DIGITAL CIRCUIT DESIGN THROUGH SIMULATED EVOLUTION

by

UTHMAN SALEM AL-SAIARI

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

In Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

Dhahran, Saudi Arabia

November 2003

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by

UTHMAN SALEM AL-SAIARI

under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER ENGINEERING

Thesis Committee

Prof. Sadiq M. Sait (Chairman)

Prof. Mostafa Abd – El – Barr (Co – Chairman)

Dr. Mohammed Farouk Khan (Member)

Prof. Sadiq M. Sait
Department Chairman

Prof. Osama A. Jannadi
Dean of Graduate Studies

Date

To my father, God's blessings be upon his grave

To my mother for her prayers all nights
To my dear wife for her love and patience.

Acknowledgements

First of all, all sincere praises and thanks are due to our God, Allah (SWT), who created us and guided us to this great religion, i.e., the Islam. No one can thank God as he deserves because of his limitless blessings on us. May God's peace and blessings be upon his prophet Mohammad who said[1] : *The person who does not thank people, does not thank God as well.*

Thanks to my advisor, Dr. Sadiq M. Sait, who was very helpful during all the phases of this work. This thesis is never possible without the support, encouragement and the valuable time spent by my advisor in bringing this work into existence. Also, thanks to Dr. Mostafa Abd-El-Barr, my co-advisor, for his help during the course of this thesis. Thanks are also due to our great department, Computer Engineering, and our great university, KFUPM, for the valuable support. Also, I would like to express my deepest thanks to every instructor who contributed in building my knowledge and experience. Thanks are also due to my colleagues, especially Mr. Sarif Bambang, who did not hesitate in providing me with any kind of help I needed. I would also to acknowledge the help of Dr. Carlos A. Coello for his valuable discussion during the course of this thesis. Also, I want to thank Syed Sanaullah for his help in the arrangement and finalizing the copying and printing of this thesis.

[1]This is only a translation of the meaning of what he said. He originally said so in Arabic. May peace and blessings be upon him.

I also thank my great parents who provided and supported me all the way. Thanks to them and may God bless them. Last but not least, thanks to my great dear wife who supported me with love and patience. She was really very patient during my work and she always inspired me to finish my work as quickly as possible. Thanks to her very much. Finally, thanks to everybody who contributed to this achievement in a direct or an indirect way.

Contents

List of Tables

List of Figures

THESIS ABSTRACT

Name: Uthman Salem Muhammed Al-Saiari

Title: DIGITAL CIRCUIT DESIGN THROUGH SIMULATED EVOLUTION

Major Field: COMPUTER ENGINEERING

Date of Degree: November 2003

Evolutionary computation presents a new paradigm shift in hardware design and synthesis. The new paradigm is expected to radically change the design procedure such that new possibilities for discovering novel designs and/or more efficient circuits can emerge. In this thesis, Simulated Evolution algorithm is used for combinational logic design. SimE algorithm consists of three steps: evaluation, selection and allocation. Two goodness measures are designed to guide the selection and allocation operations of SimE. Area, power and delay are considered in the optimization of circuits. The performance of the proposed algorithm is evaluated using selected ISCAS'85 benchmark and randomly generated circuits. The results obtained are compared to other techniques.

Keywords: *Evolutionary Circuit Design, Logic Synthesis, Simulated Evolution, Combinatorial Optimization, Multiobjective Optimization, Fuzzy Logic.*

MASTER OF SCIENCE DEGREE

King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia

November 2003

ملخص الرسالة

الاسم : عثمان سالم الصيعري

عنوان الدراسة: تصميم الدوائر الرقميه من خلال التّطوّر المحاكى

التخصص : هندسة الحاسب الآلي

تاريخ التخرج : نوفمبر ٢٠٠٣

يقدّم الحساب الارتقائيّ تحوّل جذريّ جديدًا في تصميم الدوائر الرقميه. التحول الجذري الجديد سيغير طريقة التصميم للدوائر الكهربائيه مما سيؤدى الى اكتشاف تصاميم فريده واكثر عمليه. في هذه الرساله, خوارزم التطور المحاكى استخدمت في تصميم الدوائر الكهربائيه. تتكون خوارزم التطور المحاكى من ثلاث خطوات وهي التقييم, الاختيار, والترتيب. تم اقتراح مقياسين للجوده لمساعده الخوارزم في عملية الاختيار والترتيب. المساحة, الطاقه المستهلكه والوقت اخذت في عين الاعتبار في عمليه تحسين تصميم الدوائر الكهربائيه. اختبرت الخوارزم المقترحه بأستخدام مجموعه من الدوائر الكهربائيه القياسيه (ISCAS'85) ودوائر كهربائيه اختيرت بطريقة عشوائيّة. كما تم مقارنة النتائج مع الطرق الاخرى المتوجده.

مصطلحات البحث: التصميم الارتقائيّ للدوائر, تركيب المنطق, التطور المحاكى, المنطق المبهم, المشاكل المتعدّدة الأهداف

درجة الماجستير في العلوم

الجامعة الملك فهد للبترول و معادن

ظهران - المملكة العربية السعودية

نوفمبر ٢٠٠٣

Chapter 1

Introduction

Logic circuit for digital systems may be combinational or sequential. A combinational circuit consists of logic gates whose outputs at any time are determined by combining the values of the applied inputs using logic operations. A combinational circuit performs an operation that can be specified logically by a set of Boolean expressions. In contrast, sequential circuits employ elements that store bit values. Sequential circuit outputs are function of the inputs and the bit values in the storage elements. These values, in turn, are a function of previously applied inputs and stored values. As a consequence, the outputs of a sequential circuit depend not only on the presently applied values of the inputs, but also on past inputs, and the behavior of the circuit must be specified by a sequence in time of inputs and internal stored bit values. This thesis deals with combinational logic circuits [35].

A combinational circuit consists of input variables, output variables, logic gates,

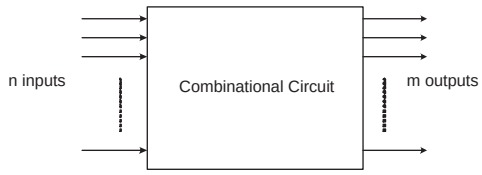


Figure 1.1: Block diagram of a combinational circuit.

and interconnections. The interconnected logic gates accept signals from the inputs and generate signals at the outputs. A block diagram of a combinational circuit is shown in Figure 1.1. The n input variables come from the environment of the circuit, and the m output variables are available for use by the environment. Each input and output variable exists physically as a binary signal that represents logic 1 or logic 0 [11].

For n input variables, there are 2^n possible binary input combinations. For each binary combination of the input variables, there is one possible binary value on each output. Thus, a combinational circuit can be specified by a truth table that lists the output values for each combination of the input variables. A combinational circuit can also be described as a function of the n input variables.

1.1 Logic Design

Design of digital circuits requires knowledge of large collections of domain-specific rules. The process of implementing a digital circuit in hardware involves transforming the original logical specification into a form suitable for the target technology,

optimizing the representation with respect to a number of user defined constraints (i.e., timing, fan-in/out, power, etc.), and finally carrying out technology mapping onto the target technology [28].

In designing a complex system, circuit designers usually have to tradeoff one design objective for another. For example, often a designer tries to find a possibly faster circuit compared to a given previously designed one. However, the number of gates used and power dissipation are strongly related to delay. Thus, in seeking a faster circuit, one may end up having a complex system or a system that has higher power dissipation. Logic synthesis attempts to provide an answer to this problem. The purpose of logic synthesis tools is to aid circuit designers in reaching an optimal tradeoff. Several logic synthesis algorithms are found in the literature [3, 4, 5, 6, 36].

Circuit designers use logic synthesis tools to create digital systems of arbitrary complexity. By using a top-down approach they tend to work in a space of lower dimensionality in which they are experts. However, this method of working is somewhat constrained both by the training and experience of the designer and by the amount of domain-specific knowledge known to the designer. On the other hand, *iterative heuristics* may allow designers to define the search space of circuit design in a way that is natural to both the problem and the implementation. These heuristics have a tendency to search for a solution to the circuit design problem in a much larger, and often richer, design space beyond the realms of the traditional hardware design space. Heuristics can thus help explore the search space regions needed to

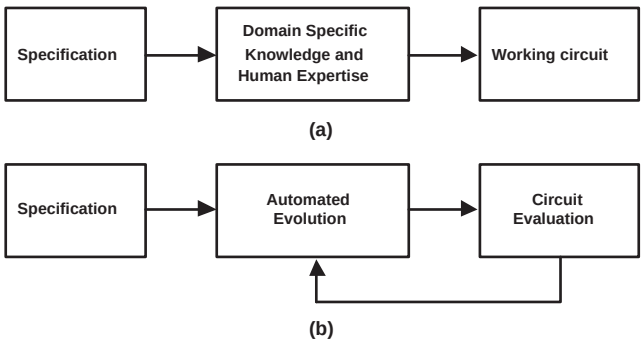


Figure 1.2: Conventional and evolutionary design methodology.

reach designs that are beyond the scope of conventional methods. It may therefore be possible to use iterative heuristics to obtain novel designs that are difficult to discover by conventional heuristics. Figure 1.2 shows the difference between the conventional and the evolutionary methods for circuit design.

Furthermore, evolutionary design approaches do not assume a prior knowledge of any particular design domain. They can be used in domains where little knowledge is available or where such knowledge is costly to obtain. It is often possible to evolve hardware that is too complex in its structure and/or dynamics for human to design.

In conventional logic design, circuit designers begin with a precise specification in the form of truth tables or Boolean expressions. These expressions are manipulated by applying logic synthesis algorithms. The outcome of the logic synthesis algorithms will always be in the space of all logically correct representations as shown in Figure 1.3. On the contrary, the evolutionary algorithms work on a larger space that may not represent the desired function, but gradually pulls the specification of the circuit towards the target function.

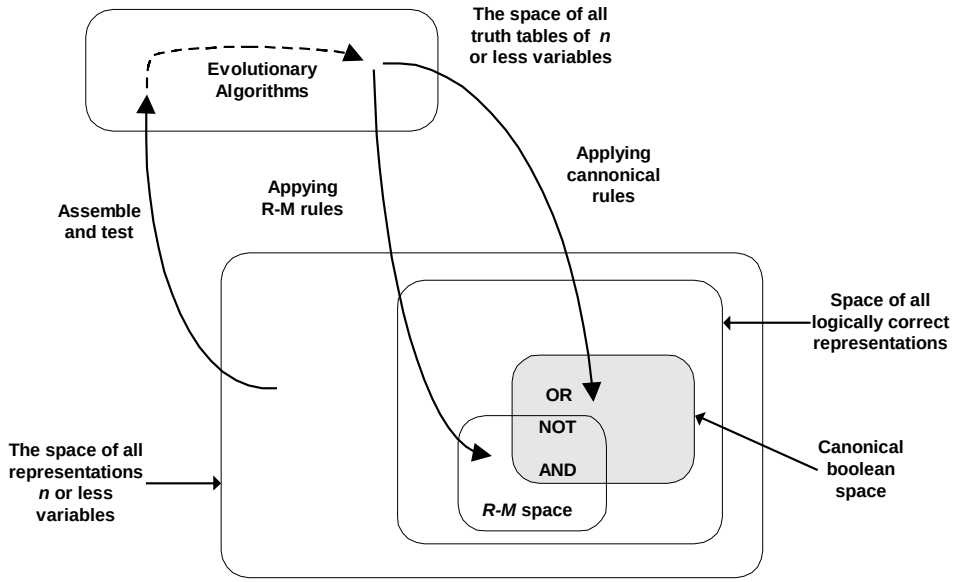


Figure 1.3: Evolutionary design process [28].

It is well-known that many optimization problems arising in computer science, engineering, management, administration or other fields cannot be solved exactly within reasonable time limits, at least not for problem sizes of practical interest. For this reason, heuristics have been introduced to find high quality solutions for these problems in reasonable time. Some heuristics are not restricted to specific problem types, but may be applied, with suitable modifications, to a broad class of optimization problems.

Heuristic algorithms, when properly exploited, will quickly enable the development of *acceptable* solutions. A heuristic algorithm will only search inside a subspace of the total search space for a *good* rather than the best solution which satisfies design constraints. Therefore, the time requirement of a heuristic is small compared

to that of full enumerative algorithms.

It is possible to classify heuristics into *iterative heuristics* and *constructive heuristic*. In constructive heuristics, a solution is generated from scratch by successive addition of certain elements or components, with or without backtracking (that is, removal of components that have been added at an earlier step). An example of constructive heuristics is the *greedy heuristic* [34]. On the other hand, iterative heuristics start with a complete feasible solution and refine this solution in order to improve the objective function value. One of the well known iterative heuristics is the *local search* algorithm. All modern iterative heuristics such as *simulated annealing*, *tabu search*, or *genetic algorithms*, are generalization of the local search heuristic. Readers are refereed to [34] for a detailed description of these algorithms.

1.2 Problem Definition

The problem under consideration can be defined as follows:

> *Given the truth table of a function f and a target technology to work within, design a combinational logic circuit that performs the function f subject to constraints using Simulated Evolution Algorithm (SimE).*

This problem is referred to as *Evolutionary Logic Design through Simulated Evolution (SimE) algorithm.*

1.3 Related Work

Evolutionary logic design has received increasing attention in the last decade. Motivated by de Garris's idea [10] back in 1993, Higuchi et al., [19] obtained an evolved circuit to solve the 6-multiplexer problem [24]. Later, in 1995, Thompson [37] managed to evolve a tone discriminator circuit in XC6200 FPGA. Thompson used the 1800-bits FPGA's configuration string as the *genotype* representation of the circuit. Thompson showed that Evolutionary Algorithms (EAs) can explore much richer design space [38] . Using EAs, he was able to produce circuits that are better than those produced by conventional methods in term of number of gates.

Koza et al. employed genetic programming [23] and pioneered the evolution of analog circuits. Using a SPICE simulator, they automatically generated circuits which are competitive with those obtained using human designs and showed that it is possible to produce designs for quite complex analog circuits, namely: low-distortion op-amp, low-pass filter, and band-pass filters [22]. Genetic Programming is also used as intrinsic evolution of analog circuits [48]. A functional level evolution was proposed by Higuchi [18]. A complete review and taxonomy of the field is described in [14, 45, 32].

In a recent development, much attention is given to the evolutionary design of arithmetic circuits as they provide the essential building blocks needed for larger DSP applications. Such effort has resulted in the development of arithmetic circuits

that range from a simple sequential adder structure to the more complex 3-bit multiplier design. Fogarty and Miller build some arithmetic circuits that cannot be produced by human designer's conventional methods [15, 27]. Both of these used Genetic Algorithms. Miller improved his findings by using Evolutionary Strategies to evolve arithmetic circuits [28]. Coello et al. proposed a similar approach to evolve a circuit using Genetic Algorithm (GA) [7, 8].

1.4 Motivation

Design is usually considered to be an activity requiring considerable human creativity and knowledge. Even the definition of the term *design* itself is quite elusive, since it can be interpreted in several different ways depending on the task to be performed.

The definition of design that fulfills the purposes of our thesis is the process of deriving, from a specified input/output behavior, a structure (in our case a certain combination of logic gates) that is functional (produces all the outputs desired for all the inputs specified) within a certain set of specified constraint and objectives.

Furthermore, we want this design to be optimum in terms of certain structural features (e.g., the number of gates used). The design process is a very tedious and error prone task that usually requires considerable human expertise.

Several techniques in evolutionary design of digital circuits have been studied.

Most of the work being done in evolutionary logic synthesis is just random search where the evolutionary algorithm will blindly evolve the circuit according to a given set of objectives without using rules and techniques of the conventional logic synthesis. It is believed that incorporating logic synthesis rules and guidelines combined with the idea of assemble-and-test could lead to better results. The motivation of our thesis is to develop a computer-based tool that can make the design process less tedious for the human designer without sacrificing quality of the designs produced. In this thesis, we limit our focus to combinational logic circuits, which contains no memory elements and no feedback paths. The use of Simulated Evolution algorithm is considered.

Simulated Evolution (SimE) algorithm showed better result in most of the other optimization problems when compared to other evolutionary algorithm [34]. This is due to the nature of the algorithm and its converging aspects. Also, SimE uses only one solution (chromosome) resulting in less memory requirement. Moreover, SimE algorithm uses *goodness measure* to guide the algorithm in the search space. The more knowledge about the problem incorporated into the *goodness measure*, the better the performance of SimE algorithm in terms of CPU time and quality of solution. Therefore, using the Simulated Evolution algorithm as our evolutionary algorithm should provide a good solution for the required automated synthesis tool.

1.5 Thesis Organization

This thesis is organized as follows. In Chapter 2, some basic background material about conventional logic design (CLD) and logic design optimization is covered. In Chapter 3, evolutionary logic design (ELD) is presented. Detailed discussion on Simulated Evolution Algorithm (SimE) is in Chapter 4. In Chapter 5 and 6, the proposed implementation of the Simulated Evolution Algorithm (SimE) for Logic Design is presented and the proposed goodness measures are explained. Experimental results of the proposed techniques is given in Chapter 7. Comparison with Tabu Search (TS), Genetic Algorithm (GA) [7], Ant Colony Algorithm (ACO) [9] and SIS [36] are given in Chapter 8. Finally, the thesis ends by some conclusions and future directions in Chapter 9.

Chapter 2

Background

This chapter provides some necessary background information. The first section discusses the conventional logic design techniques and definitions. Next, logic design optimization is presented and the difference between optimization for objectives or using constraint optimization is explained. This is followed with a section on multi-objective optimization. Following this, fuzzy logic for multi-objectives optimization is presented.

2.1 Conventional Logic Design (CLD)

The dramatic increase in designer productivity over the past decade in the area of VLSI circuit design is a direct result of the development of sophisticated computer-aided design tools. Logic synthesis techniques speed up the design cycle and reduce

the human effort. Synthesis algorithms work on a model of the circuit, not the circuit itself. Circuit representation is therefore important to understand. Reader should refer to any logic design book such as [26] for more elaborate background material.

Logic synthesis area is usually divided into two-level synthesis (PLA) and multi-level synthesis. Because of the architecture inherent to PLAs, optimization methods focus almost exclusively on minimizing the number of PLA product terms, which in turn minimizes the PLA area. The area of two-level combinational logic minimization has already matured. One can routinely find a minimum or near-minimum sum-of-product form for a logic function. These functions can be multiple output, incompletely specified, and functions with multiple-valued input variables. Functions with hundreds of inputs and outputs are within the realm of the algorithms. The optimization can also be done in a reasonable amount of computing time [5].

The other method for implementing logic is multilevel logic. Because of the increased potential for reusing sublogic, there are more degrees of freedom in the solution representation than in the PLA case. Consequently, it has been much more difficult to synthesize this type of logic using manual synthesis. In the following subsection, a survey of the existing conventional logic synthesis methodologies is given.

2.1.1 Two-level Boolean Functions

Logic function can be represented in a variety of ways. One can use a two-level representation in which literals are combined with a single operator (AND operator) and then these terms are combined with a second operator (OR operator) which is SOP form. When a logic function is expressed in terms of product terms, which involve all input variables and all true output products are present, the expression is referred to as a canonical Boolean expression. Usually the goal of logic synthesis is to represent a logic function in the simplest way by reducing the number of product terms and literals [28].

2.1.2 Multilevel Boolean Functions

The two-level logic provides the minimum logic required to implement any arbitrary Boolean function. However, there exist some Boolean functions that are very inefficiently represented in two-level logic. On the contrary, multilevel logic representation of a logic function allows the use of factoring and decomposition into sub-functions. Consider, for example, the following logic function.

$$F = a \cdot e + a \cdot f + b \cdot c \cdot e + b \cdot c \cdot f + b \cdot d \cdot e + b \cdot d \cdot f$$

The function can also be expressed as follows:

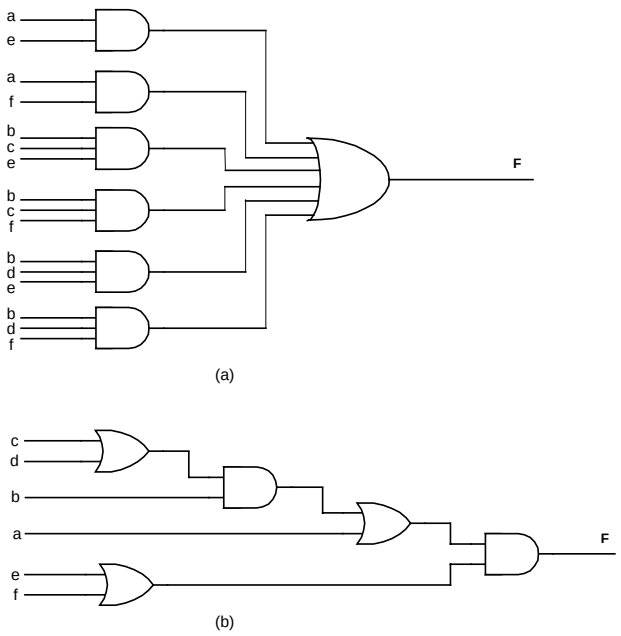


Figure 2.1: Representation of a boolean function in (a) in two-level logic (b) in multi-level logic.

$$F = (a + b \cdot (c + d)) \cdot (e + f)$$

It is easy to notice that the first representation of the function f has 16 literals, while the second has only 6. Thus, the multilevel logic representation will reduce the area requirement of a Boolean function, as shown in Figure 2.1.

The starting point for multilevel minimization is the minimum two-level canonical form. Sophisticated heuristic-based minimization algorithms have been written which try to reduce the literal counts in Boolean multilevel expressions [5, 6, 28].

2.1.3 Reed-Muller and Exclusive-OR Logic

It is well known that many Boolean functions which can be easily implemented using XOR gates are very inefficiently represented in canonical Boolean logic. The most extreme case of this being the n-bit parity functions which can be realized with n - 1 XOR gates but if AND-OR logic is used, it requires $2^{n-1} - 1$ OR gates and a large number of AND gates. When a Boolean logic function is expressed using XOR gates and uncomplemented variables it is called a *Reed-Muller* (RM) canonical form [16]. If any particular variable is allowed to be complemented or uncomplemented throughout the expansion then the representation is known as a *fixed polarity RM form*. Finding a good polarity is a difficult problem and evolutionary algorithm have been used for this purpose. Work has been done on minimizing the less restricted XOR sum-of-products representation [37].

2.2 Logic Design Optimization

Logic circuit optimization is often performed in conjunction with synthesis. Optimization is motivated not only by the desire of maximizing the circuit quality, but also by the fact that synthesis without optimization would yield noncompetitive circuits, and therefore its value would be marginal. In this thesis, the optimization objectives considered are area, power and delay [2, 13, 26]

Circuit area is measured by the sum of the areas of the circuit components and

therefore it can be computed from a structural view of a circuit, once the areas of the components are known. The area computation can be performed hierarchically. Usually, the fundamental components of digital circuits are logic gates and registers, whose area is known *a priori*. The wiring area has a minor effect and can be neglected [25, 26, 31].

However, circuit power and delay are not additive, and therefore computation for power and delay as objectives require analyzing the structure and often the behavior of the circuit. Often simplifying assumptions for the computations of power consumption and timing (delay) are made assuming that all inputs are available at the same time. Also, another simplifying assumption is to use a model for the power and delay measurements. These model can simplify the computation of the power and delay at the expense of accuracy. For timing performance, the delay of the most critical path of the circuit is considered. More details on area, power and delay computations are given later.

2.3 Multi-objective Optimization

Multi-objective optimization (also called multi-criteria optimization, multi-performance or vector optimization) can be defined as the problem of finding a vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent the objective functions. These functions form a mathematical description

of performance criteria which are usually in conflict with each other. Formally, we can state the general multi-objective optimization problem (MOP) as follows:

Definition 1 (General MOP): Find the vector $\vec{x}^* = [x_1^*, x_2^*, ..., x_n^*]^T$ which will satisfy the m inequality constraints:

$$g_i(\vec{x}) \geq 0 \; i = 1, 2, ..., m \tag{2.1}$$

the p equality constraints

$$h_i(\vec{x}) = 0 \; i = 1, 2, ..., p \tag{2.2}$$

and optimizes the vector function

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), ..., f_k(\vec{x})]^T \tag{2.3}$$

where $\vec{x}^* = [x_1^*, x_2^*, ..., x_n^*]^T$ is the vector of decision variables.

In other words, we wish to determine from among the set F of all numbers which satisfy 2.1 and 2.2 the particular set $x_1^*, x_2^*, ..., x_n^*$ which yields the optimum values of all the k objectives of the problem [26].

