **Falkowski, B.J. and C.H. Chang** (2000) Minimization of k-variable-mixed-polarity Reed-Muller expansions. *VLSI Design*, **11(4)**, 311-320.

31. **Faraj, K.** (2009*a*) Fast coding for dual Reed-Muller expressions. Proceedings of *6th WSEAS International conference on Engineering education*, World Scientific and Engineering Academy and Society, 212-219.

32. **Faraj, K.** (2009*b*) Synthesis of multi-level dual Reed-Muller expressions. Proceedings of *1st WSEAS international conference on Nanotechnology, Cambridge*, UK. World Scientific and Engineering Academy and Society, 47-51.

33. **Golberg, D.E.** *Genetic algorithms in search, optimization, and machine learning*, Addison Wesley, 1989.

34. **Gorai, R.K. and A. Pal** (1990) Automated synthesis of combinational circuits by cascade networks of multiplexers. IEE Proceedings *E-Computers and Digital Techniques*, **137(2)**, 164-170.

35. **Haddow, P.C. and A.M. Tyrrell** (2011) Challenges of evolvable hardware: Past, present and the path to a promising future. *Genetic Programming and Evolvable Machines*, **12(3)**, 183-215.

36. **Hartmanis, J.** (1961) On the state assignment problem for sequential machines. I. *IRE Transactions on Electronic computers*, **(2)**, 157-165.

37. **Higuchi, T., M. Murakawa, M. Iwata, I. Kajitani, W. Liu, and M. Salami** (1997) Evolvable hardware at function level. Proceedings of IEEE 4th *International Conference on Evolutionary Computation (CEC97)*, New Jersey, USA, April, 187-192. IEEE.

38. **Hounsell, B.I. and T. Arslan** (2000) A novel genetic algorithm for the automated design of performance driven digital circuits. Proceedings of *Congress on Evolutionary Computation*, California, USA, July, 601-608. IEEE.

39. **James, R.K., T.K. Shahana, K.P. Jacob, and S. Sasi** (2006) Delay-Reduced Combinational Logic Synthesis using Multiplexers. ESA, 105-110.

40. **Kalganova, T. and J. Miller** (1999) Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. Proceedings of *First NASA/DoD Workshop on Evolvable Hardware*, Los Alamitos, California, 54-63. IEEE Computer Society Press.

41. **Kalganova, T.G.** *Evolvable hardware design of combinational logic circuits*, *PhD Thesis*, Napier University Edinburgh, Scotland, 2000.

42. **Kalganova, T.** (2000) An extrinsic function-level evolvable hardware approach. Proceedings of *European Conference on Genetic Programming (EuroGP2000)*, Edinburg, UK, 60-75. Springer Berlin Heidelberg.

43. **Karakatic, S., V. Podgorelec, and M. Hericko** (2013) Optimization of combinational logic circuits with genetic programming. *Elektronika ir Elektrotechnika*, *19*(7), 86-89.

44. **Koza, J.R.** *Genetic programming III: Darwinian invention and problem solving* (Vol. 3). Morgan Kaufmann, 1999.

45. **Li, N.S., J.D. Huang, and H.J. Huang** (2008) Low power multiplexer tree design using dynamic propagation path control. Proceedings of *IEEE Asia Pacific Conference on Circuits and Systems APCCS208*, Nov-Dec, 838-841. IEEE.

46. **Liu, R., S.Y. Zeng, L. Ding, L. Kang, H. Li, Y. Chen, and Y. Han** (2006) An efficient multi-objective evolutionary algorithm for combinational circuit design. Proceedings of *First NASA/ESA Conference on Adaptive Hardware and Systems (AHS'06)*, June, 215-221. IEEE.

47. **Louis, S.J.** *Genetic algorithms as a computational tool for design*. *PhD thesis*, Department of Computer Science, Indiana University, Bloomington, USA, 1993.

48. **Mano, M.M.** *Digital design*, EBSCO, 2002.

49. **McCluskey, E.J. and S.H. Unger** (1959) A note on the number of internal variable assignments for sequential switching circuits. *IRE Transactions on Electronic Computers*, **(4)**, 439-440.