The idea of using multi-objective optimization techniques to handle constraints is not new. Some researchers have proposed to redefine the single-objective optimization of $f(\vec{x})$ as a multi-objective optimization problem in which we will have $m + 1$ objectives, where m is the number of constraints. Then, we can apply any multi-objective optimization technique to the new vector $\vec{v} = (f(\vec{x}), f_1(\vec{x}), ..., f_m(\vec{x}))$, where $f_1(\vec{x}), ..., f_m(\vec{x})$ are the original constraints of the problem. An ideal solution

$\vec{x}$ would thus have $f_i(\vec{x}) = 0$ for $1 \leq i \leq m$ and $f(\vec{x}) \leq f(\vec{y})$ for all feasible $\vec{y}$ (assuming minimization) [8, 39, 40, 41]. In many cases, it is not clear how one can balance different objectives by a weight function especially when the various objectives are defined over different domains. Also, it is not always possible to have a crisp ranking of the individual objectives. Another difficulty is that the outcome of such ranking is not always predictable especially when some of the criteria are correlated. Fuzzy logic provides a convenient framework for solving this problem. It allows one to map values of different criteria into linguistic values, which characterize the level of satisfaction of the designer with the numerical values of objectives. Each linguistic value is then defined by a membership function which maps numerical values of the corresponding objective criterion into the interval $[0, 1]$. The desires of the decision maker are conveniently expressed in terms of fuzzy logic rules and fuzzy preference rules. The executing/firing of such rules produces numerical values that are used to decide a solution goodness. In practice, this approach has been proven very powerful for finding compromise solutions in different areas of science and engineering [34].

2.4 Fuzzy Logic

Fuzzy logic deals with approximate rather than precise modes of reasoning. Therefore, fuzzy logic is capable of handling the uncertainty of data. Also, natural language, which is the basis of fuzzy logic, is more convenient for expressing engineering

problems. In general, fuzzy logic can be viewed as a nonlinear mapping of an input data vector into a scalar output. However, the flexibility of fuzzy logic may create lots of different mappings for a single problem instance. Therefore, a good understanding of the fuzzy set theory, fuzzy reasoning and fuzzy rules is needed.

2.4.1 Fuzzy Set Theory

An element in fuzzy logic may partially belong to a fuzzy set by a certain degree compared to classical (crisp) set in which each element can either belong to the set or not.

A fuzzy set A of universe of discourse X is defined as $A = \{(x, \mu_A(x)) \mid all\ x \in X\}$, where X is a space point and $\mu_A(x)$ is a membership function of x being an element of A. A membership function $\mu_A(x)$ is a mapping of x in A that maps X to the membership space M. The range of the membership function is a subset of the non-negative real numbers whose boundaries are finite [49]. Elements with zero degree of membership are normally not listed.

Fuzzy Reasoning

Unlike classical reasoning in which propositions are whether true of false, fuzzy logic establishes approximate truth value of propositions based on linguistic variables and inference rules [46]. A linguistic variable is a variable whose values are words or sentences in natural or artificial language. It is concerned with the use of fuzzy

values that captures the meaning of words, human reasoning and decision-making. An example of linguistic variable is circuit's area. This variable can be expressed by linguistic values like very small, small, average, large and very large circuit, rather than 20 μm^2, 30 μm^2, 50 μm^2, 75 μm^2, and 100 μm^2.

A linguistic variable carries the concept of fuzzy set qualifiers, called *hedges*. Hedges are terms that modify the shape of fuzzy sets. They include adverbs such as *very, somewhat, quite, more or less*, and *slightly*. They are used as modifiers, truth-values, probabilities, quantifiers and/or possibilities of a certain linguistic variable.

Formally, a linguistic variable comprises of five elements [47]:

1. The variable name
2. The primary term set
3. The universe of discourse U
4. A set of syntactical rules that allows composition of the primary terms and hedges to generate the term set
5. A set of semantic rules that assigns each element in the set a linguistic meaning.

Fuzzy Operators

There are two basic types of fuzzy operators. The operators for the intersection, interpreted as the logical "and", and the operators for the union, interpreted as the

logical "or" of fuzzy sets. The intersection operators are known as triangular norms (t-norms), and union operator as triangular co-norms (t-co-norms or s-norms) [49]. Some examples of s-norm operators are given below, (where A and B are the fuzzy sets of universe of discourse X).

1. Maximum. $[\mu_{A\bigcup B}(x) = max\{\mu_A(x), \mu_B(x)\}]$.

2. Algebric sum. $[\mu_{A\bigcup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)]$.

3. Bounded sum. $[\mu_{A\bigcup B}(x) = min(1, \mu_A(x) + \mu_B(x))]$.

4. Drastic sum. $[\mu_{A\bigcup B}(x) = \mu_A(x)$ if $\mu_B(x) = 0$, $\mu_B(x)$ if $\mu_A(x) = 0$, 1 if $\mu_A(x), \mu_B(x) > 0]$.

An s-norm operator satisfies commutativity, monotonicity, associativity and $\mu_{A\bigcup 0}(x) = \mu_A(x)$ properties.

Following are some examples of t-norm operators.

1. Minimum. $[\mu_{A\bigcap B}(x) = min\{\mu_A(x), \mu_B(x)\}]$.

2. Algebraic product. $[\mu_{A\bigcap B}(x) = \mu_A(x)\mu_B(x)]$.

3. Bounded product. $[\mu_{A\bigcap B}(x) = max(0, \mu_A(x) + \mu_B(x) - 1)]$.

4. Drastic product. $[\mu_{A\bigcap B}(x) = \mu_A(x)$ if $\mu_B(x) = 1$, $\mu_B(x)$ if $\mu_A(x) = 1$, 0 if $\mu_A(x), \mu_B(x) < 1]$.

Like s-norm, t-norms also satisfy commutativity, monotonicity, associativity and $\mu_A \bigcap 1(x) = \mu_A(x)$. Also, the fuzzy complementation operator is defined as follows.

$$\bar{\mu}_B(x) = 1 - \mu_B(x) \tag{2.4}$$

2.4.2 Multi-objective Optimization Using Fuzzy Logic

Approximate reasoning can be made based on linguistic variables and their values. Rules can be generated based on previous experience. The rules are expressed as **If** ... **Then** statements. Connectives such as AND and OR can be used in approximate reasoning to join two or more linguistic values. The If part (*antecedent*) is a fuzzy predicate defined in terms of linguistic values and fuzzy operators (AND and OR). The Then part is called the *consequent*.

In optimization problems, the linguistic value used in the consequent part identifies the fuzzy subset of good solutions. Therefore, the result of evaluation of the antecedent part identifies the degree of membership in the fuzzy subset of good solutions according to the fuzzy rule in question. If more than one rule is used to perform decision-making, each rule can be evaluated to generate a numerical value. Then, these numerical values from various evaluations of different rules can be combined to generate a crisp value on a higher level of hierarchy.

Consider the circuit design problem with minimization of area, delay, and power consumption. Three linguistic variables area, delay and power introduced. Then

good solutions can be characterized by the following fuzzy rule.

> *If the circuit has (small area) and (less delay) and (less power consumption) then it is a good solution.*

In the traditional fuzzy logic, the minmax operators are used to build the above fuzzy rule. However, it was shown in [44] that these operators can lead to undesirable behavior. This behavior has led to the development of other fuzzy operators such as the *Ordered Weighted Averaging (OWA)* operator explained below.

Ordered Weighted Averaging (OWA) Operator

Generally, the formulation of multi criterion decision functions neither desires the pure "AND-ing" of t-norm nor the pure "OR-ing" of s-norm. The reason for this is the complete lack of compensation of t-norm for any partial fulfillment and complete submission of s-norm to fulfillment of any criteria. Also the indifference to the individual criterion of each of these two forms of operators led to the development of Ordered Weighted Averaging (OWA) operators [42, 43]. This operator allows easy adjustment of the degree of "AND-ing" and "OR-ing" embedded in the aggregation. According to [42, 43], "OR-like" and "AND-like" OWA for two fuzzy sets A and B are implemented as given in Equations 2.5 and 2.6 respectively.

$$\mu_{A\cup B}(x) = \beta \times \max(\mu_A, \mu_B) + (1-\beta) \times \frac{1}{2}(\mu_A + \mu_B) \tag{2.5}$$

$$\mu_{A\cap B}(x) = \beta \times \min(\mu_A, \mu_B) + (1-\beta) \times \frac{1}{2}(\mu_A + \mu_B) \tag{2.6}$$

where β is a constant parameter in the range [0,1]. It represents the degree to which OWA operator resembles a pure "OR" or pure "AND" respectively.

2.5 Concluding Remarks

In this chapter, some basic background information were reviewed including the conventional logic design (CLD) and logic design optimization. Then, multi-objectives optimization was presented. Also, fuzzy logic and multi-objectives optimization using fuzzy logic is discussed.

Chapter 3

Evolutionary Logic Design (ELD)

3.1 Introduction

Evolutionary logic design considers a new concept for automatic design of digital systems: instead of using human conceived models, abstractions, and techniques, it employs search algorithms to develop good designs. This idea is summarized in Thompson [37]:

> Imagine a design space where each point in that space represents the design of a digital circuit. All possible digital circuits are there, given the component types available to electronics engineer, and the technological restrictions on how many components there can be and how they can interact. In this metaphor, we loosely visualize the circuits to be arranged in the design space so that 'similar' circuits are close to each

other.

Evolutionary logic design deals with a huge search space, requiring, therefore, powerful search techniques to handle the task. Naturally, when the search space is very large, random search has little chance to succeed. Hence, this new approach to circuit design does not resume choosing a design problem and applying a search techniques. Instead, some procedures have to be followed:

- The search space sampled by the algorithm must have its size limited. Although it is important to allow the sampling of a wide variety of topologies, some criteria should be chosen to control the number of possible solutions.
- It is usually necessary to adapt the search techniques to the particularities of the design problem.

The search space size is a very subtle issue: it should be large enough to include a good variety of novel circuit topologies; nevertheless, if the design space increases without restriction, the chances to find a good solution are very small. This is a very important issue for the search techniques applied in this thesis. These algorithms suit well to sample large design spaces, performing better than standard optimization techniques. Nonetheless, for very large search spaces, even evolutionary algorithms have their limitations.

Another important issue is the inclusion of special techniques into the search tool in order to successfully find a circuit that conforms with all specifications such

as the inclusion of some kind of previous knowledge on the circuit design.

After this discussion one may think: Why would we be interested in using search algorithms to perform logic design? Is not the size of the search space to be sampled a drawback that we would not have to face using conventional techniques? The following points will summarize the answer to this question:

- Potential to find *novel circuits*.
- The possibility to find *new design rules* from the novel circuits obtained.
- Evolutionary methods can contemplate a larger set of design specifications compared to human design techniques (area, delay, power).
- Evolutionary systems have been able to achieve competitive circuits when compared with the state of the art in electronics.
- The constant increase in the microprocessors' speed will partially alleviate the drawback of the design space size.

In this chapter, a brief description about evolutionary algorithms will given. Then, the concept of assemble and test and exploring the search space will be discussed. A survey on the work being done on evolutionary logic design and some observations on them will be presented.

3.2 Genetic Algorithm

Genetic algorithm operates on a population (or set) of individual(s) (or solution(s)) encoded as strings or matrices. These strings represent points in the search space. In each iteration, referred to as a generation, a new set of strings that represent solution(s) (called offsprings) is created by using some evolutionary operation such as mutation or crossover on the current generation. The encoding of the strings or matrices (chromosomes) is a very critical aspect for the efficiency of the algorithm. Therefore, an efficient encoding of digital circuits into chromosomes is important for the evolutionary logic design techniques. In the case of genetic algorithms, encoding represents the genotype-phenotype mapping between the circuit and the chromosomes. In order to preserve the validity of the circuit being evolved due to the evolution process, some constraints need to be set on the chromosome encoding.

In addition to the chromosome encoding, each chromosome is associated with a *fitness value (cost function)* to evaluate the goodness (quality) of the solution. The fittest the chromosome, the greater chance a chromosome will survive in the following generation. Therefore, the choice of the fitness (cost) function will affect the behavior of the evolutionary algorithm in the quality of the solution being produced.

The evolution process starts by creating an initial population. This initial population can be random or constructive, depending on the objective of the design. Miller et al. initialized the population with the chromosome of a given circuit and

its mutated copies [41]. The objective was to focus the evolution process on finding more efficient circuits. It is however noticed that many researchers prefer random initialization since it allows exploring more regions in the design space.

3.3 Exploring the Space of all Representations

The survivability of an organism can be seen as a process of assembling a larger system from a number of component parts and then testing the organism in the environment in which it finds itself (assemble-and-test). Figure 3.1 illustrates this concept in the general space of designs. The top-down rule-based space of designs is shown in gray as a small sub-region in the much larger space of all possible designs. Occasionally by a process of human inspiration or accidental discovery this space is widened as new concepts and principles are developed. Generally speaking, restrictive assumptions have to be made about the range of parts which can be used within this space. This is imposed by the constraints of a tractable system of rules. On the other hand, it is argued here that by employing the simple idea of assemble-and-test together with an evolutionary algorithm one can explore the entire design space and use a much larger collection of parts precisely because of the absence of imposed rules of design [28].

The concept of assemble-and-test together with an evolutionary algorithm to gradually improve the quality of a design has largely been adopted in the field of

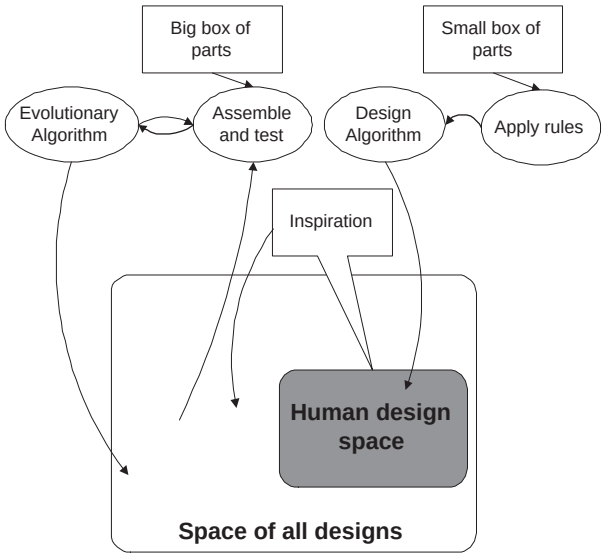


Figure 3.1: Conventional design versus evolutionary design with assemble-and-test.

Evolvable Hardware where the task is to build an electronic circuit. These electronic circuits are encoded in a *chromosome*. The complete set of chromosomes is called a *genotype*. The resulting circuit which can be constructed from a chromosome or genotype is called *phenotypes*. Research in Evolvable Hardware can be sub-divided into two main categories: intrinsic evolution and extrinsic evolution. The former refers to an evolutionary process in which each phenotype is built in electronic hardware and tested. The latter uses a model of the hardware and thus evaluates the phenotypes in software. Each of these categories can be further sub-divided into analogue or digital domains. Intrinsic evolution in the analogue domain has recently become possible because of the availability of reconfigurable analogue devices. In the introduction, it was shown how the use of an evolutionary algorithm combined with assemble-and-test could be used to explore a much larger area of design space

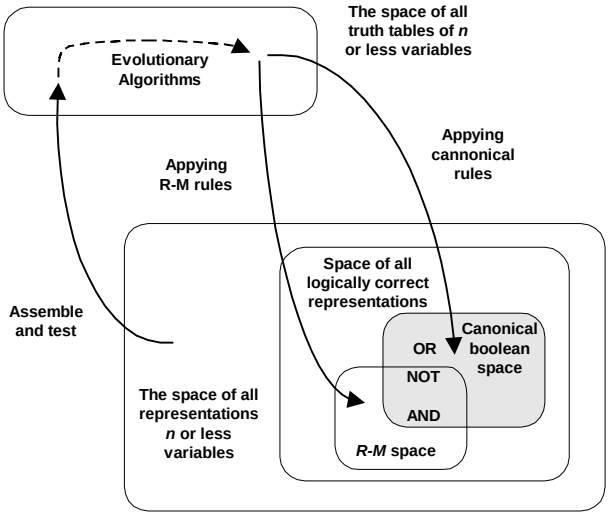


Figure 3.2: How "assemble-and-test" reaches the unknown regions of the space of all representations.

than using a top-down rule based design algorithm. Figure 3.2 shows a particular case of this for the problem of finding efficient representations of Boolean functions and it illustrates one of the fundamental concepts of this proposal. In conventional logic design one begins with a precise specification in the form of a truth table, binary decision diagram, symbolic expression etc. The expression is manipulated by applying canonical Boolean rules or Reed-Muller algebraic rules. One never escapes from the space of logically correct representations. The methods though powerful in that they can handle large numbers of input variables, yet they are not adaptable to new logical building blocks and require a great deal of analytical work to produce small optimizations in the representation. Assembling a function from a number of component parts begins in the space of all representations and maps it into the space of all the truth tables with m input variables ($m \leq n$). The evolutionary algorithm

then gradually pulls the specification of the circuit towards the target truth table (shown as a small shaded box). Thus the algorithm works in a much larger space of functions many of which do not represent the desired function.

3.4 Survey on Evolutionary Logic Design (ELD)

In the following subsections, the previous work on the evolutionary logic synthesis will be addressed in two aspects: the circuit encoding and the cost function.

3.4.1 The Circuit Encoding

A fixed length chromosome for circuit representation is used by Hounsell [20]. The first part of the chromosome is used for describing the inputs of the circuit while the last part is used for describing the outputs of the circuit. The position of each logic element is referenced within the chromosome. Figure 3.3 displays the relative location of each encoded section. Within the chromosome, a specific location is allocated for each logic element in the circuit. Interconnection between cells is not restricted to its nearest positional neighbor. Also, cells are free to connect to other cells at higher position within the chromosome. Feedback connections are not permitted. Figure 3.4 demonstrates the encoding of a macro block (full-adder) with its connectivity within the chromosome.

Another representation made by Miller who suggested that the chromosome rep-

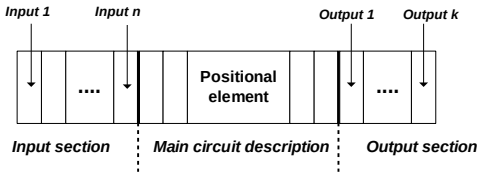


Figure 3.3: Chromosome representation in Hounsell [20].

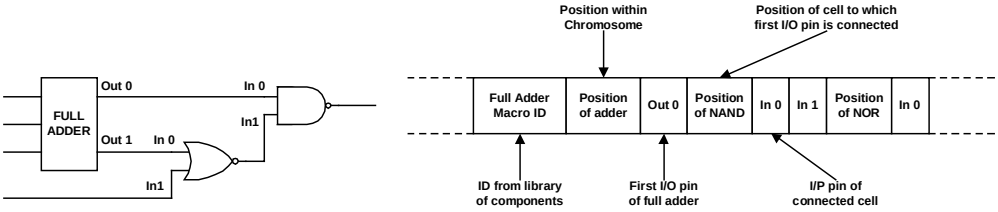


Figure 3.4: Macro blocks and its genotype representation in Hounsell [20].

resentation should match the hardware's geometry configuration [29]. A matrix of $n \times m$ array of logic cells is used as the phenotype representation in the case of FPGA. The interconnections together with gate level functionality for the cells are defined by the genotype (chromosome). This genotype-phenotype mapping is shown in Figure 3.5.

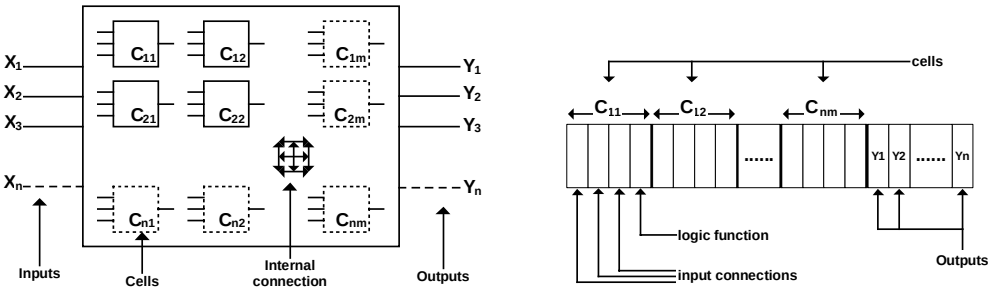


Figure 3.5: Chromosome representation in Miller [28, 29].

Any function that can be realized by an FPGA can be assigned to any gene of

Alphabet	Function	Alphabet	Function
0	0	10	$a \oplus b$
1	1	11	$a \oplus b$
2	a	12	$a + b$
3	b	13	$a + b$
4	$\bar{a}$	14	$\bar{a} + b$
5	b	15	$\bar{a} + b$
6	$a \cdot b$	16	$a \cdot \bar{c} + b \cdot c$
7	$a \cdot b$	17	$a \cdot \bar{c} + b \cdot c$
8	$\bar{a} \cdot b$	18	$\bar{a} \cdot \bar{c} + b \cdot c$
9	$\bar{a} \cdot b$	19	$\bar{a} \cdot \bar{c} + b \cdot c$

Table 3.1: Possible cell functions in Miller [28, 29].

the chromosome. Table 3.1 lists all the possible functions.

Each gene is a sequence of integers representing the target interconnection of gate's inputs and the gate type. Consider, for example, the case shown in Figure 3.6. The first quadruplet of the chromosome is 0-1-0-10, which means that the first input of the cell is connected to pin number 0, the second input to pin number 1, and the third input to pin number 0 respectively. The gate type is 10, which is a two input XOR (the third input is not used). The interconnection between cells is restricted by the *levels-back parameter*, which denotes the number of previous column in the array that a cell can be connected to. If the levels-back parameter is one, then each cell must be connected to its immediate neighbor in the previous column. Cells within any particular column cannot be connected together, and feedback connections are not allowed.

The third representation by Coello et al. [7, 8], used the same chromosome representation of circuits as those by Miller [28, 29]. However, the organization of the

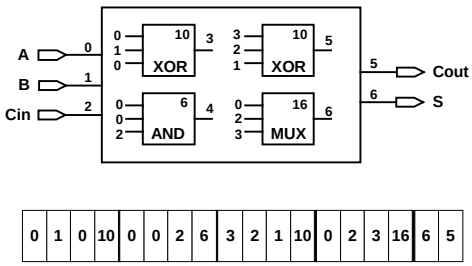


Figure 3.6: Example of genotype-phenotype mapping in Miller [28, 29].

Input 1	Input 2	Gate type

Figure 3.7: Representation of gene in chromosome in Coello [7, 8].

chromosome is different. Each cell is a gate of the type of AND, NOT, OR, XOR or WIRE. Each of them is encoded in a triplet of inputs and gate type, as illustrated in Figure 3.7.

Miller represented each cell as different integers [28, 29]. This approach is impractical for large circuit. On the other hand, Coello et al. represented a gate at position (i, j) can only be connected to the one at $((i-1), j)$ [7, 8]. This restriction reduces the cardinality of alphabet needed to represent the chromosome, since the integer number to represent each cell increases only column wise.

3.4.2 The Cost Function

The cost function *(fitness)* used by Hounsell is represented by a percentage of circuit functionality generated by the evolutionary algorithm [20]. Correctness is calculated by summing the total number of correct bits produced by the circuit solution under evaluation and comparing this to the desired output response. Fitness is expressed

mathematically as follows:

$$R_b = 2^I \times O$$

$$Fitness = E_b / R_b$$

Where R_b is the total number of bits comprising the desired output vectors, E_b is the actual number of bits matched with the desired output vectors during evaluation, O is the number of output pins and I is the number of input pins. Evaluation of the circuit (solution) is achieved through interaction with a HDL (Hardware Description Language).

Since all functions are specified by a truth table, Miller's cost function (fitness) of a genotype is based on the number of correct output bits specified by the truth table [29, 28]. For a one-bit adder with carry, there are 8 input cases and 2 outputs, this results in 16 output bits. A fully correct circuit would have fitness 16.

Colleo's cost function (fitness) evaluation works in two stages. At the beginning of the search, only validity of the circuit outputs is taken into account [7, 8] . Once a functional solution appears, then the fitness function is modified such that any valid designs produced are rewarded for each **WIRE** gate that they include, so that the algorithm tries to find a correctly functional circuit with the maximum number of **WIRE**s.

3.5 Observations

It is interesting to note that non-deterministic iterative heuristics tend to favor the use of XOR gates, since this gate allows producing, in many cases, solutions with a shorter symbolic representation. These solutions, are however not entirely obvious for human designers. Iterative heuristics tend to use multi-level XORs gates to produce the same effect as that a human designer would achieve using a combination of logic gates.

Although it has been shown that iterative heuristics can in fact produce fully functional circuits, or even efficient ones (in terms of the number of logic gates used), there is some concern about the process of producing a circuit. We have seen that in the evolutionary design approach, one considers the problem of designing a circuit as building a black box whose inputs and outputs are the inputs and outputs of the desired circuit. The details of the circuit itself are encoded in the form of a chromosome. Evolutionary algorithms will then blindly evolve circuits according to a given set of objectives.

Another point to consider is redundancy in the evolving circuit. It has been shown that some forms of redundancy could be useful for the evolution process [17]. However, if the genotype representation and genetic operators are not well implemented, we may end up with circuits having many redundancies. This redundancy may in turn spoil the evolution, making iterative heuristics to explore regions where

there are no acceptable solutions at all.

Some of the key points observed after studying the above mentioned approaches are summarized below.

1. Instead of blindly evolving a circuit, we need some procedures to guide the evolution process. These procedures should help iterative heuristics to find an optimal solution. We believe that integrating some rules in logic synthesis is the answer to this problem. These can be used to direct the search. One easy example of these rules is putting an inverter at the output of (sub-)circuit built to toggle the fitness of the current solution.

2. We can view the problem of circuit design as a problem of assembling logic blocks to implement a desired functionality. Almost similar to the growing circuits approach used by Miller [30], we shall constructively build a circuit by assembling all the required logic and/or sub-functions through the evolution process.

3.6 Concluding Remarks

In this chapter, we have introduced evolutionary logic design in the first section followed by some discussion on evolutionary algorithms. Then, a survey on pervious work of evolutionary logic design has been presented. This is followed by some observations on these implementations.

Chapter 4

Simulated Evolution (SimE) Algorithm

4.1 Introduction

Simulated Evolution (SimE) which was proposed by Kling and Banerjee in 1987 is based on an analogy with the principles of natural selection thought to be followed by various species in their biological environments [21]. During the process of biological evolution, organisms tend to develop features that allow them to adapt to the peculiarities of their environment. The more an organism adapts to its environment, the better are its chances of survival. In other words, by adapting, an organism optimizes its chances of surviving in its environment. Hence, adaptation is seen as a form of optimization. This similarity has given rise to a new class of

randomized iterative algorithms which consists of ***Genetic Algorithms, Simulated Evolution, and Stochastic Evolution***. All three algorithms of this class are general randomized search heuristics that are based on concepts learned from biological evolution. For all three algorithms, the cost function is an estimation of the degree of adaption of a particular solution to the target objective. For a maximization problem, the higher the value of the objective function is, the more that particular solution is adapted to its environment. *Simulated Evolution* will be the subject of this chapter [21, 34].

In this chapter, SimE algorithm is introduced. Also, a detailed analysis of the SimE algorithm and operators are addressed afterward. This is followed by a qualitative comparison of SimE and Genetic Algorithm (GA).

4.2 SimE Algorithm: Evaluation, Selection and Allocation

Combinatorial optimization problems seek to find a global optimum of some real valued cost functions $cost : \Omega \rightarrow \mathrm{R}$ defined over a discrete set Ω. The Set Ω is called the state space and its elements are referred to as states. A state space Ω together with an underlying neighborhood structure (the way one state can be reached from another state) form the solution space.

The Simulated Evolution (SimE) algorithm is a general search strategy for solv-

ing a variety of combinatorial optimization problems [21]. The SimE algorithm starts from an initial assignment, and then, following an evolution-based approach, it seeks to reach better assignments from one generation to the next. SimE assumes that there exists a population P of a set M of k elements. In addition, there is a cost function *Cost* that is used to associate with each assignment of an element m a cost C_m. The cost C_m is used to compute the goodness (fitness) g_m of element m, for each $m \in M$. Furthermore, there are usually additional constraints that must be satisfied by the population as a whole or by particular elements. A general outline of the SimE algorithm is given in Figure 4.1.

SimE algorithm proceeds as follows. Initially, a population[1] is created at random from all populations satisfying the environmental constraints of the problem. The algorithm has one main loop consisting of three basic steps, *Evaluation*, *Selection*, and *Allocation*. The three steps are executed in sequence until the population average *goodness* reaches a maximum value, or no noticeable improvement to the population *goodness* is observed after a number of iterations. Another possible stopping criterion could be to run the algorithm for a prefixed number of iterations (see Figure 4.1).

Combinatorial optimization problems can be modeled in a number of ways. A generic formulation is the following: *Given a finite set M of distinct movable ele-*

[1]In SimE terminology, a population refers to a single solution. Individuals of the population are components of the solution; they are the movable elements.

ALGORITHM *Simulated_Evolution(E, ,L, Stopping-Criteria)*;
$INITIALIZATION$;
Repeat
  $EVALUATION$:
    ForEach $m \in M$ **Do** $g_m = \frac{O_m}{C_m}$ **EndForEach**;

  $SELECTION$:
    ForEach $m \in M$ **Do**
      If $Random \leq Min(1 - g_m + B; 1)$
        Then $P_s = P_s \cup \{m\}$;
      Else $P_r = P_r \cup \{m\}$;
      EndIf;
    EndForEach;
  Sort the elements of P_s;
  $ALLOCATION$:
    ForEach $m \in P_s$ **Do** $F_a(m)$ **EndForEach**;
Until *Stopping-criteria are met*;
Return ($BestSolution$);
End *Simulated_Evolution.*

Figure 4.1: Simulated Evolution algorithm [21, 34].

ments and a finite set L of locations, a state is defined as an assignment function $S: M \rightarrow L$ *satisfying certain constraints.*

Many of the combinatorial problems can be formulated according to this generic model. Below is an example on Quadratic Assignment Problem (QAP):

Example 1 *Quadratic Assignment Problem*(**QAP**)

Problem: Given a set M of n modules and a set L of $|L|$ locations. $|L| \geq n$. Let $c_{i,j}$ be the number of connections between elements i and j, and $d_{k,l}$ be the distance between locations k and l.

Objective: Assign each module to a distinct location so as to minimize

the wire length needed to interconnect the modules. To formulate QAP in terms of the above state model, choose $M = \{1, 2, \ldots, \mid M \mid\}$ and $L = \{1, 2, \ldots, \mid L \mid\}$. Then a state is defined as the *onto function* $S : M = \{1, 2, \ldots, \mid M \mid\} \rightarrow \{1, 2, \ldots, \mid L \mid\}$. In this case, one additional constraint is required, which can be stated as $S(i) \neq S(j)\ \forall\ i \neq j$, i.e., no two elements are assigned to the same location.

The cost of a state, $Cost(S)$ is the wire length required to interconnect all the elements in their present locations. That is,

$$Cost(S) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} d_{S(i),S(j)}$$

4.2.1 Evaluation

The Evaluation step consists of evaluating the goodness of each individual i of the population P (see Figure 4.2). The goodness measure must be a single number expressible in the range [0,1]. Goodness is defined as follows:

$$g_i = O_i / C_i$$

where O_i is an estimate of the optimal cost of individual i, and C_i is the actual cost of i in its current location.

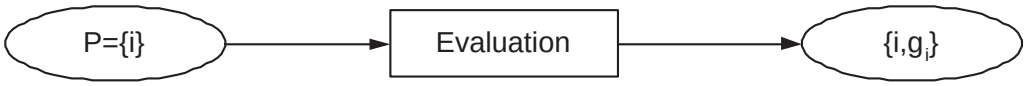


Figure 4.2: Evaluation.

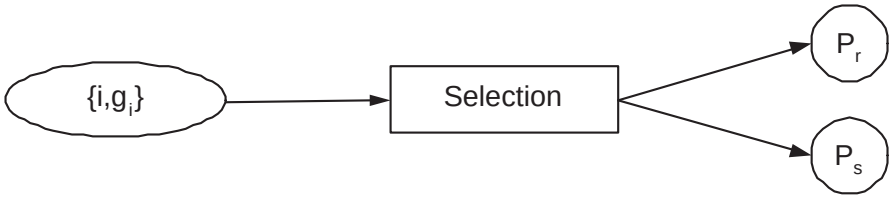


Figure 4.3: Selection.

4.2.2 Selection

The second step of the SimE algorithm is *Selection*. Selection takes as input the population P together with the estimated goodness of each individual, and partitions P into two disjoint sets, a selection set P_s and a set P_r of the remaining members of the population (see Figure 4.3). The decision whether to assign individual i to the set P_s or set P_r is based solely on its *goodness* g_i. The *Selection* operator uses a selection function *Selection* (see Figure 4.4). The *selection* operator has a nondeterministic nature. An individual with a high *goodness* still has a non zero probability of being assigned to the selected set P_s. It is this element of nondeterminism that gives SimE the capability of escaping local minima.

4.2.3 Allocation

Allocation is the SimE operator that has most impact on the quality of solution. *Allocation* takes as input the two sets P_s and P_r and generates a new population P'

Function *Selection(m, B)*;
/* m: is a particular movable element; */
/* B: Selection bias;*/
    If Random $\leq 1 - g_m + B$ **Then Return** True
        Else Return False
    EndIf
EndSelection;

Figure 4.4: Selection in the SimE of Figure 4.1.

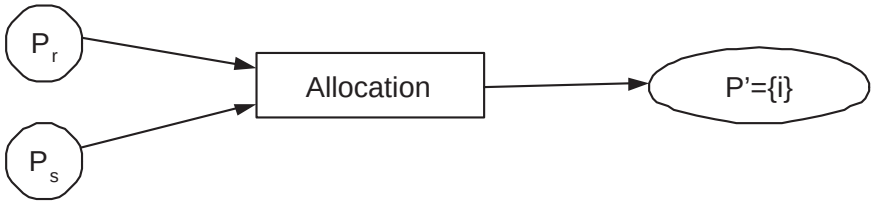


Figure 4.5: Allocation.

which contains all the members of the previous population P, with the elements of P_s mutated according to an allocation function *Allocation* (see Figure 4.5) [34].

The choice of a suitable *Allocation* function is problem specific. The decision of the *Allocation* strategy usually requires more ingenuity on the part of the designer than the *Selection* scheme. The *Allocation* function may be a nondeterministic function which involves a choice among a number of possible mutations (moves) for each element of P_s. Usually, a number of trial-mutations are performed and rated with respect to their goodnesses. Based on the resulting goodnesses, a final configuration of the population P' is decided. The goal of *Allocation* is to favor improvements over the previous generation, without begin too greedy.

Allocation functions can be *local* or *global*. With *local Allocation*, a selected individual is altered on the basis of local information so that only local alternations within the immediate neighborhood of that individual are allowed. On the other hand, *global Allocation* uses global information about all the individuals of the population so that it may affect any of the individuals in the entire population. *Allocation* alters (mutates) all the elements in the selected set P_s one after the other in a predetermined order. The order as well as the type of mutation are problem specific. For each individual e_i of the selected set P_s, W distinct trial alternations are attempted. The trial that leads to the best configuration (population) with respect to the objective being optimized is accepted and made permanent.

4.2.4 Initialization Phase

This step precedes the iterative phase. In this step, the various parameters of the algorithm are set to their desired values, namely, the maximum number of iterations required to run the main loop, the selection bias B, and the number of trial alternations W per individual. Furthermore, like any iterative algorithm, SimE requires that an initial solution be given. The convergence aspects of SimE are not affected by the quality of the initial solution.

4.3 Comparison of Simulated Evolution and Genetic Algorithm (GA)

In this section, we look at the main differences between SimE and GA. GA is another evolution based randomized iterative algorithm. GA and SimE follow a similar strategy in exploiting evolution to move from one generation to the next. However, there are significant differences between the two algorithms:

1. SimE works with a single solution called population. The constituents of a solution are called individuals or elements. On the other hand, GA works with a set of solutions. A single solution is an individual (also called a chromosome), and an individual (solution) is made up of genes.

2. GA relies on genetic reproduction. The population of next generation is selected among individuals of current generation and their offsprings. Fitter individuals have higher probabilities of surviving to the next generation. Offsprings are reproduced using crossover between selected pairs of parent individuals of current population. Also, a small fraction of the individuals may undergo mutation. In contrast, SimE maintains a single individual throughout the generations. Evolution from one generation to the next uses genetic mutation only whereby some elements of current solution are altered. SimE has no crossover since this operator requires two individuals.

3. In SimE, an individual is evaluated by estimating the fitness of each one of its genes. The single individual of the next generation is obtained by probabilistically altering some of the genes of the current individual (single parent of current generation). Genes with lower fitnesses have higher probabilities of getting altered. On the other hand, GA computes the fitness of complete solutions. In general, solutions with higher fitnesses have higher probabilities for mating. Usually, the fittest among the parents and their offsprings survive to the next generation. Therefore, though both SimE and GA perform a stochastic evolutionary-based search of the state space, SimE is more greedy, and thus usually requires fewer iterations to converge toward desirable solutions.

Overall, SimE algorithm usually runs much faster than GA algorithm. The reason is that the concept of fitness helps the algorithm converge quickly to a near optimal solution. Furthermore, since a single solution is maintained at all times, it has much less time and space requirements than GA [34].

4.4 Concluding Remarks

In this chapter, Simulated Evolution (SimE) Algorithm is introduced. Also, a detailed explanation about SimE and its different operators is presented. Finally, a comparison between SimE algorithm and Genetic Algorithm (GA) is shown.

Chapter 5

Simulated Evolution Algorithm (SimE) for Logic Design

In this chapter, the proposed use of Simulated Evolution Algorithm (SimE) in the design of logic circuits is discussed. The data structure being used in the implementation of the algorithm is presented first. Then, SimE operators being implemented in the proposed algorithm are explained in detail. Also, the proposed implementation of Tabu search algorithm is discussed and the incorporation of Tabu Search with SimE is given.

5.1 Introduction

A generic formulation of combinatorial optimization problems is the following: *Given a finite set M of distinct movable elements and a finite set L of locations, a state is defined as an assignment function $S : M \rightarrow L$ satisfying certain constraints.* Many of the combinatorial problems can be formulated according to this generic model. The design of logic circuit problem can be formulated using this generic model as follows:

Problem: Given a set M of k modules consisting of all types of logic gates and wires with all different input configurations and a set L of $|L|$ locations. $|L| \leq k$.

Objective: Selecting some of the modules from M to allocate them into the L distinct locations. The objective of this selection and allocation is to produce a required logic function given by its truth table and to have this allocation to be minimal according to some cost function (power, area and delay).

To formulate digital logic design in terms of the above state model, choose $M = \{1, 2, \ldots, k\}$ and $L = \{1, 2, \ldots, |L|\}$. Then a state is defined as the onto function $S : m = \{1, 2, \ldots, k'\} \rightarrow \{1, 2, \ldots, |L|\}$ where $m \subseteq M$. Figure 5.1 is a representation of the digital logic design problem addressed in this context. In this case one additional constraint is required, which can be stated as $S(i) \neq S(j)\ \forall i \neq j$, i.e., no two elements are assigned to the same location. The cost of a state, *Cost(S)* is a compound cost considering the correctness of the outputs of the solution circuit

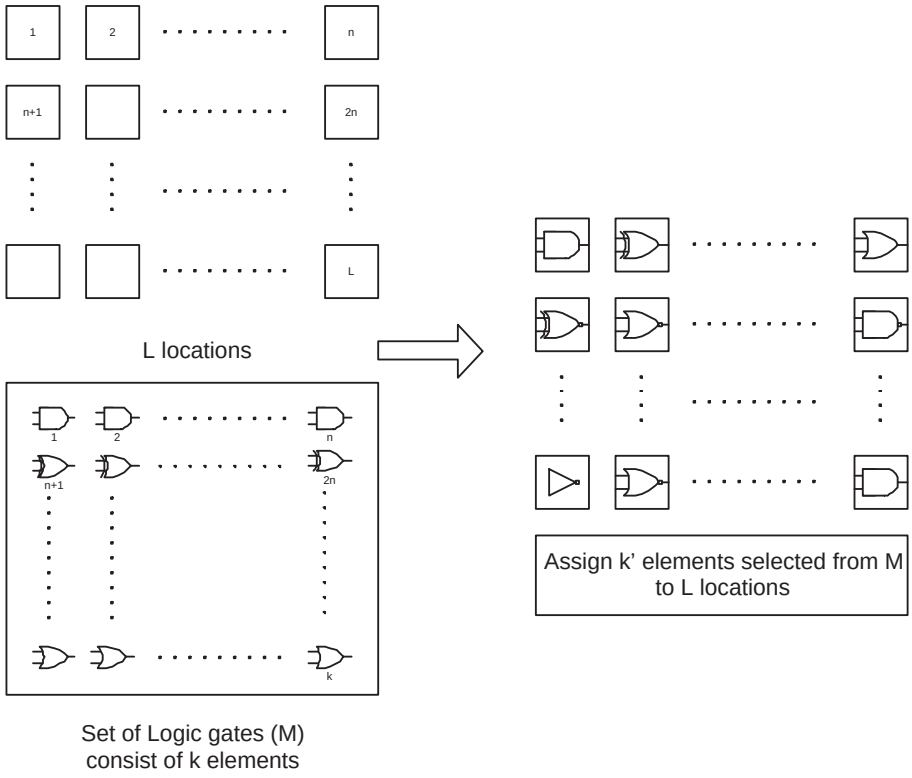


Figure 5.1: Representation of the digital logic design problem in SimE.

matching the required function truth table and other metrics such as power, delay and area. A detailed explanation on the *Cost(S)* function proposed is given later in this chapter.

5.2 Circuit Encoding

In order to represent a logic circuit, a 2 dimensional matrix is used. This representation has been adopted by Coello [7]. This type of data structure is similar to the structure of digital circuit. Therefore, the genotype-phenotype mapping is an easy

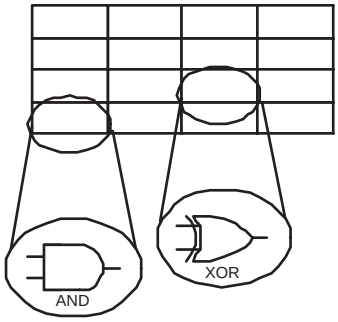


Figure 5.2: The matrix representation of a circuit.

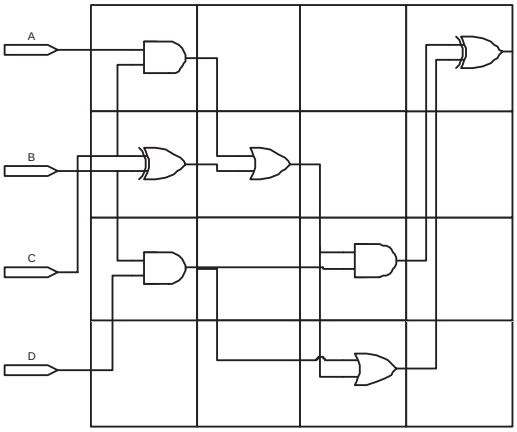


Figure 5.3: An example of a 4 input circuit.

task. Figure 5.2 shows an instance of the data structure being used in the proposed algorithm. Also, Figure 5.3 shows an example of a complete circuit with 4 inputs.

The size of the matrix is variable and it is relative to n where n is number of inputs of the required circuit. An initial value for the size of the matrix is given, which is equal to n. This value is variable during run time where columns and/or rows can be added. However, adding more columns and rows that are unneeded will

increase the size of the search space. Therefore, the performance of the algorithm will degrade. Certain measure has to be taken in order to add more columns and/or rows such as number of iteration and/or the goodness of the best fit solution. If the change in the goodness after a large number of iterations is relatively small, an increase in the number of columns and/or rows might help the algorithm to explore larger search space. On the other hand, columns and/or rows can be removed from the matrix in order to reduce the size of the search space. This case can happen if the algorithm were able to reach a high goodness value for a circuit in a smaller number of cells of the matrix. Therefore, reducing the number of columns and/or rows could help the algorithm to converge faster.

Each cell of the matrix is considered to be an individual. The collection of all individual of the matrix represents a solution. Each cell of the circuit matrix is encoded in a triplet of inputs and gate type, as illustrated in Figure 5.4. The first two number are for the inputs (input1, input2) and the third indicates the gate type. The value of input1 and input2 indicates the row from which the current cell is getting its input. The value of the gate type indicates the type of the gate being assigned to that cell from a predetermined set of gate types. A gate at position (i, j), where i is the column number and j is the row number, can only be connected to the one at $((i-1), j')$ and j' is any row in the previous column.

Table 5.1 list different type of gates with its code for the gene encoding. All of the results reported in this thesis may use any gate type as our building blocks.

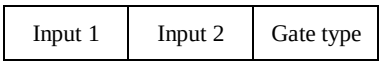


Figure 5.4: Representation of individual in matrix.

Code	Type	Function(F)
0	WIRE	a
1	NOT	a'
2	OR	$a + b$
3	AND	$a \cdot b$
4	XOR	$a \oplus b$
5	NOR	$(a + b)'$
6	NAND	$(a \cdot b)'$
7	XNOR	$(a \oplus b)'$

Table 5.1: All gate types used by SimE.

In addition to inputs and gate type, each cell of the matrix will have a field for the *goodness* of the gate at that cell. Additional information such as gate capacitance, size and delay are included in each cell in order to be used for the *cost function* computation.

5.3 Proposed SimE Algorithm: Parameters and Operators

SimE Algorithm consists of three steps: evaluation, selection and allocation. Figure 4.1 shows an outline of the algorithm. More detail about the parameters used and each step of the SimE algorithm is given in this section. The parameters used by the SimE algorithm are:

1. *Selection Bias* (B)

2. The number of iterations (W)

3. Population Size

The above parameters will be discussed in the initialization phase subsection. The operators used by the SimE algorithm are:

1. *Selection* operator

2. *Allocation* operator

Selection operator and allocation operator are considered individually.

5.3.1 Initialization Phase and Parameters

This step proceeds the iterative phase of the SimE algorithm. In this step, an initial value is given to each *Selection Bias* (B) and the number of iteration (W). Also, an initial solution is generated randomly.

The *Selection Bias* (B) is initialized to $B = -0.05$ in order to increase the number of cells being selected. However, if the size of the circuit being investigated is large, it is recommended to use $B = 0$ to have a smaller selection set.

Since the number of trial alternations W (moves) per individual is problem specific, W is initialized to a value that is relative to the number of gate types being used as building blocks and the size of the matrix. If the number of gate types used

is m and the size of the matrix is $n \times n$ where n is the number of the inputs of the circuit, then W is initialized as follows:

$$W = m\frac{n!}{2!\,(n-2)!}$$

where m is the number of gate types, 2 is the number of inputs for each gate and n is the number of inputs of the required logic circuit.

5.3.2 Selection Operator

The selection step is the second step in the SimE algorithm. It scans all cells of the population P. Based on the cell's *goodness* value, some cells is selected for reallocation in the *Allocation* function. The *goodness* of each cell is computed at the evaluation step. For every cell_i, there is a goodness value g_i associated with it. If randomly generated number between [0,1], with normal distribution, is less than or equal to $1 - g_i + B$, element i will be selected. This step is shown in Figure 5.5.

Therefore, if $B = -0.05$, then each element i has a probability $p_i = \min(1 - g_i - 0.05, 0.95)$ of getting selected for alteration. Therefore, when $g_i = 1$, $p_i = 0$. Also, when $g_i = 0$, $p_i = 0.95$.

Function *Selection(m, B)*;
/* m: is a particular movable element; */
/* B: Selection bias;*/
If Random $\leq 1 - g_m + B$ **Then Return** True
Else Return False
EndIf
EndSelection;

Figure 5.5: Selection in the SimE.

5.3.3 Allocation Operator

Allocation step is the third step in the SimE algorithm. It takes all the cells that have been selected by the selection step and reallocate them after mutation. After allocation, a new population P' is generated.

In our case, the allocation function will allocate each element e_i from the selection set P_s and mutates it with W trial. If there is any improvement in the goodness, it will make the move permanent.

The allocation function implemented is hybrid (global and local). When allocating the elements after mutation, it will consider the individual goodness (cell goodness) change. If the goodness improves, it will accept the move. However, after allocating all of the elements in the selection set P_s, it will check the global goodness and if it improves, the solution will be made as a current solution. If it did not improve, it will keep the old solution as current solution. A pseudo code of the allocation function implemented in our proposed SimE algorithm is outlined in Figure 5.6.

```
Allocation:
/* m: a cell from the matrix either selected or not */
/* m': mutated cell */
/* P_r: set of all remaining cells (unselected) */
/* P_s: set of selected cells */
/* W: number of trial */
/* P: current solution */
/* P': new solution */
    ForEach m ∈ P_r  Do
        Allocate m in P' in the same cell location in P
    EndForEach
    ForEach m ∈ P_s  Do
        For i = 0 to W  Do  /* W trial mutations */
            Mutate(m) /* Change randomly gate type and/or inputs */
            Place m' in the original selected cell location
            Evaluate (m') /* local allocation */
            If goodness (m') > goodness (m) then
                Accept move
                Exit ForLoop /* Exit W trial */
            EndIf
        EndFor /* End for W trial */
    EndForEach
    If Evaluate(P') > Evaluate(P) then /* global allocation */
        P = P'
        Exit ForLoop /* Exit W trial */
    Else
        P = P /* keep original */
    EndIf
```

Figure 5.6: Allocation function in SimE.

EVALUATION($*p$)
/* $*p$: a pointer to either a cell (individual) or a solution (population P) */
/* O_i: estimate of optimal cost of i individual */
/* C_i: actual cost of i individual */
/* g_i: goodness of i individual */
/* m: a cell from the matrix */
If $*p$ is an individual **then**
$g_{*p} = O_{*p}/C_{*p}$
ElseIf $*p$ is a solution **then**
ForEach $m \in *p$ **Do**
$g_m = O_m/C_m$
EndForEach
$Cost(*p)$ /* the cost of the whole population (power, area, delay) */
EndIf

Figure 5.7: Evaluation function in SimE.

5.3.4 Evaluation Function

As discussed in Chapter 4, the Evaluation step consists of evaluating the goodness of each individual i of the population P. The goodness measure must be a single number expressible in the range [0,1].

In our case, the Evaluation step will also evaluate the goodness of the whole solution P in order to count for other cost functions such as power, area and delay. An outline of the proposed evaluation function in the SimE algorithm is shown in Figure 5.7.