50. **Miller, J.F., P. Thomson, and T. Fogarty** (1997) Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: A case study. *Genetic algorithms and evolution strategies in engineering and computer science*, *8*.

51. **Miller, J.F. and P. Thomson** (1998) Discovering novel digital circuits using evolutionary techniques. In IEE Colloquium on *Evolvable Hardware Systems (Digest No. 1998/233)*, IET.

52. **Miller, J.F.** (1999) Evolution of digital filters using a gate array model. In Workshops on *Applications of Evolutionary Computation* (pp. 17-30). Springer Berlin Heidelberg.

53. **Miller, J.F., D. Job, and V.K. Vassilev** (2000*a*) Principles in the evolutionary design of digital circuits—Part I. *Genetic programming and evolvable machines*, **1(1-2)**, 7-35.

54. **Miller, J.F., D. Job, and V.K. Vassilev** (2000*b*) Principles in the evolutionary design of digital circuits—Part II. *Genetic programming and evolvable machines*, **1(3)**, 259-288.

55. **Murakawa, M., S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi** (1996) Hardware evolution at function level. Proceedings of *International Conference on Parallel Problem Solving from Nature (PPSN1996)*, 62-72. Springer Berlin Heidelberg.

56. **Oh, P. and A.E.A Almaini** (2007) 2 variable Reed Muller binary decision diagrams. Proceedings of *6th WSEAS International Conference on Electronics, Hardware, Wireless and Optical Communications*, Wisconsin, USA, 197-202. World Scientific and Engineering Academy and Society.

57. **Pal, A.** (1986) An algorithm for optimal logic design using multiplexers. *IEEE transactions on computers*, **100(8)**, 755-757.

58. **Pradhan, S.N. and S. Chattopadhyay** (2008) Two-level AND-XOR networks synthesis with area-power trade-off. *International Journal of Computer Science and Network Security*, **8(9)**, 365-375.

59. **Reis, C., J.A.T. Machado, and J.B. Cunha** (2004) Evolutionary design of combinational logic circuits. *JACIII*, **8(5)**, 507-513.

60. **Reis, C. and J.A.T. Machado** (2007) Computational intelligence in circuit synthesis. *JACIII*, **11(9)**, 1122-1127.

61. **Sagar, K. and S. Vathsal** (2013) Design of Combinational Circuits Using Evolutionary Techniques, *International Journal of Science and Modern Engineering*, **1(9)**, 47-51.

62. **Sarrafzadeh, M. and C.K. Wong** *An introduction to VLSI physical design*. McGraw-Hill Higher Education, 1996.

63. **Šeda, M.** (2008) Heuristic Set-Covering-Based Post-processing for Improving the Quine-McCluskey Method. *International Journal of Computational Intelligence*, *4*(2), 139-143.

64. **Sekanina, L.** (2009) Evolvable hardware: From applications to implications for the theory of computation. Proceedings of *International Conference on Unconventional Computation,* Portugal, 24-36. Springer Berlin Heidelberg.

65. **Shahana, T.K., R.K. James, K.P. Jacob, and S. Sasi** (2005) Automated synthesis of delay-reduced Reed-Muller universal logic module networks. In *2005 NORCHIP,* 90-93. IEEE.

66. **Shanthi, A.P., B. Vijayan, M. Rajendran, S. Veluswami, and R. Parthasarathi** (2002) Automatic GA Based Evolution of Fault Tolerant Digital Circuits. Proceedings of *4th Asia–Pacific Conference on Simulated Evolution and Learning (SEAL02),* Singapore, 845-849.

67. **Slowik, A. and M. Bialko** (2008) Evolutionary design of combinational digital circuits: State of the art, main problems, and future trends. Proceedings of *1st International Conference on Information Technology (IT2008),* May, 1-6. IEEE.

68. **Soleimani, P., R. Sabbaghi-Nadooshan, S. Mirzakuchaki, and M. Bagheri** (2011*a*) Using genetic algorithm in the evolutionary design of sequential logic circuits. *arXiv preprint arXiv:1110.1038*.