5.4 Hybrid SimE Using Tabu Search (TS)

In this section a proposed hybrid SimE algorithm for the deign of digital logic circuits is presented. First, *tabu search* (TS) is discussed in general. Next, a proposed implementation of *tabu search* for logic design is introduced. Finally, a hybrid SimE algorithm for logic design using tabu search is proposed.

5.4.1 Tabu Search (TS)

Tabu search is a form of local neighborhood search. Each solution $\mathcal{S} \in \Omega$ has an associated set of neighbors $\aleph(\mathcal{S}) \subseteq \otimes$. A solution $\mathcal{S}' \in \aleph(\mathcal{S})$ can be reached from $\mathcal{S}$ by an operation called a *move* to $\mathcal{S}'$. Normally, the neighborhood relation is assumed symmetric. That is, if $\mathcal{S}'$ is a neighbor of $\mathcal{S}$ then $\mathcal{S}$ is a neighbor of $\mathcal{S}'$ [34].

At each step, the local neighborhood of the current solution is explored and the best solution in that neighborhood is selected as the new current solution. Tabu search continues the search from the best solution in the neighborhood even if it is worse than the current solution. To prevent cycling, information pertaining to the most recently visited solution are inserted in a list called *tabu list*. Move to tabu solution are not allowed. The tabu status of a solution is overridden when certain criteria (aspiration criteria) are satisfied. One example of an aspiration criterion is when the cost of the selected solution is better than the best seen so far, which is an indication that the search is actually not cycling back, but rather moving to

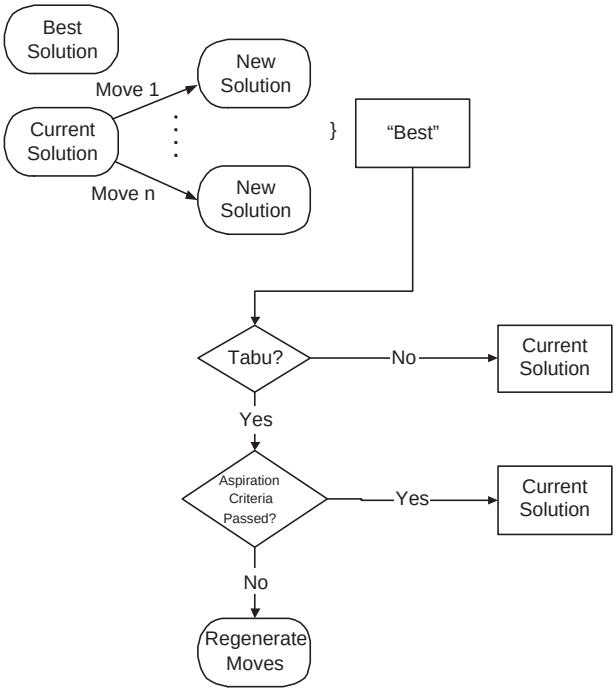


Figure 5.8: Flow chart of the tabu search algorithm [34].

a new solution not encountered before. A flow chart illustrating the basic tabu search algorithm is given in Figure 5.8. Also, an algorithmic description of a simple implementation of the tabu search is given in Figure 5.9.

5.4.2 Tabu Search for Logic Design

In the context of combinational logic circuits design, *tabu search* (TS) algorithm is implemented. The same circuit encoding used in SimE is used in TS. Also, the same cost function proposed in SimE have been used in TS.

The size of the tabu list used in the implementation for logic design is variable and it is between 5 and 7. Also, the attributes that are considered in the tabu

ALGORITHM Tabu_Search();
/* Ω: Set of feasible solutions */
/* $\mathcal{S}$: Current solution and $\mathcal{S}^*$: Best admissible solution */
/* $Cost$: Objective function */
/* $\aleph(\mathcal{S})$: Neighborhood of $\mathcal{S} \in \Omega$. */
/* $\mathbf{V}^*$: Sample of neighborhood solutions. */
/* **T**: Tabu list. */
/* **AL**: Aspiration Level */
INITIALIZATION (initial solution $S \in \Omega$, initialize **T** and **AL**);
For fixed number of iterations **Do**
Generate neighbor solution $\mathbf{V}^* \subset \aleph(\mathcal{S})$.
Find best $\mathcal{S}^* \in \mathbf{V}^*$.
If move $\mathcal{S}$ to $\mathcal{S}^*$ is not in **T** **Then**
Accept move, Update($\mathcal{S}$, **T**, **AL**), increment iteration
Else
If $Cost(\mathcal{S}^*) <$ **AL** **Then**
Accept move, Update($\mathcal{S}$, **T**, **AL**), increment iteration
EndIf
EndIf
EndFor

Figure 5.9: Tabu Search algorithm (TS).

list is the type of the gate being changed in the move and the inputs connections (row numbers). Moreover, the size of the samples of neighborhood solutions ($\mathbf{V}^*$) is adjusted to n where n is the number of inputs in the required circuit. Unlike SimE, each new solution generated contains only a single move of a cell where a change of one of the cells in a matrix is considered. In addition, the **AL** is considered to be as the best solution seen so far in the global optimization cost function where both functional correct circuit (functional fitness) and optimization fitness have to be the best seen so far.

ALGORITHM Hybrid_Simulated_Evolution()
  INITIALIZATION
  Repeat
    EVALUATION
    SELECTION
    ALLOCATION
    TABU SEARCH /* intensifying the search and escape some local optima */
  Until Stopping-criteria are met
  TABU SEARCH /* intensifying the search */
  Return (BestSolution)
End Hybrid_Simulated_Evolution

Figure 5.10: Hybrid Simulated Evolution algorithm.

5.4.3 Hybrid SimE Algorithm

The investigation in TS lead to the proposal of a hybrid SimE where a combination of both SimE and TS is considered in the design of combinational logic circuit.

The proposed hybrid algorithm uses SimE in order to diversify the search in the search space since each iteration in SimE is evaluating a compound moves (allocation of selection set). Moreover, hybrid SimE uses TS in order to intensify the search whenever SimE converges into some optima. TS will try to search for better solution in the neighboring solutions since it is using only single moves and evaluating each move. Also, it might help SimE to escape some local optima. Figure 5.10 shows pseudo code of the proposed hybrid SimE.

5.5 Concluding Remarks

In this chapter, a proposed implementation of the Simulated Evolution Algorithm (SimE) in the design of logic circuits is presented. Also, a detailed explanation about SimE operators is given. Finally, Tabu Search (TS) in logic design is presented and a discussion on the proposed hybrid SimE is given.

Chapter 6

Goodness Measurements

In this part, two *goodness* measures are proposed for the problem of evolutionary logic design. Then, the inclusion of power, area, delay in the cost function is proposed. Also, the use of fuzzy logic in the cost function is presented.

6.1 Proposed Goodness Measures

As pointed out in Chapter 4, the ***goodness measure*** has a great impact on the performance of the SimE algorithm and the quality of the solution obtained. In this part, two proposed *goodness measures* will be discussed in detail. The first *goodness measure* is based on the output pattern of the required circuit (Pattern Based Goodness). The second proposed *goodness measure* is based on the multilevel logic synthesis (Multilevel Logic Based Goodness). The values that these *goodness*

measures take is between 0 and 1 [0,1]. Each *goodness* measure will be explained in detail.

Both of the above *goodnesses* are categorized as *functional goodness measures* where the correctness of the obtained logic circuit should match the input truth table of the required function in order to get a goodness of 1. The other category of *goodness* measure is the *optimization goodness measure* where the extent of the optimality of the logic circuit synthesized is taken into consideration. Such measures of optimality are power, delay and area. The later category is presented in the objectives optimization subsection.

Two functional goodness measures are proposed and these are: pattern based goodness and multilevel logic based goodness.

6.1.1 Pattern Based Goodness

The pattern based goodness measure relies on the truth table of the required function to be implemented. Each cell i in the solution matrix is evaluated based on the number of correct outputs being generated compared to the required truth table. However, there are some cases where a cell i with high goodness might have its input cells to have low goodness. As a result, the input cells might get selected for reallocation due to its low goodness which will disturb the goodness of cell i. Therefore, some measures has to be taken to avoid such problems.

In order to solve the above case, *pattern based goodness measure* is divided into

two: *intrinsic goodness measure* denoted by g_i and *extrinsic goodness measure* denoted by G_i. The *intrinsic goodness measure* is related to the goodness of cell i based on the number of correct output patterns produced by cell i and matching the objective truth table. The *extrinsic goodness measure* is related to the effect on other cells inputs of cell i. The following is a formulation of the ***Pattern Based Goodness Measure***:

$g_i = {}^{O_i}/_{C_i}$

$\rho_i =$ # of matching output patterns of the i^{th} cell

$n =$ # of inputs in the circuit

$g_i = \rho_i / 2^n$ Intrinsic Functional Pattern Based Goodness Measure

while the Extrinsic Functional Pattern Based Goodness is done as in Figure 6.1. An

Extrinsic Functional Pattern Based Goodness Evaluation
/* g_i: Intrinsic Functional Pattern Based Goodness of cell i */
/* g_j: Intrinsic Functional Pattern Based Goodness of one of the input cells of cell i */
/* G_i: Extrinsic Functional Pattern Based Goodness of cell i */
For $i = n^2$ downto 1 **Do**
If $g_i > g_j$ of any of its inputs cells or $G_i > g_j$ of any of its inputs cells **then**
$G_j = (g_i + g_j)/2$ for the input cell with g_j less than g_i
or $G_j = (G_i + g_j)/2$ for the input cell with g_j less than G_i
EndIf
If gate type of i is inverter **then**
$g_j = g_i$ and $G_j = G_i$
EndIf
EndFor

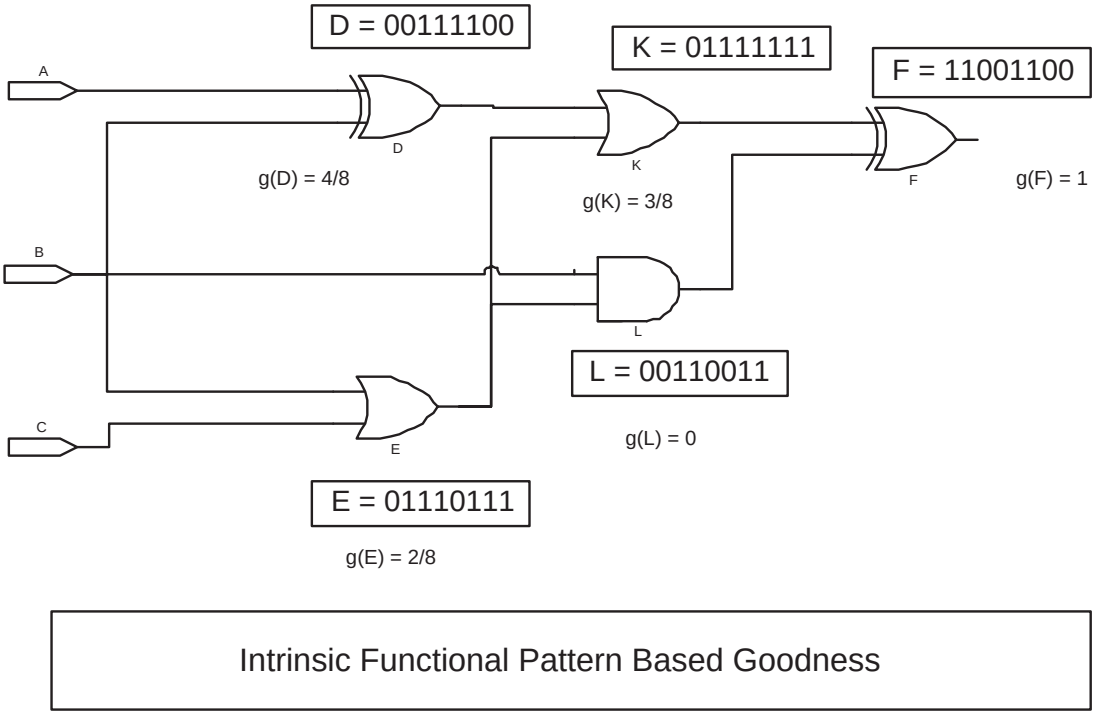
Figure 6.1: Extrinsic functional pattern based goodness.

example on the usage of the previous goodness measure is shown in Figure 6.2. In this example, the required function F is 11001100.

Both *Selection Function* and *Allocation Function* will use this *goodness measure* in order to guide the algorithm to reach to an optimal circuit. The selection function considers the extrinsic goodness measure during selection procedure of cells. On the other hand, the allocation function considers the intrinsic goodness measure in evaluating the mutated cells after allocation in order to accept a certain move. This goodness measure has been used to implement a version of the SimE algorithm denoted by SimE-G1.

6.1.2 Multilevel Logic Based Goodness

The Multilevel Logic Based *goodness* is based on the multilevel logic synthesis. It assumes that if a logic gate is located at higher level in the circuit, a larger portion of the required truth table minterms is covered. The higher the level of a gate in a multilevel logic circuit, the more minterms assumed to be covered at the output of the gate. Therefore, the *goodness* of a gate is affected by number of minterms covered at its output and the level where the gate is. Figure 6.3 illustrates this assumption. In the figure, the size of the circuit is 4 so there are 16 minterms should be generated at the output correctly. Initially, these 16 minterms are distributed among the levels of the circuit evenly and progressively. Also, it is assumed that the initial number of levels is 4 since there are 4 columns in the search matrix. Therefore, a good



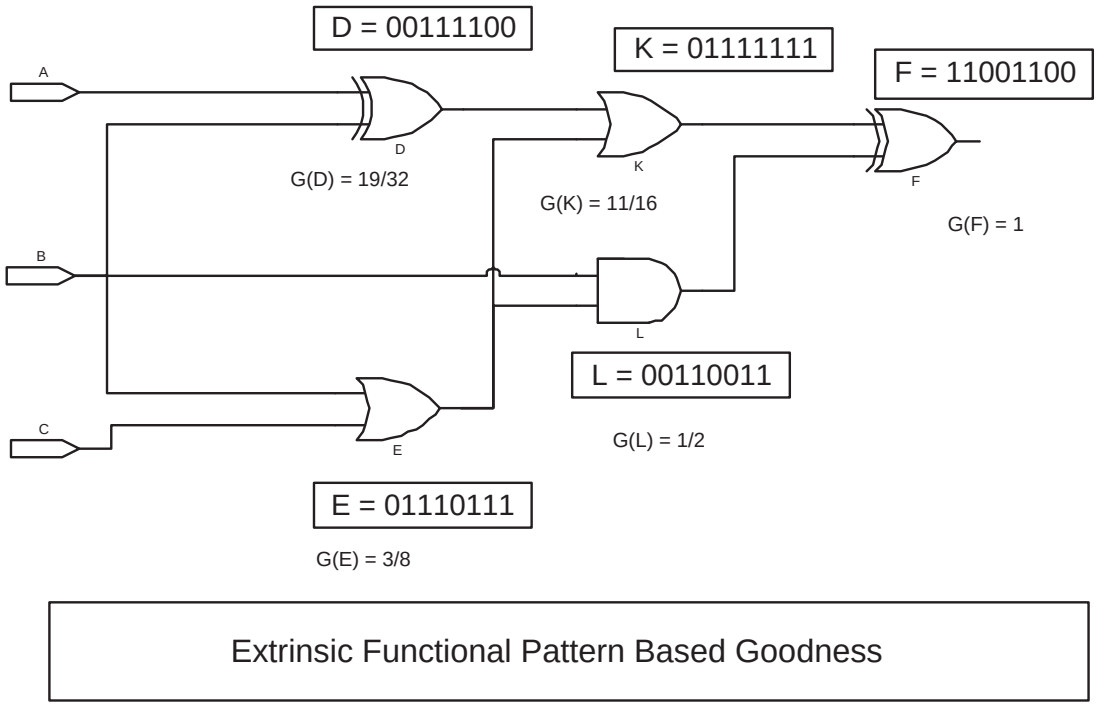


Figure 6.2: An example on the pattern based goodness measure.

logic gate located at the second level should preferably cover at most 8 minterms while a good logic gate located at the first level should preferably cover at most 4 minterms. In general, if a gate at level i and the required logic circuit has n inputs (2^n minterms), then to have a goodness of 1 at the output of that gate, there should be $\lceil (2^n/n)i \rceil$ correct minterms at the output of that gate. Then, the *multilevel logic goodness measure* is formulated as follows:

$$g_i = \frac{\rho}{\lceil 2^n/n \rceil j}$$

where g_i is the *goodness* of cell i, j is the level number or column number, n is the number of inputs of the required circuit and ρ is the number matching minterms at the output of the cell i compared to the required logic function. The value of g_i should be between 0 and 1 [0,1]. There are several scenarios resulting from this assumption and these are:

- If a cell i at level j produces more than $((2^n/n)j)$.
- If no gate at cell i located at level j produces $((2^n/n)j)$ or no gate at level n produces 2^n minterms.
- Intrinsic and Extrinsic complication.

Each of the above items will be addressed individually.

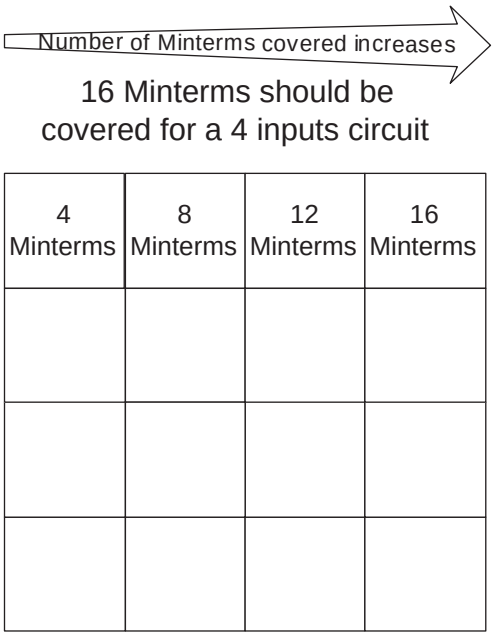


Figure 6.3: Multilevel logic goodness assumption.

First, if a cell i at level j produces more than $((2^n/n)j)$ means $\rho_i > ((2^n/n)j) \Rightarrow g_i > 1$. The number of levels that the SimE algorithm is searching at should be decreased by a factor relative to the number of minterms produced by the cell i. The new number of levels is $l = \lceil (2^n/\rho_i) \rceil$. Also, the ranges of number of minterms at each level will change accordingly. At level j of cell i, the new maximum number of minterms should be covered is $\rho_i + 1$. At $j + 1$ level, the new maximum number of minterms is

$$(\rho_i + 2) + \lceil \frac{2^n - (\rho_i + 1)}{l} \rceil$$

Then, the maximum number of minterms at level $i > j$ is

$$((\rho_i + 2) + \lceil \frac{2^n - (\rho_i + 1)}{l} \rceil (i - j)$$

Eventually, our goodness measure also should be changed and it is

$$g_i = \rho / m$$

where m is the new maximum number of minterms that gate i should get at level j.

On the other hand, if no gate at cell i located at level j can produce $((2^n/n)j)$ or no gate at level n can produce 2^n minterms which is the second case. This case means that the number of possible levels at which the SimE algorithm is searching is not enough. This number of levels has to be increased unlike the previous case where the number of levels has to be decreased. This will increase the probability of getting a correct solution for the required function at the expense of increase in the size of the search space and computation time, but there is no free pie. However, this step should not be taken unless a certain number of trials to get a solution using the initial number of levels. This number of trial is taken to be Wn, where W is the number of trial moves discussed in allocation and n is number of columns of the matrix (number of inputs). The choice of Wn is to allow the SimE algorithm to do an approximate of W trial moves at each of the n columns before going to a larger

search space. Also, the ranges of the maximum number of minterms will change if the number of levels increased. The increase in number of levels is done gradually, i.e., one by one. If l is the new number of levels, then the ranges of the number of minterms for each level is $\lfloor 2^n/l \rfloor$. Therefore, the *goodness* measure for cell i located at level j will change as follows:

$$g_i = \frac{\rho}{\lfloor 2^n/l \rfloor j} \quad \text{if } j < l$$

$$g_i = \frac{\rho}{2^n} \quad \text{if } j = l$$

where l is the new number of levels.

The third complication is similar to the one in the first goodness measure, which is, a cell i with high goodness might have its input cells to have low goodness. The same approach as in the previous one in computing the intrinsic and extrinsic goodnesses is considered.

Similarly, in order to solve the above case, *multilevel logic based goodness measure* is divided into two: *intrinsic goodness measure* denoted by g_i and *extrinsic goodness measure* denoted by G_i. The *intrinsic goodness measure* is related to the goodness of cell i. The *extrinsic goodness measure* is related to the effect on other cells inputs of cell i.

In this case the *intrinsic goodness measure* is computed by using the multilevel logic based goodness measure equations for g_i.

An example on the usage of the previous goodness measure is shown in Figure 6.4 and Figure 6.5. In this example, the required function F is 11001100. After the first run, the number of levels changed from 3 to 2. Also, after several runs, the heuristic will find that it can be implemented using one level only.

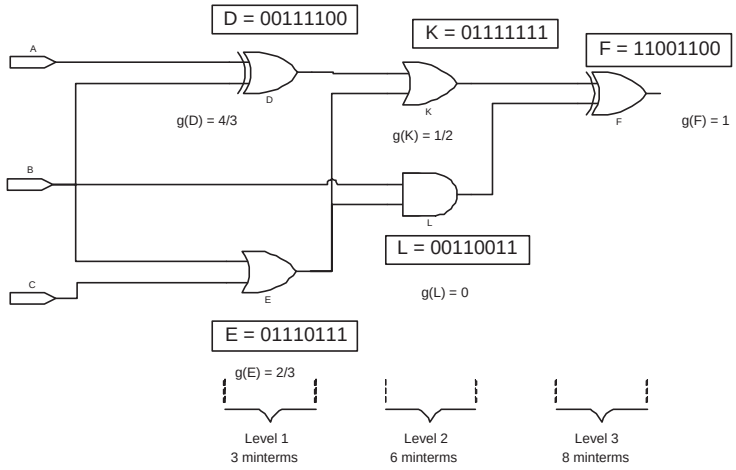


Figure 6.4: An example on the multilevel goodness measure first step.

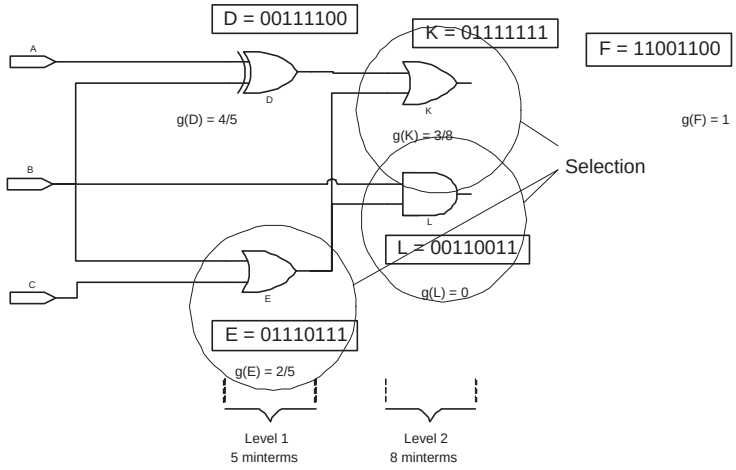


Figure 6.5: An example on the multilevel goodness measure after adjusting the number of columns.

Both *Selection Function* and *Allocation Function* will use this *goodness measure* in order to guide the algorithm to an optimal circuit. The selection function considers the extrinsic goodness measure while the allocation function considers the intrinsic goodness measure. This goodness measure have been used to implement a version of the SimE algorithm denoted by SimE-G2.

The results for both versions of the SimE algorithm will be discussed in the following chapter.

6.2 Optimization Goodness Measure

In this part, the inclusion of power, delay and area in the objective cost function will be presented. The computation of each of the above measures will be presented individually. In order to include the above objectives, a working circuit has to be obtained. This means that the *functional goodness measure* should be equal to 1. Whenever the functional goodness measure is 1, it is possible to optimize the circuit for other measures.

In order to optimize the circuit for other measures such as power, delay and area, a *global optimization goodness cost function* is proposed. This is done during the evaluation step and after allocation when the algorithm measures the improvement of global cost function to make the new allocation as permanent. It is denoted by G_O and it is computed as in Figure 6.6. The cost functions for optimizing area,

delay and power are explained next.

Global Optimization Goodness Cost Function
/* g_i: Intrinsic Functional Goodness of cell i */
/* G_O: Global Optimization Goodness of P */
/* P: the current population (solution) */
/* k: number of objectives for optimization */
/* g_j: the goodness of the solution considering the j^{th} objective [0,1]*/
/* $g_j = \frac{1}{1+\delta}$ */
/* $\delta = \frac{N}{N_{max}}$ */
/* N is the current cost of the objective function (power, area or delay) */
/* N_{max} is the maximum acceptable value for the given constraint */
/* W_i: a wight for functional goodness */
/* W_j: a wight for optimization goodness */
If $g_i < 1$ **then**
$G_O = W_i g_i$ $[0, W_i]$
ElseIf $g_i = 1$ **then**
$G_O = W_i g_i + \sum_{j=1}^{k} W_j g_j$ $[W_i,1]$
EndIf

Figure 6.6: Global optimization goodness cost function.

6.2.1 Area Estimation

The size of a VLSI circuit consists of the area for logic gates (blocks) and the interconnection wires. With the advanced technology used for implementing VLSI circuits, the size of transistors become smaller and smaller. Thus, the area requirement for interconnection wire becomes significant. However, the length of interconnection wires in VLSI circuits is determined by routing algorithms. Therefore, in this thesis, only area from logic gates is used to estimate the overall size of the circuit. The size

of these gates is obtained from a VLSI design library. Formally, the cost for area of VLSI circuits can be stated as follows.

$$Cost_{area} = \sum_{i \in G,\ i \neq WIRES} A(g_i) \tag{6.1}$$

Where $A(g_i)$ is the area of gate g_i and $g_i \in G$.

6.2.2 Delay Estimation

The overall performance of a VLSI circuit depends upon how fast it can process signals, i.e., its clock speed. The propagation delay of signals in VLSI circuits consists of two elements: switching delay of gates and interconnect delay. Due to improved technology, libraries with considerably low switching delay are available. This fact and the increased gate density on the chip make the interconnect delay a prominent factor in the overall circuit delay.

If a path π consists of nets $\{v_1, v_2, ..., v_n\}$, then, the delay T_π along π is expressed by the following Equation.

$$T_\pi = \sum_{i=1}^{n-1} (CD_i + ID_i) \tag{6.2}$$

Where CD_i is the switching delay of the cell driving gate v_i and ID_i is the interconnect delay of net v_i.

Using the RC delay model, ID_i depends on the load factor, interconnect resistance and load capacitance, as shown in Equation 6.3.

$$ID_i = (LF_i + R_i) \times C_i \tag{6.3}$$

Where LF_i is the load factor of the driving block (which is independent of the layout), R_i is the interconnect resistance of net v_i, and C_i is the load capacitance of cell i.

The load capacitance C_i of gate i comprises the interconnect capacitance at the output node of gate i and the sum of the capacitances of the input nodes of the gates driven by gate i.

$$C_i = C_i^r + \sum_{j \in M_i} C_j^g \tag{6.4}$$

Where C_j^g is the capacitance of the input node of a gate j driven by gate i and C_i^r represents the interconnect capacitance at the output node of cell i.

The overall circuit delay is determined by the delay along the longest path (the most critical path) in the layout. If the most critical path is denoted by π_c then, the cost function for the circuit delay can be given as follows.

$$Cost_{delay} = T_{\pi_c} = \underset{j}{max}\{T_{\pi_j}\} \quad \forall j \in \{1, 2, \ldots, K\} \tag{6.5}$$

Where K represents the total number of critical paths determined by a timing

analysis program.

6.2.3 Power Consumption Estimation

In standard CMOS circuit, the total power consumption can be given by the following Equation:

$$P_t = \sum_{i \in M} (\frac{1}{2} \cdot C_i \cdot V_{DD}^2 \cdot f \cdot S_i \cdot \beta) + \sum_{i \in V} Q_{SC_i} \cdot V_{DD} \cdot f \cdot S_i + I_{leak} \cdot V_{DD} \qquad (6.6)$$

In Equation 6.6, P_t is the total power consumption, V_{DD} is the supply voltage, S_i is the switching probability at the output node of cell i, i.e., the number of transitions per clock cycle at the output of gate i and f is the clock frequency.

The first term in the above equation gives the dynamic power consumption during charging or discharging of a node in the circuit. Here, M is the set of all nodes in the circuit, C_i denotes the total capacitance of node i whereas β is a technology dependent constant. The second term in Equation 6.6 gives the power consumption due to the short circuit current. Here, V is the set of all wires connecting V_{DD} to ground during output transition, Q_{SC_i} represents the charge carried by the short circuit current per transition. The third term represents the static power dissipation due to leakage current I_{leak}.

In VLSI circuits with well designed logic gates, the dynamic power consumption contributes about 90% to the total power consumption [12]. Hence, most of the

reported work is focused on minimizing the dynamic power consumption. Also, in case of standard-cell placement, the cells are obtained from the technology library and nothing can be done to reduce the power consumption due to the second and the third term in Equation 6.6. Due to this fact, the emphasis in this work is on optimizing the dynamic power consumption. Since the first term is dominant, Equation 6.6 can be approximated as follows.

$$P_t \simeq \sum_{i \in M} \frac{1}{2} \cdot C_i \cdot V_{DD}^2 \cdot f \cdot S_i \cdot \beta \tag{6.7}$$

Assuming the clock frequency and input voltage to be fixed, the total power consumption of the circuit becomes a function of the total capacitance and the switching probabilities as shown below.

$$P_t \simeq \sum_{i \in M} C_i \cdot S_i \tag{6.8}$$

Thus, the cost of the overall power consumption in VLSI circuits can be approximated as follows.

$$Cost_{power} = \sum_{i \in M} S_i \cdot C_i \tag{6.9}$$

6.3 Weighted Sum Fitness Function Calculation

6.3.1 Functional Fitness

Several functional fitness formulations are reported in the literature [33]. The commonly used one is the ratio of the number of correct hits to the length of the truth table. If FF denotes the functional fitness, then the formulation below is applied.

$$FF = \frac{Number\ of\ hits}{Length\ of\ the\ truth\ table} \tag{6.10}$$

The solution has to be 'inverted' if the value of FF is less than 0.5. Therefore, the formulation of *normalized* FF (FF_n) below is applied.

$$FF_n = Max\{FF, 1 - FF\} \tag{6.11}$$

6.3.2 Objective Fitness

The objective fitness ($OF(i)$) is a measure of the quality of solution in terms of optimization objectives such as area, delay, and power consumption. It consider two aspects: constraints satisfaction and multi-objective optimization. In order to indicate whether a solution is satisfying a certain constraint, objective fitness is

formulated as follow.

$$OF(i) = \frac{Cost(i)}{Cost(i) + Constraint(i)} \tag{6.12}$$

For example, objective fitness of the solution in terms of area is:

$$OF(area) = \frac{Cost(area)}{Cost(area) + Constraint(area)} \tag{6.13}$$

With this formulation, a circuit satisfying the constraint in terms of area will have $OF(area)$ greater than or equal to 0.5. Any solution that has $OF(area)$ less than 0.5 will not be considered. The constraint values are given by the user. It states the upper bound for specific optimization objectives. Since there are four attributes to optimize, there is objective fitness for each of these attributes. These are computed using Equation 6.12.

The weights assigned to each attributes, w_i, indicate the emphasis of the optimization process. For example, for area optimization, w_{area} can be set equal to three while other weights are set to one, each. It is also possible to have more than one optimization objectives. For example, if we want to build a circuit with less area and less delay, w_{area} and w_{delay} can be set equal to k while w_{gc} and w_{power} are set equal to l, $k > l, k > 1, l \geq 0$. Note that some other weighting scheme can be applied.

The weighted sum objective fitness function calculation can be expressed as follows.

$$OF = \frac{\sum_{i \in obj} W_i \cdot OF(i)}{\sum_{i \in obj} W_i} \tag{6.14}$$

Where *obj* represents the set of optimization objectives.

6.4 Fuzzy Fitness Function Calculation

In this section a fuzzy-based fitness function is formulated. Similar to the weighted sum approach, the overall fitness of a solution consists of two parts: functional fitness and objective fitness. In this approach membership functions are used and these membership functions will be aggregated into a single function using a fuzzy operator.

6.4.1 Functional Fitness

Using Equation 6.11, the value of functional fitness lies in the range [0.5, 1]. Thus the membership function for functional fitness is trivial. It is shown in Equation 6.15.

$$\mu_{func}(x) = \begin{cases} x & if\ 0.5 \leq x \leq 1 \\ 0 & otherwise \end{cases} \tag{6.15}$$

6.4.2 Objective Fitness

In order to build the membership function for all objectives, estimated value for lower bound and/or upper bound of the objective is required.

Each characteristic of the circuit (area, delay and power consumption) can be used either as constraint or objective. The membership function for each case (objective or constraint) will be different. This will be discussed next.

Area as Optimization Objective

The lower bound on area can be estimated by referring to the VLSI circuit design and logic synthesis principles. For any n-input single-output circuit, the minimum area for the given circuit is equal to the area of $n-1$ 2-input gates representing binary tree structure. Since any circuit can be implemented using NAND gates and NAND gates happen to be the smallest among other gate (except NOT gate), then the minimum area is:

$$min_{area} = (n-1) \times Area(NAND\ gate)$$

In order to guide the search intelligently, the maximum value must be carefully estimated. For this purpose, SIS tools [36] are used to obtain circuits with minimum area. In this context, *rugged.script* is used to generate the circuits' netlist files. These files are then fed to our own tool to obtain the estimated value for area, delay and

power consumption. The reason behind this is twofold. Firstly because the delay optimization in SIS does not consider switching delay (see Equation 6.3). Secondly, SIS does not consider power optimization.

Since we want to obtain circuits better than SIS, these values (area, delay, and power) are used as the target values. In the case of area as optimization objectives, the target area is equal to the area of circuits obtained by SIS and denoted as tg_{area1} (see Figure 6.7). Thus, the membership function for area as optimization objectives is:

$$\mu_{area\_obj} = \begin{cases} 1 & 0 \leq area \leq min_{area} \\ 1 - \frac{(area - min_{area})}{tg_{area1} - min_{area}} & min_{area} \leq area \leq tg_{area1} \\ 0 & otherwise \end{cases} \tag{6.16}$$

The shape of the membership function is depicted as the bold line shown in Figure 6.7.

Area as Constraint

In this case, the area of circuit obtained from SIS is used as target value. For this purpose, the max_{area} and tg_{area2} should be defined. The following settings are applied, $tg_{area2} = k_1 \times tg_{area1}$ and $max_{area} = k_2 \times tg_{area1}$, $k_1,\ k_2 \in \Re,\ 0 < k_1 \leq$

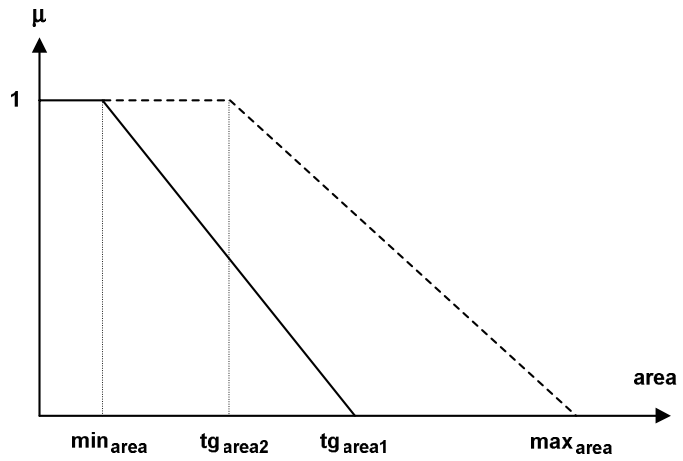


Figure 6.7: Membership function for area.

1, $k_2 \geq 1$. The membership function is then:

$$\mu_{area\_con} = \begin{cases} 1 & 0 \leq area \leq tg_{area1} \\ 1 - \frac{area - k_1}{max_{area} - k_1} & 1 \leq area \leq max_{area} \\ 0 & otherwise \end{cases} \tag{6.17}$$

The shape of the membership function is depicted as dashed line shown in Figure 6.7.

Delay as Optimization Objective

The minimum delay (min_{delay}) is estimated as the delay of two-level logic consisting of NAND gates without considering the switching delay. The tg_{delay1} is estimated from circuit generated by SIS with *delay.script* executed. The membership function

for delay as optimization objectives is:

$$\mu_{delay\_obj} = \begin{cases} 1 & 0 \leq delay \leq min_{delay} \\ 1 - \frac{delay - min_{delay}}{tg_{delay1} - min_{delay}} & min_{delay} \leq delay \leq tg_{delay1} \\ 0 & otherwise \end{cases} \quad (6.18)$$

The shape of the membership function is depicted as bold line shown in Figure 6.8.

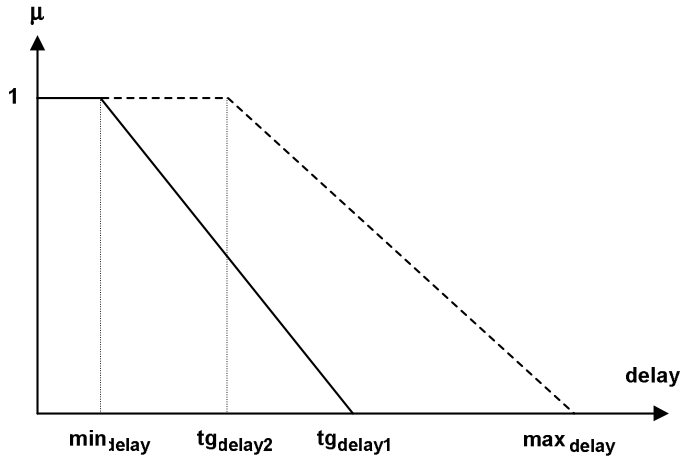


Figure 6.8: Membership function for delay.

Delay as Constraint

In this case, the following settings are applied, $tg_{delay2} = k_1 \times tg_{delay1}$ and $max_{delay} = k_2 \times tg_{delay1}$, $k_1,\ k_2 \in \Re,\ 0 < k_1 \leq 1,\ k_2 \geq 1$. The membership

function is then

$$\mu_{delay\_con} = \begin{cases} 1 & 0 \leq delay \leq tg_{delay1} \\ 1 - \frac{delay - k_1}{max_{delay} - k_1} & 1 \leq delay \leq max_{delay} \\ 0 & otherwise \end{cases} \tag{6.19}$$

The shape of the membership function is depicted as dashed line shown in Figure 6.8.

Power as Optimization Objective

The minimum power (min_{power}) is estimated as the power consumption of minimum area circuit in which each gate has the least switching activity (see Equation 6.6). It is assumed that for a given truth table, the output of each gate will be '1' only once. Thus the minimum power consumption (switching activity) can be estimated as follows.

$$min_{power} = 2 \cdot \frac{length\ of\ truth\ table - 1}{(length\ of\ truth\ table)^2} \cdot capacitance(NAND)$$

The tg_{power1} is estimated from minimum area circuit generated by SIS. The

membership function for power as optimization objectives is:

$$\mu_{power\_obj} = \begin{cases} 1 & 0 \leq power \leq min_{power} \\ 1 - \frac{power - min_{power}}{tg_{power1} - min_{power}} & min_{power} \leq power \leq tg_{power1} \\ 0 & otherwise \end{cases} \quad (6.20)$$

The shape of the membership function is depicted as bold line shown in Figure 6.9.

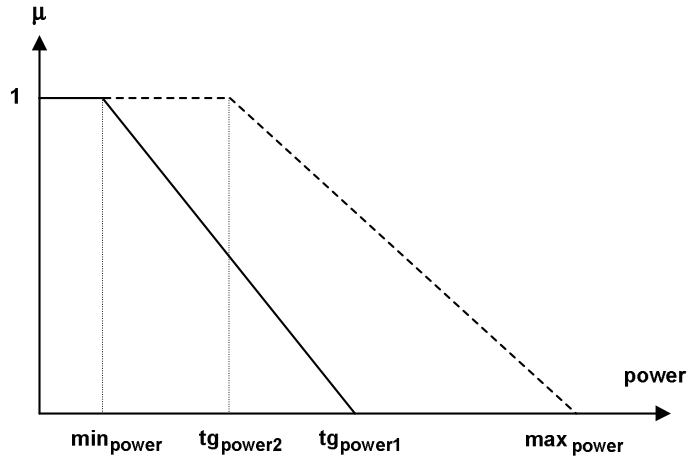


Figure 6.9: Membership function for power.

Power as Constraint

The following settings are applied, $tg_{power2} = k_1 \times tg_{area1}$ and $max_{power} = k_2 \times tg_{power1}$, $k_1,\ k_2 \in \Re,\ 0 < k_1 \leq 1,\ k_2 \geq 1$. The membership function is:

$$\mu_{power\_con} = \begin{cases} 1 & 0 \leq power \leq tg_{power1} \\ 1 - \frac{power - k_1}{max_{power} - k_1} & 1 \leq power \leq max_{power} \\ 0 & otherwise \end{cases} \tag{6.21}$$

The shape of the membership function is depicted as dashed line shown in Figure 6.9.

OWA operator is used to aggregate the above membership functions to calculate the objective fitness. The choice of which type of membership function used (either objective or constraint) depends on the course of the current action. For example, for minimization of area with delay and power as constraint, the μ_{area_obj}, μ_{delay_con} and μ_{power_con} are considered. The objective fitness is then calculated as

$$\begin{aligned} \mu_{obj}(x) &= \beta \times \min(\mu_{area\_obj}, \mu_{delay\_con}, \mu_{power\_con}) \\ &\quad +(1-\beta) \times \tfrac{1}{3}(\mu_{area\_obj} + \mu_{delay\_con} + \mu_{power\_con}) \end{aligned} \tag{6.22}$$

6.5 Concluding Remarks

In this chapter, the cost function formulation for the proposed algorithm was discussed. Two goodness measures were proposed for the SimE algorithm. These goodness measures were discussed in detail. These goodness measures have a major role on the performance of the SimE algorithm. Moreover, the use of weighted sum and fuzzy logic for targeting multi-objective functions were presented.

Chapter 7

Experiments and Results

Chapters 7 and 8 are dedicated for experiments and results of the solution approach. This chapter concentrates on the performance evaluation of different possible scheme, while next chapter will have some comparison with the existing techniques.

7.1 Experimental Setup

The following is the setup that is needed for conducting the experiments. This includes the setting of inputs, tools and parameters.

Input: In order to perform the experiment, some benchmark circuits are used as test cases. The benchmark consists of twenty randomly generated truth tables of different complexity in addition to twelve circuits obtained from ISCAS'85 benchmark.

All circuits are in PLA format. Detail of these circuits is given in the appendix.

Technology parameters used in the experiments is obtained from MOSIS .25 μ library [1]. These parameters are written into a text file with a certain format. This can be seen in the appendix as well. Table 7.1 shows a summary of these circuits.

Circuits used	Single Output		Multiple Output		Notes
	Number of circuits	Specification	Number of circuits	Specification	
Random	20	Number of inputs: 2 - 6	-	-	-
ISCAS'85	3	Number of inputs: 3 - 7	9	Number of inputs: 3 - 7 Number of outputs: 2 - 10	arithmetic circuits: majority, parity, adder and multiplier

Table 7.1: Summary of circuits used for the experiments.

Tools: SIS tool is used for the purpose of comparison with the existing tools.

Parameters for the Algorithm: Performance of iterative heuristics depends on the fine-tuning of its parameters. If it is not mentioned explicitly, these parameters are used as general parameters for the experiments. Note that n is the number of variables for a specific circuit. More details about the initial values of the solution approach can be found in Chapter 5.

1. Parameters for SimE algorithm

   - Selection Bias $B = 0.0$
   - The number of alteration trial $W = m\frac{n!}{2!(n-2)!}$
   - Population Size = 1

- Number of iteration (stopping criteria) = 250 * n

2. Parameters for the solution approach

- Number of runs = 30
- Minimum size of the matrix = $n \times n$
- Maximum size of the matrix = $(4 \cdot n) \times (3 \cdot n)$

Three types of experiments are carried out to measure the performance of the solution approach. These include experiments for different goodness measures, effect of hybrid SimE on the quality of solution and effect of different optimization objectives.

7.2 Performance of Different Goodness Measures

The objective of this experiment is to know which goodness measures works best for the solution approach.

From the previous chapter, we have two proposed goodness measures, namely, the pattern based goodness measure and the multilevel based goodness measure. To evaluate the two goodness measures, we need to look at how the fitness function will behave during the execution time and how the quality of the solution will be in order to compare them. In the following case, area and power were objectives and delay a constraint.

In order to compare both goodness measures, the behavior of the average fitness

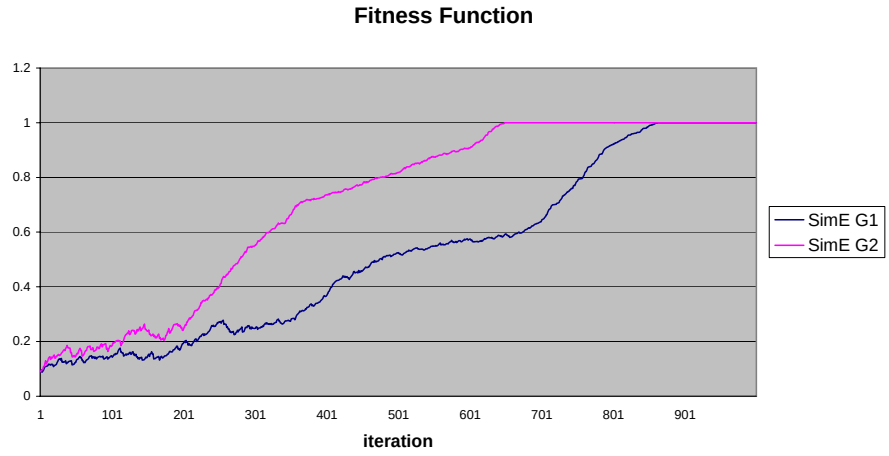


Figure 7.1: Fitness function for SimE-G1 and SimE-G2 for 4 inputs circuit (circuit2).

function in the SimE algorithm using each goodness measure is compared. The averages of 30 runs of a 4 inputs circuit (circuit2) and mul3 circuit (a large circuit) are shown in Figure 7.1 and 7.2. The behavior of the fitness measures in Figure 7.1 shows that the SimE-G2 is better than SimE-G1 slightly in time since the fitness function for SimE-G2 converges to 1 earlier than SimE-G1 fitness function. In fact, this is due to the ability of the multilevel goodness measure to resize the matrix solution dynamically. However, the difference between SimE-G1 and SimE-G2 can be considered minor for smaller circuits due to the randomness of the SimE algorithm which might not guarantee the results to be consistent. The behavior of the fitness measures in Figure 7.2 indicates that the SimE-G2 is better than SimE-G1 by 50% in time since the fitness function for SimE-G2 converges to 1 earlier than SimE-G1 fitness function. This behavior was noticed clearly in all large circuits where the size of the search space has a major effect on the performance of the algorithm.

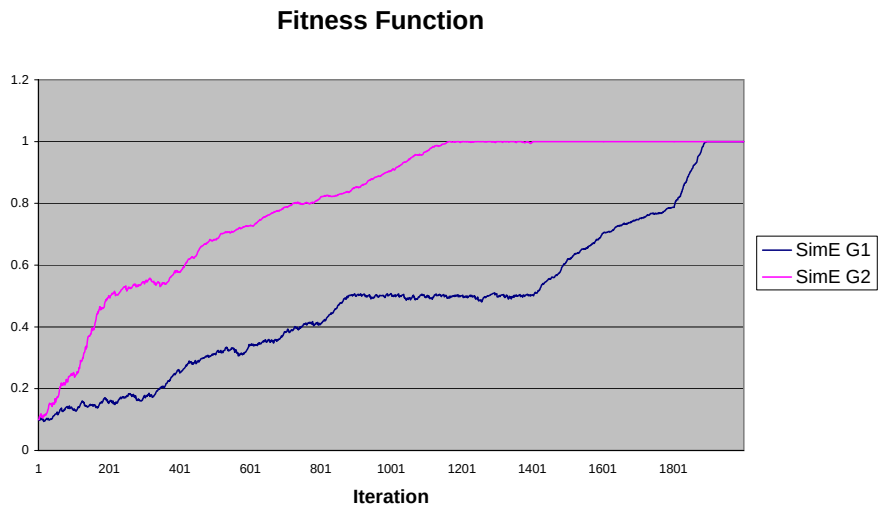


Figure 7.2: Fitness function for SimE-G1 and SimE-G2 for mul3.

In addition, the quality of the solution generated by using both goodness measures is compared. The results of the experiments for some selected circuits is shown in Table 7.2. Notice that the experiments conducted were area and power as objectives and delay as constraint. In most of the cases, the average area and power of the SimE-G2 is better than the area and power of SimE-G1. Therefore, the quality of the solution using SimE-G2 is better than the quality of the solution of SimE-G1.

A graphical representation of the normalized area and power for SimE-G2 to the area and power of SimE-G1 is shown in Figure 7.3. Notice that 1 in the graph is the area and power of SimE-G1.