69. **Soleimani, P., S. Mirzakuchaki, K. Mohammadi, and M. Bagheri** (2011*b*) A novel evolutionary design of sequential logic circuits by using genetic algorithm. *International Journal of Modeling and Optimization*, **1(3)**, 231.

70. **Soliman, A.T. and H.M. Abbas** (2003) Combinational circuit design using evolutionary algorithms. Proceedings of *Canadian Conference on Electrical and Computer Engineering (IEEE CCECE 2003),* May, **1**, 251-254. IEEE.

71. **Soliman, A.T. and H.M. Abbas** (2004). Synchronous sequential circuits design using evolutionary algorithms. Proceedings of *Canadian Conference on Electrical and Computer Engineering,* May, 4, *2013-2016*. IEEE.

72. **Stearns, R.E. and J. Hartmanis** (1961) On the state assignment problem for sequential machines II. *IRE Transactions on Electronic Computers*, **(4)**, 593-603.

73. **Stomeo, E., T. Kalganova, C. Lambert, N. Lipnitsakya, and Y. Yatskevich** (2005) On evolution of relatively large combinational logic circuits. Proceedings of *NASA/DoD Conference on Evolvable Hardware (EH'05),* July, 59-66. IEEE.

74. **Story, J.R., H.J. Harrison, and E.A. Reinhard** (1972) Optimum state assignment for synchronous sequential circuits. *IEEE Transactions on Computers,* **100(12)**, 1365-1373.

75. **Tao, Y., J. Cao, Y. Zhang, J. Lin, and M. Li** (2012) Using module-level evolvable hardware approach in design of sequential logic circuits. Proceedings of *IEEE Congress on Evolutionary Computation*, June, 1-8. IEEE.

76. **Tao, Y., Y. Zhang, J. Cao, and Y. Huang** (2013) A module-level three-stage approach to the evolutionary design of sequential logic circuits. *Genetic Programming and Evolvable Machines,* **14(2)**, 191-219.

77. **Thompson, A.** (1996) An evolved circuit, intrinsic in silicon, entwined with physics. Proceedings of *International Conference on Evolvable Systems, 390-405*. Springer Berlin Heidelberg.

78. **Thomson, P. and J.F. Miller** (1996) Symbolic method for simplifying AND-EXOR representations of Boolean functions using a binary-decision technique and a genetic algorithm. In *IEE Proceedings-Computers and Digital Techniques*, **143(2)**, 151-155.

79. **Umbarkar, A.J., M.S. Joshi, and P.D. Sheth** (2015) Dual population genetic algorithm for solving constrained optimization problems. *International Journal of Intelligent Systems and Applications,* **7(2)**, 34.

80. **Vassilev, V.K. and J.E. Miller** (2000) Scalability problems of digital circuit evolution evolvability and efficient designs. Proceedings of *the Second NASA/DoD Workshop on Evolvable Hardware,* July, 55-64. IEEE.

81. **Vijayakumari, C.K. and P. Mythili** (2012) A faster 2D technique for the design of combinational digital circuits using Genetic Algorithm. In proceedings of *International Conference on Power, Signals, Controls and Computation (EPSCICON2012), Thrissur, India,* Jan, 1-5. IEEE.

82. **Vijayakumari, C.K., P. Mythili, R.K. James, and S.A. Kumar** (2014) Optimal design of combinational logic circuits using genetic algorithm and Reed-Muller Universal Logic Modules. Proceedings of *International Conference on Embedded Systems (ICES2014),* July, 1-6. IEEE.

83. **Vijayakumari, C.K., P. Mythili, and R.K. James** (2015*a*) A simplified efficient technique for the design of combinational logic circuits. *International Journal of Intelligent Systems and Applications,* **7(9)**, 42-48.

84. **Vijayakumari, C.K., P. Mythili, R.K. James, and S.A. Kumar** (2015*b*) Genetic algorithm based design of combinational logic circuits using universal logic modules. *Procedia Computer Science*, **46**, 1246-1253.

85. **Wang, L.** *Automated synthesis and optimization of multilevel logic circuits, Doctoral dissertation,* Napier University Edinburgh, UK, 2000.