Table 7.3 shows the improvements in execution time of SimE-G2 compared to SimE-G1 for the same circuits. In all cases, SimE-G2 has less execution time than SimE-G1. The modification of matrix size dynamically results in changing the size of the search space. In most of the cases, the size of the matrix is reduced which

Circuit	SimE-G1			SimE-G2			% Improvement	
	area	delay	power	area	delay	power	% area	% power
circuit1	13290.40	4.08	5.20	12910.42	3.87	4.99	2.94	4.21
circuit2	13170.54	5.23	5.37	13170.76	5.17	5.29	0.00	1.51
circuit3	10822.45	3.82	3.89	10783.84	3.79	3.73	0.36	4.29
circuit4	1458.00	0.005	0.66	1458.00	0.005	0.66	0.00	0.00
circuit5	13977.62	3.99	5.91	13903.92	3.87	5.85	0.53	1.03
circuit6	11023.00	3.54	3.91	10988.36	3.33	3.84	0.32	1.82
circuit7	12452.87	5.62	4.57	12400.26	5.10	4.39	0.42	4.10
circuit8	7776.00	5.22	2.73	7776.00	5.22	2.73	0.00	0.00
mul2	14229.69	4.23	4.76	13887.54	4.60	4.36	2.46	9.17
mul3	77230.56	15.12	22.77	74456.42	13.00	21.66	3.73	5.12
cm42a	41092.17	8.91	14.01	40982.44	8.89	13.67	0.27	2.49
cm82a	25489.41	12.82	10.64	25109.93	11.90	9.27	1.51	14.78
b1	12264.93	3.44	3.07	11342.86	2.90	2.79	8.13	10.04
c17	11439.26	5.82	3.51	11008.74	5.24	3.41	3.91	2.93
con1	30709.48	7.03	14.58	30351.12	6.92	14.24	1.18	2.39
majority	13976.50	4.67	5.22	13908.33	4.55	5.08	0.49	2.76

Table 7.2: Results comparison between SimE-G1 and SimE-G2.

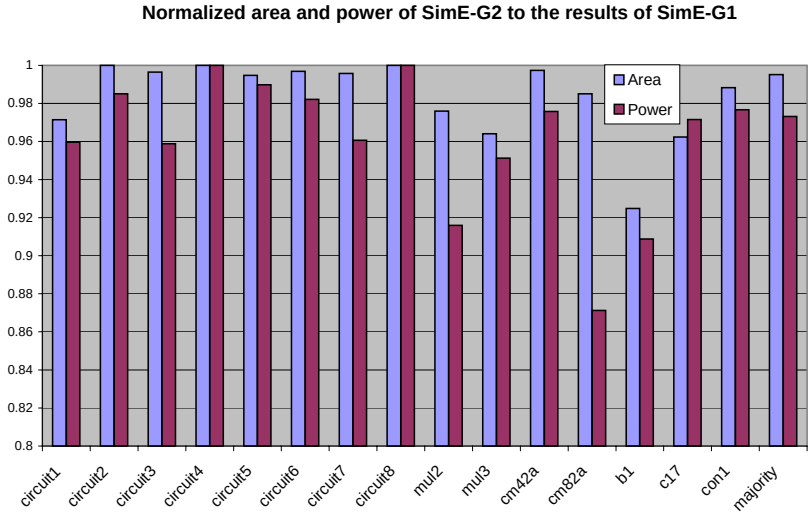


Figure 7.3: Normalized area and power of SimE-G2 to the area and power of SimE-G1.

Circuit	SimE-G1 Time (s)	SimE-G2 Time (s)	% Improvement
circuit1	5.68	3.12	45.07
circuit2	4.67	3.77	19.27
circuit3	6.55	4.57	30.23
circuit4	1.20	0.48	60.00
circuit5	17.34	6.51	62.46
circuit6	12.65	6.02	52.41
circuit7	13.62	7.44	45.37
circuit8	12.51	5.02	59.87
mul2	55.21	25.72	53.41
mul3	304.20	150.40	50.56
cm42a	239.10	123.68	48.27
cm82a	70.25	35.81	49.02
b1	56.96	25.93	54.48
c17	82.09	42.11	48.70
con1	459.51	160.00	65.18
majority	35.87	15.32	57.29

Table 7.3: Improvements in execution time in SimE-G2 over SimE-G1.

results in reducing the size of the search space. As a result, less time is required to arrive at a better solution.

7.3 Effect of Hybrid SimE on the Quality of Solution

In Chapter 6, the use of Tabu search in digital logic design was introduced in order to compare it with SimE since both algorithms belong to the same family of non-deterministic iterative algorithm. Also, the hybrid SimE where Tabu Search is incorporated into SimE is discussed in Chapter 6. In this section, the effect of incorporating Tabu Search into SimE for the logic design is analyzed.

Hybrid SimE is compared to the original proposed SimE algorithm in terms of execution time and quality of solution. The second goodness measure, which is the multilevel based goodness measure, is used with Tabu Search in order to implement the hybrid SimE. The choice of the second goodness measure is due to the previous results presented which shows that SimE-G2 is better than SimE-G1 in execution time and quality of solution. The experiments conducted on the circuits are for area and power optimization as objectives and delay as constraints.

The results of the experiments of some selected circuits are shown in Table 7.4 and Table 7.5.

Circuit	SimE-G1			Hybrid SimE-G2			% Improvement	
	area	delay	power	area	delay	power	% area	% power
circuit5	13903.92	3.87	5.85	6.51	13870.00	3.90	0.24	0.86
circuit6	10988.36	3.33	3.84	6.02	10935.00	3.34	0.49	0.58
circuit7	12400.26	5.10	4.39	7.44	12393.00	5.05	0.06	0.23
circuit8	7776.00	5.22	2.73	5.02	7776.00	5.22	0.00	0.00
circuit9	14988.65	5.19	4.92	8.22	14970.00	5.20	0.12	0.20
mul2	13887.54	4.60	4.36	25.72	13650.00	4.67	1.74	0.46
mul3	74456.42	13.00	21.66	150.40	74377.39	13.12	0.11	0.05
cm42a	40982.44	8.89	13.67	123.68	40866.43	8.87	0.28	0.15
cm82a	25109.93	11.90	9.27	35.81	25055.81	11.86	0.22	0.22
b1	11342.86	2.90	2.79	25.93	11206.00	2.91	1.22	0.36

Table 7.4: Results comparison between SimE-G2 and Hybrid SimE-G2.

Table 7.4 shows that the Hybrid SimE-G2 has better quality of solution than SimE-G2. The Hybrid SimE-G2 were able to converge to better solutions because of the local search done by the tabu algorithm. Tabu Search will intensify the search resulting in better solutions by searching the neighborhood of the current solution.

Circuit	SimE-G2 Time (s)	Hybrid SimE-G2 Time (s)	% Improvement
circuit5	5.80	7.01	-17.26
circuit6	3.82	6.13	-37.72
circuit7	4.38	7.51	-41.68
circuit8	2.73	5.05	-45.94
circuit9	4.91	8.98	-45.32
mul2	4.34	27.62	-84.29
mul3	21.65	160.36	-86.50
cm42a	13.65	127.92	-89.33
cm82a	9.25	37.95	-75.63
b1	2.78	28.36	-90.20

Table 7.5: Improvements in execution time in SimE-G2 over Hybrid SimE-G2.

On the other hand, the increase in solution quality resulted in an increase in time. Also, the time consumed by the additional processing of tabu search in order to explore the neighboring solution increased the overall algorithm execution time. In multiple output circuits, tabu search will consume more time exploring neighboring solution because multiple output circuits has common subexpression, a modification in one of the cells (gate type) can disturb other outputs.

7.4 Effect of Different Optimization Objectives

The cost function considered in this work includes area, delay and power consumption of the circuits. Thus, there exist four sets of experiment:

1. Area optimization with delay and power as constraints (AODPC)

2. Delay optimization with area and power as constraints (DOAPC)

3. Power optimization with area and delay as constraints (POADC)

4. Area and power optimization with delay as constraints (APODC)

In this section, the area obtained for different circuits where area optimization with delay and power as constraints (AODPC) using SimE-G1 and SimE-G2 is compared to the area obtained in other sets of experiments (DOAPC, POADC, APODC) in order to see the effect of area optimization as objective. Table 7.6 and Table 7.7 show the area of some selected circuits using AODPC compared to the other 3 cases (DOAPC, POADC, APODC) for SimE-G1 and SimE-G2 respectively.

Circuit	AODPC	DOAPC	POADC	APODC
circuit1	14094	15360	14094	12879
circuit2	14823	16870	14823	13122
circuit3	10692	11907	10752	10752
circuit8	7290	7290	7776	7776
circuit9	17454	17689	17689	14970
b1	11206	12745	13220	11206
c17	9963	11087	10923	10923
con1	30233	30233	40632	30233
majority	13851	17821	17821	13851
xor8	20655	20655	20655	20655
xor9	23328	27216	23814	23814
rd53	38073	40201	40231	40231

Table 7.6: Area for selected circuits using SimE-G1 with the four set of experiments.

A graphical representation of the normalized area of the above selected circuits using SimE-G1 considering DOAPC, POADC and APODC to the area results of SimE-G1 considering AODPC is shown in Figure 7.4. The result of the area using SimE-G1 considering area optimization (AODPC) as 1 in the figure. It is clear that in most cases SimE-G1 considering area optimization will produce better results

Circuit	AODPC	DOAPC	POADC	APODC
circuit1	12879	15360	12879	12879
circuit2	13122	16870	13122	13122
circuit3	10692	11907	10752	10752
circuit10	9963	13771	9963	9963
circuit11	16354	23785	16354	16354
add2	24300	35270	24317	24300
add3	40265	53703	40265	40265
cm82a	25029	55872	25029	25029
b1	11206	12745	13220	11206
c17	9963	11087	10923	10923
con1	30233	30233	40632	30233
majority	13851	17821	17821	13851

Table 7.7: Area for selected circuits using SimE-G2 with the four set of experiments.

when considering other objectives. Also, it is noticed that in most of the cases optimizing for power will also produce good results in term of area. Moreover, considering area and power as objectives for optimization will help the SimE-G1 algorithm in producing better results in term of area and power compared to optimizing for area or power alone.

A graphical representation of the normalized area of the above selected circuits using SimE-G2 considering DOAPC, POADC and APODC to the area results of SimE-G2 considering AODPC is shown in Figure 7.5. Also, it is clear that in most cases SimE-G2 considering area optimization will produce better results when considering other objectives. Moreover, similar conclusion can be drawn as above considering power and area optimization can help in producing better circuits in term of area and power rather than considering area or power individually.

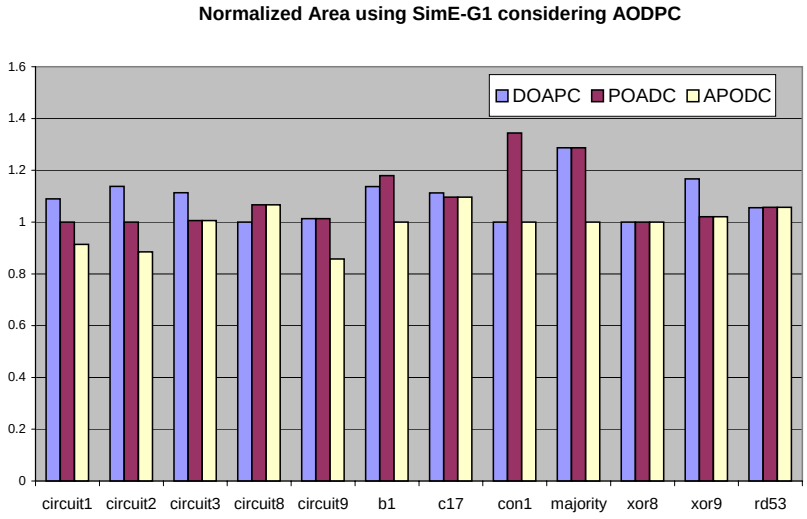


Figure 7.4: Normalized area of SimE-G1 (DOAPC, POADC, APODC) to the area of SimE-G1 considering area optimization (AODPC).

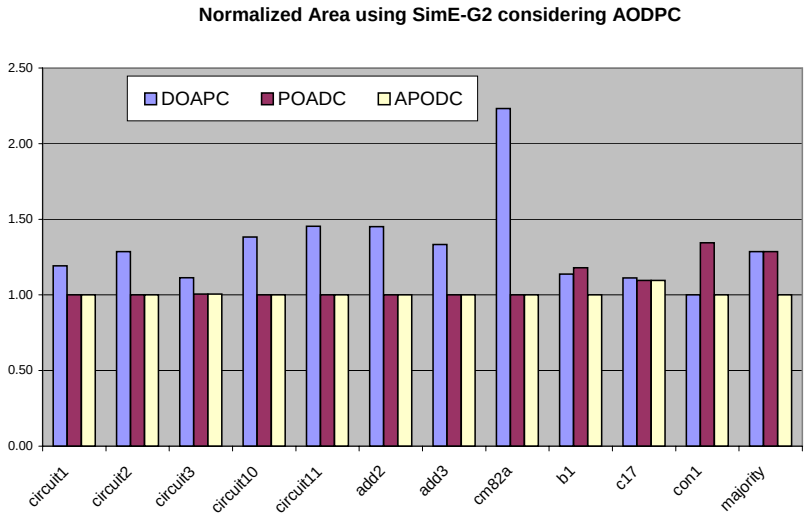


Figure 7.5: Normalized area of SimE-G2 (DOAPC, POADC, APODC) to the area of SimE-G2 considering area optimization (AODPC).

In addition, the delay obtained for different circuits where delay optimization with area and power as constraints (DOAPC) using SimE-G1 and SimE-G2 is compared to the delay obtained in other sets of experiments in order to see the effect of delay optimization as objective. Table 7.8 and Table 7.9 show the delay of some selected circuits using DOAPC compared to the other 3 cases for SimE-G1 and SimE-G2 respectively.

Circuit	DOAPC	AODPC	POADC	APODC
circuit9	5.66	6.65	5.66	5.20
circuit10	5.50	6.42	6.42	6.42
circuit11	5.50	7.07	6.40	6.40
circuit12	13.42	16.55	16.55	16.55
circuit13	8.68	8.85	8.93	9.77
add2	9.37	11.48	14.73	11.48
add3	12.98	24.99	26.73	26.73
mul2	2.96	3.56	4.67	4.67
mul3	13.14	13.14	13.14	13.14
cm42a	5.19	8.86	9.44	8.86

Table 7.8: Delay for selected circuits using SimE-G1 with the four set of experiments.

Circuit	DOAPC	AODPC	POADC	APODC
circuit5	3.90	5.57	4.89	3.90
circuit6	2.68	3.34	3.85	3.34
circuit7	5.05	5.05	5.60	5.05
circuit8	2.96	2.96	5.22	5.22
circuit10	5.50	6.42	6.42	6.42
add2	9.37	11.48	14.73	11.48
add3	12.98	26.73	26.73	26.73
mul2	2.96	3.56	4.67	4.67
mul3	13.14	13.14	13.14	13.14
cm42a	5.19	9.44	9.44	8.86

Table 7.9: Delay for selected circuits using SimE-G2 with the four set of experiments.

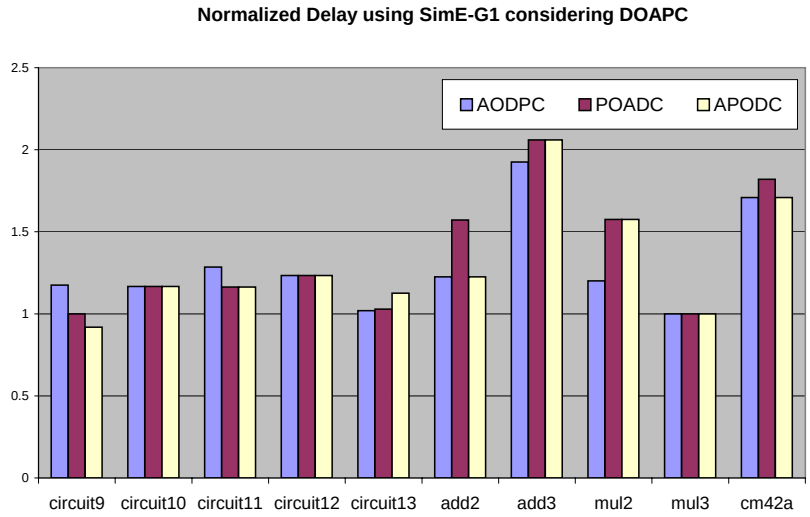


Figure 7.6: Normalized delay of SimE-G1 to the delay of SimE-G1 considering delay optimization.

A graphical representation of the normalized delay of the above selected circuits using SimE-G1 to the delay results of DOAPC is shown in Figure 7.6. In 95% of the cases, SimE-G1 considering delay optimization will produce better results compared to others. This is due to the contradiction between delay compared to area and power. In most of the cases, a decrease in the delay will increase the area and power. For example, a function implemented in two level will have less delay compared to the implementation of multilevel. However, power and area will increase due to this kind of implementation.

A graphical representation of the normalized delay of the above selected circuits using SimE-G2 to the delay results of DOAPC is shown in Figure 7.7. Similar conclusion can be drawn as above considering power and area optimization will produce circuits with high delay compared to delay optimization.

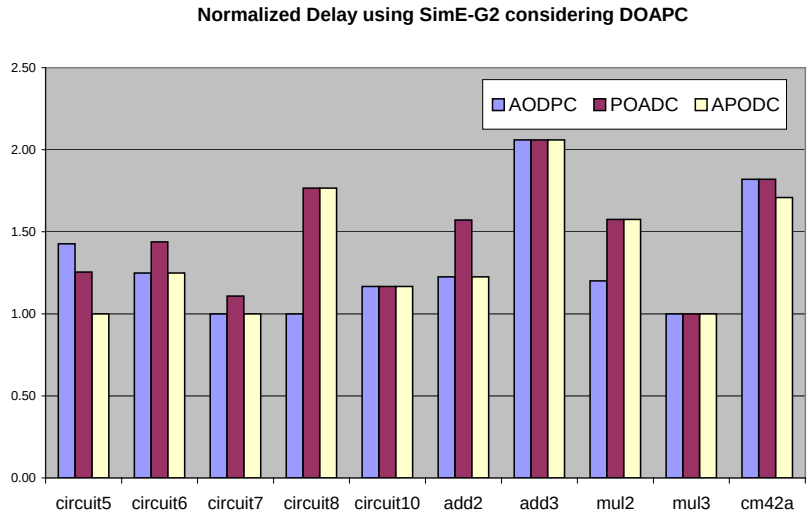


Figure 7.7: Normalized delay of SimE-G2 to the delay of SimE-G2 considering delay optimization.

Moreover, the power obtained for different circuits where power optimization with area and delay as constraints (POADC) using SimE-G1 and SimE-G2 is compared to the power obtained in other sets of experiments in order to see the effect of power optimization as objective. Table 7.10 and Table 7.11 show the power of some selected circuits using POADC compared to the other 3 cases for SimE-G1 and SimE-G2 respectively.

A graphical representation of the normalized power of the above selected circuits using SimE-G1 and SimE-G1 to the power results of POADC are shown in Figures 7.8 and 7.9 respectively.

Considering Figures 7.8 and 7.9, it can be concluded that power and area optimization are related in many cases. A circuit that has less area, will have less power consumption. However, delay is the opposite. A circuit with less delay will be high

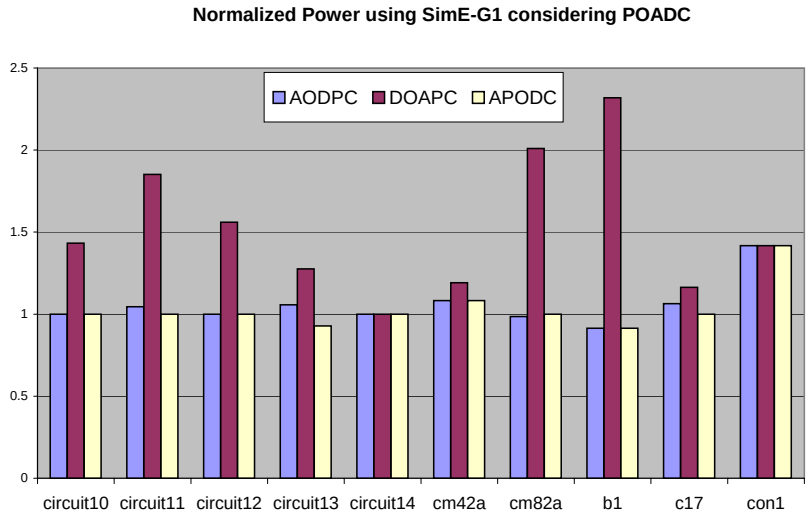


Figure 7.8: Normalized power of SimE-G1 to the power of SimE-G1 considering power optimization.

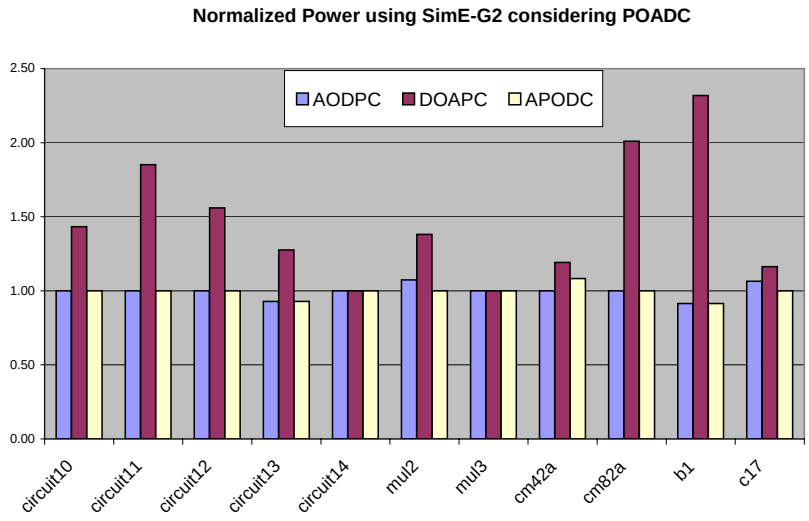


Figure 7.9: Normalized power of SimE-G2 to the power of SimE-G2 considering power optimization.

Circuit	POADC	AODPC	DOAPC	APODC
circuit10	3.33	3.33	4.77	3.33
circuit11	5.75	6.01	10.65	5.75
circuit12	18.94	18.94	29.55	18.94
circuit13	8.40	8.88	10.72	7.80
circuit14	9.20	9.20	9.20	9.20
cm42a	12.60	13.64	15.02	13.64
cm82a	9.24	9.10	18.56	9.24
b1	3.04	2.78	7.05	2.78
c17	3.42	3.64	3.98	3.42
con1	10.04	14.23	14.23	14.23

Table 7.10: Power for selected circuits using SimE-G1 with the four set of experiments.

Circuit	POADC	AODPC	DOAPC	APODC
circuit10	3.33	3.33	4.77	3.33
circuit11	5.75	5.75	10.65	5.75
circuit12	18.94	18.94	29.55	18.94
circuit13	8.40	7.80	10.72	7.80
circuit14	9.20	9.20	9.20	9.20
mul2	4.34	4.66	5.99	4.34
mul3	21.65	21.65	21.65	21.65
cm42a	12.60	12.60	15.02	13.64
cm82a	9.24	9.24	18.56	9.24
b1	3.04	2.78	7.05	2.78
c17	3.42	3.64	3.98	3.42

Table 7.11: Power for selected circuits using SimE-G2 with the four set of experiments.

in power consumption. In most of the cases, the implementation that can be got for better delay is in 2 level implementation, this results in larger area and power.

From the three sets of experiments, it was observed that the performance of DOAPC is the worst using SimE-G1 and SimE-G2. The reason behind that is SimE tries to allocate the selected cells in new location where it will give better goodness. Delay optimization should consider the worst path which is not applicable to SimE

during allocation because it does not know all possible paths. However, it can measure the delay according to the provided estimation after allocation. Therefore, optimizing for delay will be difficult unlike optimizing for area where the algorithm knows the current cost of area during allocation.

7.5 Concluding Remarks

In this chapter, results obtained from the solution approach was discussed. It started from the results of different fitness function calculation. Then, the results of different set of experiments were also presented. Comparison of the solution approach with the existing technique will be presented in the next chapter.

Chapter 8

Comparison with Existing Techniques

8.1 Comparison with Existing ELD Techniques

In this section the results of our experiments are contrasted to the results obtained from the existing technique in GA-based ELD proposed by Coello [8]. Since this technique tries to minimize the use of gate count only, the comparison is performed with the results of AODPC of the solution approach SimE-G1 and SimE-G2. This includes comparison in terms of area, delay and power. In the following, the results of GA will be compared with the average results of SimE-G1 and SimE-G2.

Table 8.1 shows the quality of solutions in terms of area, delay and power for both techniques. The table shows that except for circuit4 that contains a single gate only,

Circuit	Coello [8]			SimE-G1			SimE-G2		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
circuit1	12393.00	5.05	4.38	12393.00	5.05	4.38	12393.00	5.05	4.38
circuit2	21870.00	6.18	6.61	15451.60	5.26	6.77	13150.93	5.21	5.32
circuit3	19926.00	4.34	5.15	10843.10	3.00	4.35	10745.45	3.12	3.96
circuit4	1458.00	0.005	0.66	1458.00	0.005	0.66	1458.00	0.005	0.66
circuit5	27945.00	8.76	7.89	13412.44	6.67	6.29	11723.95	5.61	5.14
majority	21141.00	7.53	6.07	14029.51	4.88	4.41	13977.51	4.50	5.12
xor8	32805.00	9.53	11.64	20880.23	6.02	9.78	20655.00	5.90	9.32
xor9	35266.00	11.34	13.79	23814.00	9.57	10.65	23814.00	9.57	10.65

Table 8.1: Comparison with Coello [8] technique in terms of area, delay and power.

there are significant improvements in terms of area for all cases. The comparison of execution time for both techniques is provided. The experiments were carried out in P4 2GHz CPU, 512 MB RAM. Although the execution time depends on some variables such as the number of iterations, the size of the matrix and the number of gate types used, the comparison is required in order to see the performance of both techniques.

Table 8.2 shows the average execution time for the given test cases. It is obvious that Coello's approach is slower compared to SimE-G1 and SimE-G2 in time and quality of solution. Moreover, Coello's approach suffers a lot in multiple output circuits.

Circuit	Coello [7](s)	SimE-G1 (s)	SimE-G2 (s)
circuit1	91.66	12.52	10.85
circuit2	102.32	15.09	11.01
circuit3	155.78	20.14	14.07
circuit4	275.10	20.50	13.10
circuit5	266.36	21.25	15.87
majority	6290.32	224.71	119.62
xor8	7430.01	221.10	120.34
xor9	10856.55	320.92	273.63

Table 8.2: Comparison with Coello's GA algorithm in terms of execution time [7].

8.2 Comparison with Existing Conventional Techniques

In this section, the comparison of the proposed algorithm with existing conventional technique is given. For this purpose, SIS tools [36] were required to provide the results. However, since SIS does not consider capacitance load in their delay calculation and does not do power optimization, the result from SIS was the netlists for optimized area and delay circuits. These netlists were later given to the cost function calculation procedure of the solution approach in order to generate the area, delay and power.

Area Optimization:

In order to compare the performance of solution approach for area optimization, the AODPC experiment was used. The results from SIS were the area optimized circuits using *rugged.script*, mapped for area minimization. SimE and SIS use the

same gate library.

Table 8.3 shows the results for single output circuits for SimE-G1 and SIS. The table shows SimE-G1 provides circuits with less area as compared to SIS. The highest improvements were observed for the circuit5. Figure 8.1 shows the graphical view of the results from SimE-G1. Note that these results are normalized to SIS.

Circuit	SIS			SimE-G1			% Improvement		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
circuit5	20169	8.17	7.57	11664	5.57	5.08	72.92	46.74	49.05
circuit6	15066	5.09	5.66	10935	3.34	3.82	37.78	52.31	48.26
circuit7	14580	6.79	4.75	12393	5.05	4.38	17.65	34.51	8.44
circuit8	7776	5.23	2.74	7290	2.96	3.16	6.67	76.77	-13.43
circuit9	20412	7.46	7.12	17454	6.65	7.01	16.95	12.19	1.54
circuit10	9963	6.42	3.34	9963	6.42	3.33	0.00	0.01	0.27
circuit11	17496	7.08	6.01	17496	7.07	6.01	0.00	0.13	0.04
circuit12	52731	17.34	19.11	48655	16.55	18.94	8.38	4.77	0.92
circuit13	24543	11.02	9.72	22599	8.85	8.88	8.60	24.55	9.45
circuit14	31347	10.03	11.55	24786	7.25	9.20	26.47	38.30	25.46
circuit15	24057	8.80	7.99	20898	6.61	6.37	15.12	33.22	25.64
circuit16	9720	8.50	3.45	9720	8.50	4.30	0.00	0.10	-19.69
circuit17	12636	6.38	4.76	11565	3.89	5.34	9.26	64.13	-10.87

Table 8.3: Comparison of SimE-G1 and SIS in area optimization for single output circuits.

The results of multiple outputs circuits for area optimization is shown in Table 8.4 using SimE-G1 with AODPC. The improvement in area varied where the highest improvements observed for multiplier circuits (mul2 and mul3) and cm82a circuit. However, SimE-G1 failed to deliver better circuits in terms of area for b1 and this due to the excessive utilization of subexpressions in b1 using SIS. Also, SimE-G1 uses multi-objective optimization, the result generated for b1 is better than SIS in term of delay and power.

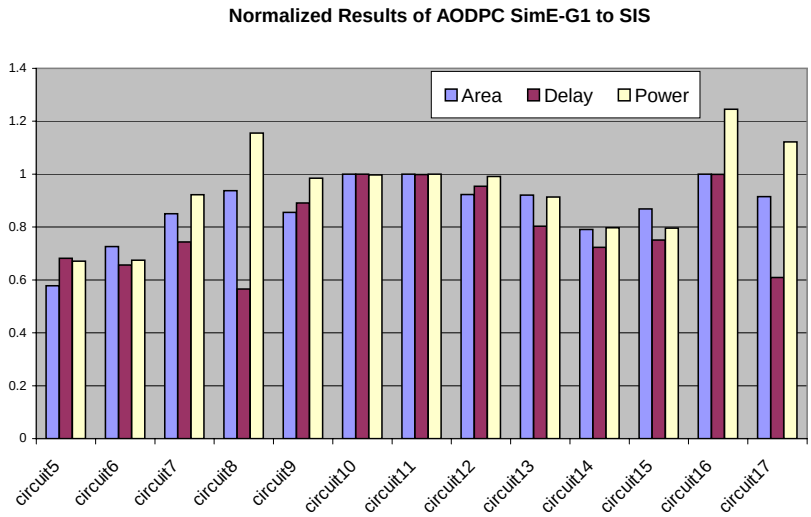


Figure 8.1: Results of SimE-G1 with AODPC for single output functions, normalized to SIS.

Circuit	SIS			SimE-G1			% Improvement		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
mul2	18225	6.59	5.56	12636	3.56	4.66	44.23	84.98	19.35
mul3	112752	43.39	37.75	74358	13.14	21.65	51.63	230.23	74.38
cm42a	40824	8.86	13.65	40824	8.86	13.64	0.00	0.05	0.06
cm82a	39609	19.54	14.88	28552	9.34	9.10	38.73	109.21	63.51
b1	10206	3.23	3.99	11206	2.91	2.78	-8.92	10.85	43.68
c17	9963	3.56	3.64	9963	3.55	3.64	0.00	0.27	0.06
con1	31590	8.64	11.21	30233	6.90	14.23	4.49	25.19	-21.21
majority	14823	6.28	5.41	13851	4.57	5.06	7.02	37.32	6.93

Table 8.4: Comparison of SimE-G1 and SIS in area optimization for multiple output circuits.

The graphical view of SimE-G1 results, normalized to SIS is shown in Figure 8.2.

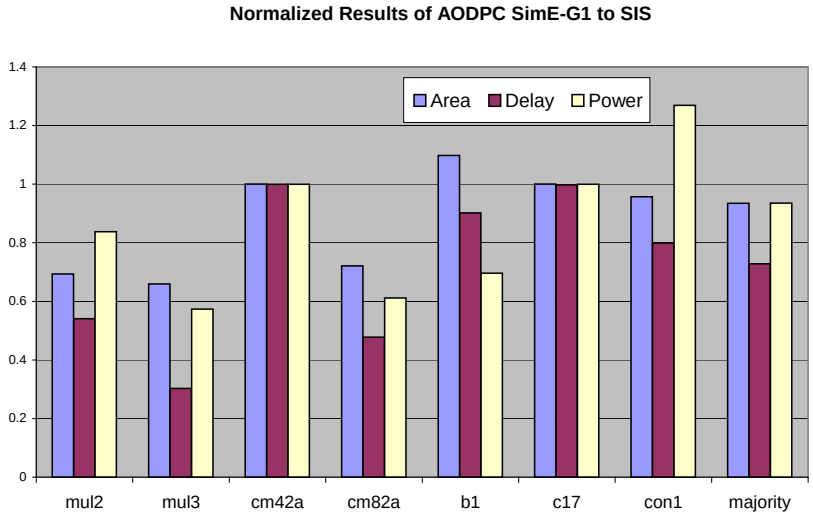


Figure 8.2: Results of SimE-G1 with AODPC for multiple outputs functions, normalized to SIS.

Table 8.5 shows the results for single output circuits for SimE-G2 with AODPC compared to SIS area minimization. The table shows that SimE-G2 provides circuits with less area as compared to SIS. The highest improvements were observed for the circuits 5 and 6. Figure 8.3 shows the graphical view of the results from SimE-G2. Note that these results are normalized to SIS.

The results of multiple outputs circuits for area optimization is shown in Table 8.6 for SimE-G2 with AODPC compared to SIS area. The improvement in area varied where the highest improvements observed for multiplier circuits (mul2 and mul3) and cm82a circuit. However, SimE-G2 failed to deliver better circuits in terms of area for b1 due to the same reason mentioned above.

The graphical view of SimE-G2 results, normalized to SIS is shown in Figure 8.4.

Circuit	SIS			SimE-G2			% Improvement		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
circuit5	20169	8.17	7.57	11664	5.57	5.08	72.92	46.74	49.05
circuit6	15066	5.09	5.66	10935	3.34	3.82	37.78	52.31	48.26
circuit7	14580	6.79	4.75	12393	5.05	4.38	17.65	34.51	8.44
circuit8	7776	5.23	2.74	7290	2.96	3.16	6.67	76.77	-13.43
circuit9	20412	7.46	7.12	14970	5.20	4.91	36.35	43.48	44.97
circuit10	9963	6.42	3.34	9963	6.42	3.33	0.00	0.01	0.27
circuit11	17496	7.08	6.01	16354	6.40	5.75	6.98	10.61	4.56
circuit12	52731	17.34	19.11	48655	16.55	18.94	8.38	4.77	0.92
circuit13	24543	11.02	9.72	18523	9.77	7.80	32.50	12.81	24.62
circuit14	31347	10.03	11.55	23814	6.80	9.20	31.63	47.41	25.46
circuit15	24057	8.80	7.99	20655	6.40	5.89	16.47	37.47	35.77
circuit16	9720	8.51	3.45	9720	8.50	4.30	0.00	0.10	-19.69
circuit17	12636	6.38	4.76	10692	4.43	4.39	18.18	44.22	8.44

Table 8.5: Comparison of SimE-G2 and SIS in area optimization for single output circuits.

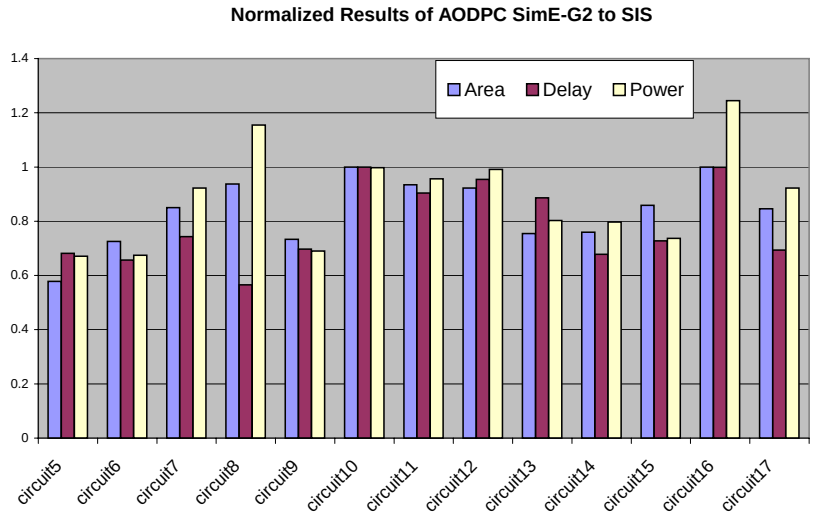


Figure 8.3: Results of SimE-G2 with AODPC for single output functions, normalized to SIS.

Circuit	SIS			SimE-G2			% Improvement		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
mul2	18225	6.59	5.56	12636	3.56	4.66	44.23	84.98	19.35
mul3	112752	43.39	37.75	74358	13.14	21.65	51.63	230.23	74.38
cm42a	40824	8.86	13.65	38456	9.44	12.60	6.16	-6.10	8.32
cm82a	39609	19.54	14.88	25029	11.84	9.24	58.25	64.98	61.12
b1	10206	3.23	3.99	11206	2.91	2.78	-8.92	10.85	43.68
c17	9963	3.56	3.64	9963	3.55	3.64	0.00	0.27	0.06
con1	31590	8.64	11.21	30233	6.90	14.23	4.49	25.19	-21.21
majority	14823	6.28	5.41	13851	4.57	5.06	7.02	37.32	6.93

Table 8.6: Comparison of SimE-G2 and SIS in area optimization for multiple output circuits.

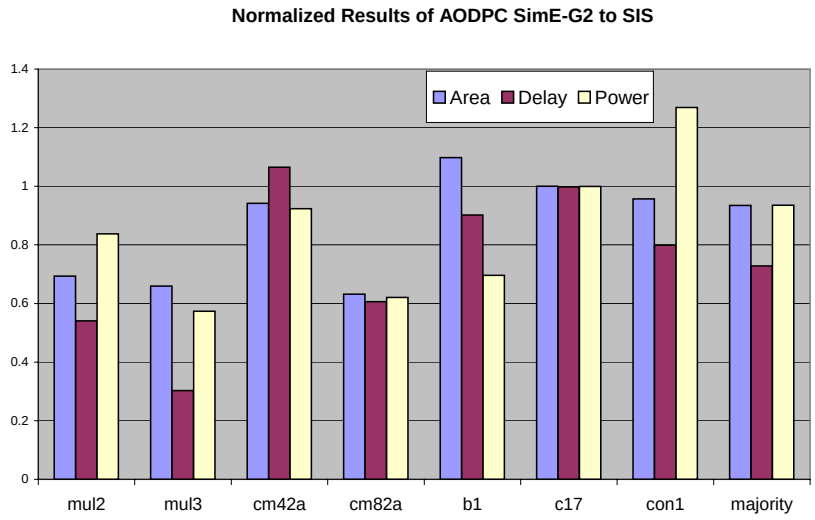


Figure 8.4: Results of SimE-G2 with AODPC for multiple outputs functions, normalized to SIS.

Delay Optimization:

For delay optimization, the results from SIS were obtained by using *delay.script*, mapped for delay minimization. All SimE-G1, SimE-G2 and SIS used the same gate library for the experiments. Only SimE-G2 with DOAPC will be compared to

SIS in order to get an insight of the results because results of SimE-G2 and SimE-G1 are similar with minor improvements in SimE-G2. Table 8.7 and Table 8.8 shows the results for single output circuits and multiple output circuits respectively. The graphical view of the normalized results was given in Figure 8.5 and Figure 8.6.

Circuit	SIS			SimE-G2			% Improvement		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
circuit5	20169	8.17	7.57	11664	5.57	5.08	72.92	46.74	49.05
circuit6	15066	5.09	5.66	10935	3.34	3.82	37.78	52.31	48.26
circuit7	14580	6.79	4.75	12393	5.05	4.38	17.65	34.51	8.44
circuit8	7776	5.23	2.74	7290	2.96	3.16	6.67	76.77	-13.43
circuit9	20412	7.46	7.12	17454	6.65	7.01	16.95	12.19	1.54
circuit10	9963	6.42	3.34	9963	6.42	3.33	0.00	0.01	0.27
circuit11	17496	7.08	6.01	17496	7.07	6.01	0.00	0.13	0.04
circuit12	52731	17.34	19.11	48655	16.55	18.94	8.38	4.77	0.92
circuit13	24543	11.02	9.72	22599	8.85	8.88	8.60	24.55	9.45
circuit14	31347	10.03	11.55	24786	7.25	9.20	26.47	38.30	25.46
circuit15	24057	8.80	7.99	20898	6.61	6.37	15.12	33.22	25.64
circuit16	9720	8.51	3.45	9720	8.50	4.30	0.00	0.10	-19.69
circuit17	12636	6.38	4.76	11565	3.89	5.34	9.26	64.13	-10.87

Table 8.7: Comparison of SimE-G2 and SIS in delay optimization for single output circuits.

Circuit	SIS			SimE-G2			% Improvement		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
mul2	18225	6.59	5.56	12636	3.56	4.66	44.23	84.98	19.35
mul3	112752	43.39	37.75	74358	13.14	21.65	51.63	230.23	74.38
cm42a	40824	8.86	13.65	40824	8.86	13.64	0.00	0.05	0.06
cm82a	39609	19.54	14.88	28552	9.34	9.10	38.73	109.21	63.51
b1	10206	3.23	3.99	11206	2.91	2.78	-8.92	10.85	43.68
c17	9963	3.56	3.64	9963	3.55	3.64	0.00	0.27	0.06
con1	31590	8.64	11.21	30233	6.90	14.23	4.49	25.19	-21.21
majority	14823	6.28	5.41	13851	4.57	5.06	7.02	37.32	6.93

Table 8.8: Comparison of SimE-G2 and SIS in delay optimization for multiple output circuits.

The tables show that the improvement in delay can reach up to 76% (circuit8)

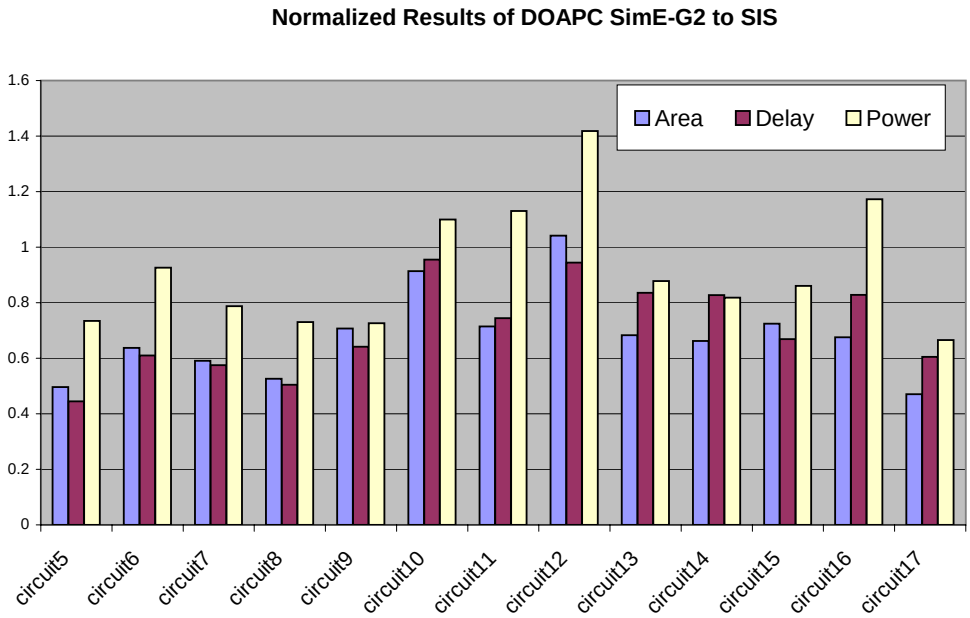


Figure 8.5: Results of SimE-G2 with DOAPC for single outputs functions, normalized to SIS.

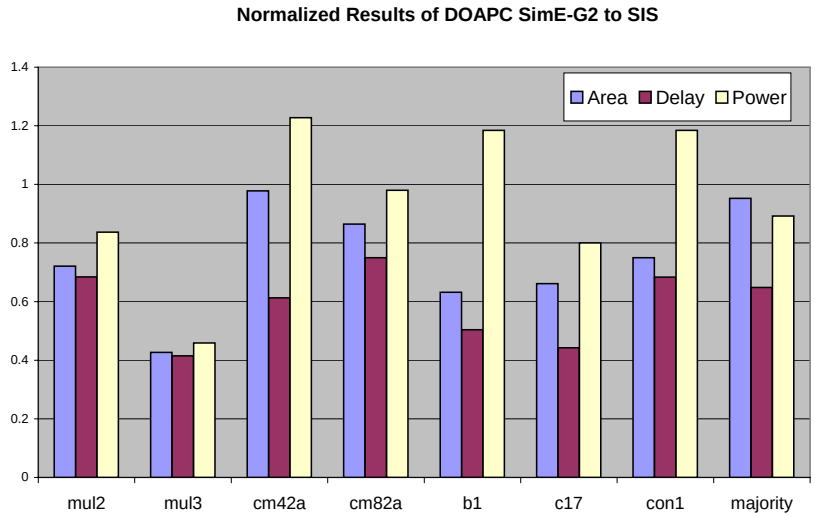


Figure 8.6: Results of SimE-G2 with DOAPC for multiple outputs functions, normalized to SIS.

and 230% (mul3). However, area did not improve in certain cases and it get worst in some cases like b1. In contrast with AODPC, the results of DOAPC is very positive in term of delay. However, the experiments show that DOAPC is better compared to SIS in delay but not in power and area. One of the reasons behind this is the nature of SimE algorithm, since it is similar to the placement (allocation) problem.

8.3 Comparison with Other Techniques

Other techniques includes Tabu Search which have been proposed in Chapter 6, and, Ant Colony Optimization Technique is compared to SimE algorithm. The results obtained by SimE-G2 is compared to the result of the other techniques in terms of area, power and delay.

8.3.1 Comparison with Tabu Search

Table 8.9 shows the results for all circuits obtained using TS considering APODC compared to SimE-G2 considering APODC. The results show that for small size circuit, both TS and SimE-G2 have equal value of area, delay and power. However, when the size of the circuits grow bigger, SimE-G2 gives better results than TS. Moreover, the decline in performance for TS is observed greatly in multiple outputs circuits. Some multiple output circuits in the benchmark being used for the work could not be generated using TS such as mul3, add2 and add3 and this is due to

the increasing size of the problem. Also, for larger multiple output circuits, there is an increase usage of subexpression which TS can not handle.

Circuit	TS			SimE-G2			% Improvement		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
circuit1	12879	3.90	4.97	12879	3.90	4.97	0.00	0.00	0.00
circuit2	13122	5.18	5.28	13122	5.18	5.28	0.00	0.00	0.00
circuit3	10752	3.78	3.70	10752	3.78	3.70	0.00	0.00	0.00
circuit4	1458	0.005	0.66	1458	0.005	0.66	0.00	0.00	0.00
circuit5	13870	3.90	5.80	13870	3.90	5.80	0.00	0.00	0.00
circuit6	10935	3.34	3.82	10935	3.34	3.82	0.00	0.00	0.00
circuit7	12393	5.05	4.38	12393	5.05	4.38	0.00	0.00	0.00
circuit8	7776	5.22	2.73	7776	5.22	2.73	0.00	0.00	0.00
circuit9	16255	5.48	5.40	14970	5.20	4.91	8.58	5.38	9.98
circuit10	13472	5.78	5.25	9963	6.42	3.33	35.22	9.97	57.66
circuit11	19367	6.89	8.92	16354	6.40	5.75	18.42	7.66	55.13
circuit12	110266	24.82	32.67	48655	16.55	18.94	126.63	49.97	72.49
circuit13	25680	10.24	9.87	18523	9.77	7.80	38.64	4.81	26.54
circuit14	43266	10.45	15.42	23814	6.80	9.20	81.68	53.63	67.54
circuit15	25446	8.53	7.34	20655	6.40	5.89	23.20	33.24	24.60
circuit16	15338	7.88	5.20	9720	8.50	3.45	57.80	7.29	50.72
circuit17	21376	7.32	9.24	10692	4.43	4.39	99.93	65.35	110.53
add2	42566	11.50	16.83	24300	11.48	9.96	75.17	0.17	68.94
mul2	120344	15.62	19.43	13650	4.67	4.34	781.64	234.48	347.70
b1	30989	14.99	15.44	11206	2.91	2.78	176.54	415.12	455.40
c17	12330	8.34	9.41	10923	5.23	3.42	12.88	59.46	175.15
majority	33478	10.66	11.19	13851	4.57	5.06	141.70	133.21	121.36
xor8	45876	11.46	14.85	20655	5.90	9.32	122.11	94.24	59.42
xor9	60912	18.24	22.46	23814	9.57	10.65	155.78	90.54	110.95

Table 8.9: Comparison of TS and SimE-G2 in area and power optimization with delay constraint.

Comparing SimE-G2 to TS in terms of time is shown in Table 8.10. For smaller circuits, TS has less time than SimE-G2. On the other hand, for larger circuits, TS spends large amount of time in order to give a solution compared to SimE-G2. The reason is that SimE-G2 allocation makes several mutation (compound moves)

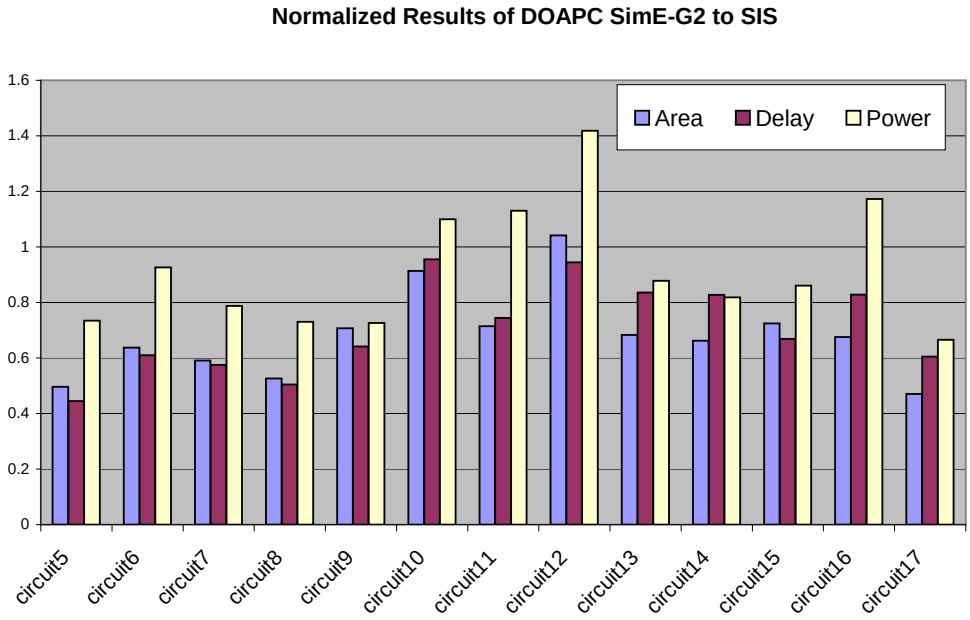


Figure 8.7: Results of SimE-G2 with APODC normalized to TS.

compared to TS which make single move (change) in every iteration to generate N solution which might not effect the goodness of solution.