86. **Xia, Y., B. Ali, and A.E.A Almaini** (2003*a*) Area and power optimization of FPRM function based circuits. Proceedings of *International Symposium on Circuits and Systems (ISCAS2003),* Bangkok Thailand, May, 5, 329. IEEE.

87. **Xia, Y., X. Wu, and A.E.A. Almaini** (2003*b*) Power minimization of FPRM functions based on polarity conversion. *Journal of Computer Science and Technology*, **18(3)**, 325-331.

88. **Yan, X., Q. Wu, C. Hu, and Q. Liang** (2011) Electronic circuits optimization design based on cultural algorithms. *Journal of Information Processing and Management*, **2(1)**, 49-56.

89. **Yanagiya, M.** (1995) Efficient genetic programming based on binary decision diagrams. Proceedings of *IEEE International Conference on Evolutionary Computation*, Nov-Dec, 1, 234. IEEE.

90. **Yau, S.S. and C.K. Tang** (1970) Universal logic modules and their applications. *IEEE Transactions on Computers*, **100(2)**, 141-149.

LIST OF PAPERS

SUBMITTED ON THE BASIS OF THIS THESIS

I REFEREED JOURNALS

1. **Vijayakumari, C.K., P. Mythili, and R.K. James** (2015). A Simplified Efficient Technique for the Design of Combinational Logic Circuits. *International Journal of Intelligent Systems and Applications*, **7(9)**, 42-48.

2. **Vijayakumari, C. K., R. K. James & P. Mythili** (2016). A GA Based Simple and Efficient Technique to Design Combinational Logic Circuits Using Universal Logic Modules. *Journal of Circuits, Systems and Computers*, 25(**07**), 1650074(1-22).

II PRESENTATIONS IN CONFERENCES

1. **Vijayakumari, C. K. and P. Mythili** (2012). A faster 2D technique for the design of combinational digital circuits using Genetic Algorithm. In proceedings of *International Conference on Power, Signals, Controls and Computation (EPSCICON2012), Thrissur, India*, Jan, 1-5. IEEE.

2. **Vijayakumari, C. K., D. Lukose, P. Mythili, and R.K. James** (2013). An improved design of combinational digital circuits with multiplexers using genetic algorithm. In *Emerging Research Areas and 2013 International Conference on Microelectronics, Communications and Renewable Energy (AICERA/ICMiCR), 2013 Annual International Conference on*, June, 1-5. IEEE.

3. **Vijayakumari C. K, P. Mythili, R. K. James** (2013). A modified technique for the optimal design of combinational digital circuits with multiplexers using genetic algorithm**,** In IET Proc. of *International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, 225 – 229.

4. **Vijayakumari, C.K., P. Mythili, R.K. James, and S.A. Kumar** (2014). Optimal design of combinational logic circuits using genetic algorithm and Reed-Muller Universal Logic Modules. Proceedings of *International Conference on Embedded Systems (ICES2014)*, July, 1-6. IEEE.

5. **Vijayakumari, C.K., P. Mythili, R.K. James, and S.A. Kumar** (2015*a*). Genetic algorithm based design of combinational logic circuits using universal logic modules. *Procedia Computer Science*, **46**, 1246-1253.

6. **Vijayakumari, C. K., Mythili, P., & K. J. Rekha** (2015*b*). Genetic Algorithm Based Design of Combinational Logic Circuits using Reed Muller blocks. In *Proceedings of the World Congress on Engineering* . **1**, 978-988.

CURRICULUM VITAE

1. **NAME** : Vijayakumari C. K

2. **DATE OF BIRTH** : 05 may 1963

3. **EDUCATIONAL QUALIFICATIONS**

1985 Bachelor of Technology (B. Tech)

Institution	:	T. K. M. College of Engineering, Kollam, Kerala
Specialization	:	Electrical Engineering
Class / Division obtained	:	First Class with Distinction

1987 Master of Technology (M. Tech)

Institution	:	College of Engineering, Thiruvanathapuram, Kerala
Specialization	:	Electrical Machines
Class / Division obtained	:	First Class with Distinction

Doctor of Philosophy (Ph. D)

Institution	:	Cochin University of Science and Technology, Kerala
Registration Date	:	20-09-2008