8.3.2 Comparison with ACO

In this section, SimE-G2 is compared to Ant Colony Optimization (ACO) in terms of area optimization, delay optimization and power optimization.

Table 8.11 shows the comparison between ACO and SimE-G2 for Area Optimization. The area produced by SimE-G2 is better than the area produced by ACO in 10 out of the 15 cases. Only in 2 cases where ACO outperformed SimE-G2. Also, Table 8.12 shows the results of ACO and SimE-G2 for Delay Optimization. The delay produced by ACO is better than the delay produced by SimE-G2 in 9 out

Circuit	TS(s)	SimE-G2 (s)	% Improvement
circuit1	1.20	3.12	61.54
circuit2	1.01	3.77	73.21
circuit3	2.55	4.57	44.20
circuit4	0.07	0.48	85.42
circuit5	1.32	6.51	79.72
circuit6	1.22	6.02	79.73
circuit7	2.05	7.44	72.45
circuit8	1.77	5.02	64.74
circuit9	1472.83	8.22	-17817.64
circuit10	5623.75	9.43	-59536.80
circuit11	6573.87	25.30	-25883.68
circuit12	15673.24	30.52	-51254.00
circuit13	10972.26	28.94	-37813.82
circuit14	18592.62	40.73	-45548.47
circuit15	22376.91	70.94	-31443.43
circuit16	16578.42	65.83	-25083.69
circuit17	25438.77	77.49	-32728.46
add2	9.55	24.84	61.55
mul2	20.68	25.72	19.60
b1	15.43	25.93	40.49
c17	12.50	42.11	70.32
majority	18.71	15.32	-22.13
xor8	16.52	220.43	92.51
xor9	22.91	231.00	90.08

Table 8.10: Comparison with TS in terms of execution time.

of the 15 cases. Only in 3 cases where SimE-G2 outperformed ACO in terms of delay. This is due to the nature of ACO is optimizing the paths unlike SimE-G2 is optimizing locations of cells. Moreover, Table 8.13 shows the comparison between ACO and SimE-G2 for Power Optimization. The power produced by SimE-G2 is better than the power produced by ACO in 11 out of the 15 cases. Only in 1 case where ACO outperformed SimE-G2.

Circuit	ACO				SimE-G2				% Improvement			
	Area	Delay	Power	Time	Area	Delay	Power	Time	Area	Delay	Power	Time
circuit1	14094	4.51	5.28	15.27	12879	3.90	4.97	2.52	-9.43	-15.70	-6.25	-505.82
circuit2	14823	5.89	6.57	15.17	13122	5.18	5.28	3.09	-12.96	-13.70	-24.38	-390.83
circuit10	12393	6.05	4.48	75.70	9963	6.42	3.33	10.52	-24.39	5.73	-34.62	-619.58
circuit14	24786	7.25	9.20	222.40	23814	6.80	9.20	74.81	-4.08	-6.59	0.00	-197.29
circuit16	12393	5.71	4.26	188.33	9720	8.50	4.30	55.91	-27.50	32.78	0.88	-236.85
circuit17	10692	4.43	4.39	96.10	10692	4.43	4.39	80.52	0.00	0.00	0.00	-19.35
add2	24300	11.48	9.96	133.14	24300	11.48	9.96	30.71	0.00	0.00	0.00	-333.53
add3	49086	21.96	18.474	2009.50	40265	26.73	14.83	140.43	-21.91	17.85	-24.57	-1330.96
mul2	12636	3.56	4.66	57.20	12636	3.56	4.66	51.62	0.00	0.00	0.00	-10.81
mul3	59292	15.03	17.541	625.24	74358	13.14	21.65	239.11	20.26	-14.40	18.96	-161.48
cm42a	38880	7.45	13.25	1004.21	38456	9.44	12.60	388.43	-1.10	21.08	-5.17	-158.53
cm82a	25272	10.45	9.96	168.83	25029	11.84	9.24	110.20	-0.97	11.78	-7.89	-53.21
b1	13851	1.53	5.91	89.53	11206	2.91	2.78	60.30	-23.60	47.46	-112.45	-48.48
con1	38151	6.698	13.741	398.23	30233	6.90	14.23	330.43	-26.19	2.93	3.44	-20.52
rd53	35235	15.32	14.00	180.61	38073	14.75	15.34	70.44	7.45	-3.88	8.75	-156.40

Table 8.11: Comparison of ACO and SimE-G2 considering area optimization.

Circuit	ACO				SimE-G2				% Improvement			
	Area	Delay	Power	Time	Area	Delay	Power	Time	Area	Delay	Power	Time
circuit1	14823	3.86	6.41	15.27	15360	4.20	5.80	2.52	3.50	8.14	-10.47	-505.82
circuit2	17739	3.70	7.16	15.17	16870	3.88	6.90	3.09	-5.15	4.54	-3.70	-390.83
circuit10	12393	5.32	4.48	75.70	13771	5.50	4.77	10.52	10.01	3.22	6.02	-619.58
circuit14	30132	9.75	10.63	222.40	24786	7.25	9.20	74.81	-21.57	-34.52	-15.47	-197.29
circuit16	13122	5.71	4.26	188.33	10340	7.60	5.22	55.91	-26.91	24.82	18.35	-236.85
circuit17	12150	4.43	4.72	96.10	12354	5.30	5.38	80.52	1.65	16.47	12.23	-19.35
add2	31347	8.96	11.46	133.14	35270	9.37	12.66	30.71	11.12	4.41	9.45	-333.53
add3	53703	12.98	21.48	2009.50	53703	12.98	21.48	140.43	0.00	0.00	0.00	-1330.96
mul2	18225	2.96	5.99	57.20	18225	2.96	5.99	51.62	0.00	0.00	0.00	-10.81
mul3	74358	13.14	21.65	625.24	74358	13.14	21.65	239.11	0.00	0.00	0.00	-161.48
cm42a	91854	7.07	32.89	1004.21	42768	5.19	15.02	388.43	-114.77	-36.37	-118.97	-158.53
cm82a	48843	7.75	17.09	168.83	55872	14.25	18.56	110.20	12.58	45.61	7.95	-53.21
b1	14580	1.49	6.24	89.53	12745	2.08	7.05	60.30	-14.40	28.61	11.49	-48.48
con1	38394	6.24	12.65	398.23	30233	6.90	14.23	330.43	-26.99	9.55	11.08	-20.52
rd53	53946	15.45	19.00	180.61	40201	13.10	15.98	70.44	-34.19	-17.90	-18.88	-156.40

Table 8.12: Comparison of ACO and SimE-G2 considering delay optimization.

Circuit	ACO				SimE-G2				% Improvement			
	Area	Delay	Power	Time	Area	Delay	Power	Time	Area	Delay	Power	Time
circuit1	15552	4.88	5.28	15.27	12879	3.90	4.97	2.52	-20.75	-25.21	-6.25	-505.82
circuit2	15066	5.94	6.57	15.17	13122	5.18	5.28	3.09	-14.81	-14.51	-24.38	-390.83
circuit10	11178	5.42	4.06	75.70	9963	6.42	3.33	10.52	-12.20	15.58	-21.98	-619.58
circuit14	23814	6.80	9.20	222.40	23814	6.80	9.20	74.81	0.00	0.00	0.00	-197.29
circuit16	13122	5.71	4.19	188.33	9720	8.50	3.45	55.91	-35.00	32.78	-21.57	-236.85
circuit17	10692	4.43	4.39	96.10	10692	4.43	4.39	80.52	0.00	0.00	0.00	-19.35
add2	25029	13.30	9.57	133.14	24317	14.73	9.25	30.71	-2.93	9.69	-3.44	-333.53
add3	54675	14.36	20.55	2009.50	40265	26.73	14.83	140.43	-35.79	46.29	-38.58	-1330.96
mul2	14823	4.44	4.66	57.20	13650	4.67	4.34	51.62	-8.59	5.03	-7.37	-10.81
mul3	73386	16.331	19.11	625.24	74358	13.14	21.65	239.11	1.31	-24.30	11.73	-161.48
cm42a	46170	6.39	15.27	1004.21	38456	9.44	12.60	388.43	-20.06	32.25	-21.17	-158.53
cm82a	25029	11.84	9.24	168.83	25029	11.84	9.24	110.20	0.00	0.00	0.00	-53.21
b1	14580	3.05	5.37	89.53	13220	4.86	3.04	60.30	-10.29	37.35	-76.55	-48.48
con1	42282	9.79	13.97	398.23	40632	11.78	10.04	330.43	-4.06	16.92	-39.11	-20.52
rd53	42525	14.84	16.19	180.61	40231	15.33	13.89	70.44	-5.70	3.22	-16.56	-156.40

Table 8.13: Comparison of ACO and SimE-G2 considering power optimization.

8.4 Concluding Remarks

In this chapter, the proposed solution is compared to the existing techniques. It proved to be better in terms of quality of solution and execution time in contrast with the existing ELD and ACO techniques. It was also compared to existing conventional tools. It was observed that the area optimization applied by the solution approach performs better in most of the cases.

Chapter 9

Conclusion and Future Directions

Design of digital circuits requires knowledge of large collections of domain-specific rules. The process of implementing a digital circuit in hardware involves transforming the original logical specification into a form suitable for the target technology, optimizing the representation with respect to a number of user defined constraints (i.e., timing, fan-in/out, power, etc.), and finally carrying out technology mapping onto the target technology [28].

Circuit designers use logic synthesis tools to create digital systems of arbitrary complexity. Logic synthesis tools can be classified into two types according to the algorithms used as conventional logic synthesis tools and evolutionary logic synthesis tools.

In conventional logic synthesis, given a precis specification of a circuit in the form of truth table or boolean expression, a circuit is synthesized for the target

technology using deterministic algorithms. Therefore, the solutions search space is limited to specific representations including two-level, multi-level or Reed Muller.

In evolutionary logic synthesis, a non-deterministic algorithms are used. Evolutionary techniques explore a larger search space compared to the conventional one. The number of possible solutions is huge and the algorithm will search for an optimum one according to the objectives considered such as power, area and delay.

9.1 Conclusion

In this thesis, Simulated Evolution (SimE) algorithm is used for evolutionary logic design. Two goodness measures are proposed to guide the algorithm in the search space. The performance of the proposed algorithms has been compared to both existing evolutionary techniques and conventional techniques. The proposed algorithm successfully addressed the issue of circuit design targeting area, power and delay optimization. The proposed implementation has outperformed the existing evolutionary techniques in terms of quality of solution and time requirements. Also, it has outperformed the conventional techniques in quality of solution in most of the cases. More work need to be done in order the improve the time requirements of evolutionary logic synthesis compared to conventional techniques. Conventional techniques have outperformed evolutionary techniques in term of execution time and problem size.

9.2 Future Directions

There are more work can be done in order to improve the performance of evolutionary algorithms in the design of logic circuits. Some possible improvements including the investigation of the following:

1. Using some of the conventional logic techniques such as kernel extraction.
2. Incorporation the ability of utilizing sub-function by proposing new goodness measures
3. Using hybrid algorithms
4. Using other evolutionary algorithms
5. Considering paralyzation of algorithms.
6. Incorporate the cost of power, area and delay in the goodness function.

Appendix A

File Format and Circuit Used as Test Cases

The following are the format of the required input files and the list of circuits used as test cases during the experiments.

A.1 Library File Format

The gates' parameters are obtained from CMOS MOSIS 0.25 m library. These includes the following information:

1. The number of transistors (TRCOUNT)
2. The size of the gate (AREA)
3. The switching delay of the gate (BDELAY)
4. The input capacitance (CIN)

5. The output capacitance (COUT)
6. The load factor (LF)

Except for inverters and wires, there is more than one value for input capacitance. In this regards, the higher value is considered. The same applied for the load factor.

The following shows the format of the library files used.

<GATE NAME> <TRCOUNT> <AREA> <BDELAY> <CIN> <COUT> <LF>

Using the above format, the library file considered in this thesis is given next.

WIRE	0	0	0	0	0	0
NOT	2	1215	3	2.661	0.005	516
AND	6	2187	9	2.661	0.005	553
OR	6	1944	11	2.661	0.004	567
XOR	10	3159	12	5.321	0.006	688
NAND	4	1458	5	2.661	0.007	444
NOR	4	1458	7	2.661	0.005	694
XNOR	10	2916	12	5.321	0.004	551

A.2 Input File Format

The proposed algorithms use PLA files as input. The parameters required by the proposed algorithms are then generated automatically. However, if the circuits' specification is not in the form of PLA files, the input file required by the algorithm must contain the following information.

Field	**Description**
maxrun	The number of maximum runs to be performed by the algorithm
maxiter	The number of maximum iterations for a single run
objtype	The optimization objective of the current action. (0 = gatecount, 1 = area, 2 = delay, and 3 = power)
ifdweight	Static (0) or dynamic (1) weight is employed
weight	The value Wf at the beginning of the iteration
ifphase	Positive (0) or negative (1) phase is used. In positive mode, all literals are uncomplemented. In negative mode, some literals can be in the complemented forms, depending on its functional fitness. (not used, default = 0)
dataset	The number of input dataset. Useful for generating truth table for arithmetic circuits such as adders, multipliers. For example, a 2-bit adder has dataset value equal to 2. (not used, default value = 1)
numvar	Number of inputs
numout	Number of outputs
invin	Set to '1' if complemented literals are used in addition to uncomplemented ones. (default = 0)
row	The number of rows at the beginning of the iteration
level	The number of columns at the beginning of the iteration
iflocal	Set to 1 if local search is employed (not used)
gatemode	The type of library used. Values are within [0-7]
gcmax	Set the maximum number of gates allowed for a circuit (not used)
gcmin	Set the minimum number of gates allowed for a circuit (not used)
numant	Set the number of ants used

Field	**Description**
maxgen	Set the maximum number of generations of ant
gamma	Set the constant for pheromone update calculation
rho	Set the pheromone evaporation rate
range_tau	Set the maximum range of pheromone values allowed
ttf[0]	The string of truth table of the first output of the circuit
ttf[1]	The string of truth table of the second output of the circuit (if available)
ttf[k]	The string of truth table of the k+1 output of the circuit (if available)

A.3 Randomly Generated Circuits

There are 20 randomly generated circuits used in the experiments. These circuits are listed below.

circuit	input(s)	output(s)	circuit	input(s)	output(s)
circuit1.pla	4	1	circuit11.pla	5	1
circuit2.pla	4	1	circuit12.pla	6	1
circuit3.pla	4	1	circuit13.pla	5	1
circuit4.pla	2	1	circuit14.pla	6	1
circuit5.pla	4	1	circuit15.pla	6	1
circuit6.pla	4	1	circuit16.pla	6	1
circuit7.pla	4	1	circuit17.pla	5	1
circuit8.pla	4	1	circuit18.pla	6	1
circuit9.pla	6	1	circuit19.pla	6	1
circuit10.pla	6	1	circuit20.pla	6	1

A.4 Benchmark Circuits

There are 14 benchmark circuits used in the experiments. These circuits are listed below.

circuit	input	output	circuit	input	output
majority.pla	5	1	b1.pla	3	4
xor8.pla	8	1	C17.pla	5	2
xor9.pla	9	1	cm138a.pla	6	8
add2.pla	5	3	cm42a.pla	4	10
add3.pla	7	4	cm82a.pla	5	3
mul2.pla	4	4	con1.pla	7	2
mul3.pla	6	6	rd53.pla	5	3

Bibliography

[1] Standard Cell Library for MOSIS CMOS, http://www.mosis.org/Technical/Designsupport/std-cell-library-scmos.html.

[2] S. B. Akers. Binary decision diagrams. *IEEE Trans. on Computers*, 27:509–516, 1978.

[3] D. Bostick et al. The Boulder Optimal Logic Design System. *Proceeding of the International Conference on CAD*, pages 62–65, Nov. 1987.

[4] R. Brayton, G. D. Hachtel, C. T. McMullen, and A.L. Sangiovanni-Vincentelli. *Logic Minimisation Algorithms for VLSI Synthesis.* Kluwer Academic Publisher, 1984.

[5] R. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. Multilevel Logic Synthesis. *Proceeding of the IEEE*, 78:264–300, Feb. 1990.

[6] R. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. Wang. MIS: A Muliple-Level Logic Optimisation System. *IEEE Trans. on Computer-Aided*

Design, CAD-6:1062–1081, Nov. 1987.

[7] C. A. Coello, A. D. Christiansen, and A. H. Aguirre. Use of Evolutionary Techniques to Automate the Design of Combinational Circuits. *International Journal of Smart Engineering System Design, Elsevier Science*, 2(4):299–314, June 2000.

[8] C. A. Coello, A. D. Christiansen, and A. H. Aguirre. Towards Automated Evolutionary Design of Combinational Circuits. *Computers and Electrical Engineering, Pergamon Press*, 27(1):1–28, Jan. 2001.

[9] Coello Coello, Carlos A., and Hernandez Luna, Erika and Hernandez Aguirre, Arturo,. Use of Particle Swarm Optimization to Design Combinational Logic Circuits. *Evolvable Systems: From Biology to Hardware. 5th International Conference, ICES 2003*, 2606:398–409, Mar 2003.

[10] Hugo de Garis. Evolvable Hardware: Genetic Programming of a Darwin Machine. *Proceedings of the International Conference in Innsbruck, Austria*, pages 441–449, Springer-Verlag, 1993.

[11] D. Debnath and T. Sasao. An Optimization of AND-EXOR Three-Level Networks. *Proceeding of Asia and South Pacific Design Automation Conference*, pages 545–550, Jan. 1997.

[12] Srinivas Devadas and Sharad Malik. A Survey of Optimization Techniques Targeting Low Power VLSI Circuits. *32nd ACM/IEEE Design Automation Conference*, pages 242–247, 1995.

[13] Chih-Shun Ding, Chi-Ying Tsui, and Masoud Pedram. Gate Level Power Estimation Using Tagged Probabilistic Simulation. *IEEE Trans. Computer-Aided of Integrated Circuit and Systems*, 17(11):1099 – 1107, Nov. 1998.

[14] Moshe Sipper et. al. A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems. *IEEE Trans. on Evolutionary Computation*, 1(1):83–97, 1997.

[15] T. Fogarty, J. F. Miller, and P. Thomson. Evolving Digital Logic Circuits on Xilinx 6000 Family FPGAs. *The 2nd Online Conference on Soft Computing in Engineering Design and Manufacturing*, pages 299–305, Springer-Verlag, London, 1998.

[16] D. Green. *Modern Logic Design.* Addison-Wesley, Reading, MA, 1986.

[17] Inman Harvey and Adrian Thompson. Through the Labyrinth Evolution Finds a Way: A Silicon Ridge. *Proceedings of Second International Conference on Evolvable Systems*, pages 406–422, 1996.

[18] T. Higuchi, M. Iwata, I. Kajitani, M. Murakawa, S. Yoshizawa, and T. Furuya. Hardware Evolution at Gate and Functional Level. *Proceeding of the Interna-*

tional Conference on Biologically Inspired Autonomous Systems: Computation, Cognition and Action, March Durham, USA, 1996.

[19] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, and T. Furuya. Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. *Proceeding of the 2nd Int. Conf. on The Simulation of Adaptive Behavior (SAB92)*, MIT Press, 1993.

[20] B. Hounsell and T. Arslan. A Novel Genetic Algorithm for the Automated Design of Performance Driven Digital Circuits. *CEC-2000*, pages 601–608, 2000.

[21] R. M. Kling and P. Banerjee. ESP: A New Standard Cell Placement Package using Simulated Evolution. *Proceeding of 24th Design Automation Conference*, pages 60–66, 1987.

[22] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, San Fransisco, CA, 1999.

[23] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane. Automated Design of Both the Topology and Sizing of Analogue Electrical Circuits Using Genetic Programming. *Artificial Intelligent in Design*, pages 151–170, Kluwer Academic Publishers, 1996.

[24] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[25] C. Y. Lee. Representation of Switching Circuit by Binary Decision Programs. *Bell Systems Technical Journal*, 38:985–999, 1959.

[26] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York, 1994.

[27] J. F. Miller, T. Fogarty, and P Thomson. Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science, John Wiley and Sons, Chichester*, pages 105–131, 1998.

[28] J. F. Miller, D. Job, and Vassilev V. K. Principles in the Evolutionary Design of Digital Circuits - Part I. *Journal of Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.

[29] J. F. Miller and P. Thomson. Discovering Novel Digital Circuits Using Evolutionary Techniques. *IEE Colloquium on Evolvable Systems, Savoy Place, London*, March 1998.

[30] J. F. Miller and Peter Thomson. A Developmental Method for Growing Graphs and Circuits. *GECCO'03*, 1(1):8–35, 2003.

[31] Massoud Pedram. Power Minimization in IC Design: Principles and Applications. *ACM Trans. Design Automation of Electronic Systems*, 1(1):3 – 56, Jan. 1996.

[32] M. A. Pacheco R. S. Zebulum and Marley Vellasco. Evolvable systems in hardware design: taxonomy, survey and applications. *Evolvable System: From Biology to Hardware. Proceeding of the First International Conference, ICES 96 Tsukba, Japan, Lecture Notes in Computer Science*, 1259:344–358, Oct. 1997.

[33] R. S. Zebulum and M. A. Pacheco and Maria Vellasco. *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms.* CRC Press, 2002.

[34] Sadiq M. Sait and H. Youssef. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems.* IEEE Computer Society Press, 1999.

[35] T. Sasao. *Logic Synthesis and Optimisation.* Kluwer Academic Publisher, 1993.

[36] E. M. Sentovic, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, University of California, Berkeley, May 1992.

[37] Adrian Thompson. Silicon Evolution. *Proceedings of the First Annual Conference on Genetic Programming*, pages 444–452, MIT Press, 1996.

[38] Adrian Thompson, Paul Layzell, and Ricardo Salem Zebulum. Exploration in Design Space: Unconventional Electronics Design through Artificial Evolution. *IEEE Trans. On Evolutionary Computation*, 3(3), 2000.

[39] P. Thomson and J. P. Miller. Symbolic Method for Simplifying AND-EXOR Representation of Boolean Functions Using a Binary Decision Technique and a Genetic Algorithm. *IEE Proceedings in Computers and Digital Techniques*, 143:151–155, 1996.

[40] V. K. Vassilev, D. Job, and J. F. Miller. Towards the Automatic Design of More Efficient Digital Circuits. *Proceedings of the 2nd NASA/DOD Workshop on Evolvable Hardware*, pages 151–160, IEEE Computer Society, 2000.

[41] V. K. Vassilev and J. F. Miller. Scalability Problems of Digital Circuit Evolution: Evolvability and Efficient Designs. *Proceedings of the 2nd NASA/DOD Workshop on Evolvable Hardware*, pages 55–64, IEEE Computer Society, 2000.

[42] R. Yager. Multiple objective decision-making using fuzzy sets. *International Journal of Man-Machine Studies*, pages 9:375–382, 1977.

[43] R. Yager. Second order structures in multi-criteria decision making. *International Journal of Man-Machine Studies*, pages 36:553–570, 1992.

[44] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transaction on Systems, MAN, and Cybernetics*, 18(1):183–190, Jan 1988.

[45] Xin Yao and Tetsuya Higuchi. Promises and challenges of evolvable hardware. *IEEE Trans. on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, 29(1):87–97, 1999.

[46] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transaction Systems Man. Cybern*, SMC-3(1):28–44, 1973.

[47] L. A. Zadeh. The concept of linguistic variable and its application to approximate reasoning. *Information Science*, 8:199–249, 1975.

[48] R. S. Zebulum, M. A. Pacheco, and Marley Vellasco. Analogue Circuits Evolution in Extrinsic and Intrinsic Modes. *Proceedings of the 2nd International Conference on Evolvable System: From Biology to Hardware. Lecture Notes on Computer Science*, 1478:154–165, 1998.

[49] H. J. Zimmerman. *Fuzzy Set Theory and Its Applications*. Kluwer Academic Publishers, 3rd edition, 1996.

Vita

- Uthman Salem Al-Saiari
- Born in Jeddah, Saudi Arabia on July 10, 1975.
- Received B.S. degree in Computer Engineering from KFUPM, Saudi Arabia in January 1999.
- Completed M.S. degree requirements at KFUPM, Saudi Arabia in November 2003.