Enhanced Simulated Evolution Algorithm For Digital Circuit Design Yielding Faster Execution in a Larger Solution Space

Sadiq M. Sait **Muhammad Al-Ismail**
College of Computer Sciences & Engineering
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
Email: {sadiq, st941659}@ccse.kfupm.edu.sa

Abstract— **Evolutionary algorithms have been studied by several researchers for the design of digital circuits. Simulated Evolution (SimE) is used in this paper due to it simplicity and customizability to combinatorial problems. A tree data structure based circuits are evolved. Thus, a larger solution space is investigated. In addition, a new pattern based goodness measure is presented.**

I. Introduction

Evolutionary algorithms present a new methodology in hardware design and synthesis. It is expected, to some extent, that evolutionary algorithms will lead to the discovery of novel digital circuits in terms of design style, area, power and delay [1] The process of implementing a digital circuit involves transforming the logical specifications using conventional logic rules into a logic functional representation of digital cells. Using evolutionary algorithms to assemble digital circuits out of n variables and then iteratively testing the functional fitness of the assembled circuit will eventually lead to a logically correct representation that a human engineer may have not reached. Figure 1 depicts the principles of evolutionary design solution space as compared to conventional design techniques.

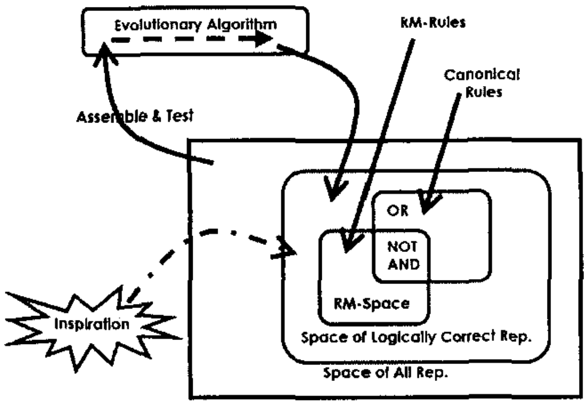

Fig. 1. Evolutionary Design Solution Space

This process was pioneered by de Garis [2] back in 1993 and since then this field of research has received increasing attention [3]. The work presented here is driven by the fact that evolutionary algorithms can be used to explore a much richer design space. In addition, it is also anticipated that the evolved circuits would be better according to an objective function.

II. SimE Background

Simulated evolution -SimE- is conceptually simple and elegant [4]. It is general in the sense that it can be tailored to solve most known combinatorial optimization problems. SimE was proposed by Kling and Banerjee in 1987 which is based on an analogy with the principles of natural selection thought to be followed by various species in their biological environment [5]. During the process of evolution, biological organisms tend to develop features that allow them to adapt to the peculiarities of their environment. Therefore, by adapting, an organism optimizes its chances of survival [4].

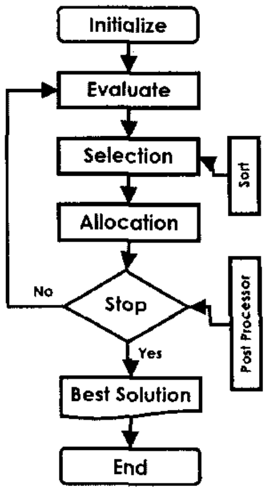

Fig. 2. Simulated Evolution Algorithm

SimE is stochastic because the selection of which components of the solution to change is done according to a stochastic rule. Well located components have a high probability to remain where they are. This probabilistic feature gives SimE a hill-climbing property.

It is not the intention of this paper to give a detailed description of SimE. It is only used to briefly outline the required knowledge to understand the terminology used in the context to this work. Simulated Evolution is very well described in [4]. Figure 2 pictures the basic steps of SimE. The creators of SimE have originally used it to solve the standard cell placement problem [5], hence, the terms selection and allocation. Various operators of SimE are introduced below.

A. *Evaluation*

The evaluation step proceeds by evaluating the goodness of each individual of the population. The goodness measure is a single number in the range $[0, 1]$. The Goodness is defined as follows:

$$g_i = \frac{O_i}{C_i}$$

Where O_i is an estimate of the optimal cost of the individual, and C_i is the actual cost of the individual in its current location. O_i and C_i are problem specific that are customized according to the problem objective.

B. *Selection*

Secondly, the selection step is the decision-maker of SimE. It takes the population as its input and partitions it into two disjoint sets P_s and P_r. The components selected for reallocation are assigned to set P_s. Components are either sorted or not based on a correlation with the problem objective. In other words, the sorting process is problem specific. Figure 3 outlines the basics of the selection procedure. Sorting is thought of as arranging the components in P_s for the allocation operator to act upon them in a meaningful way with respect to the problem objective.

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

Fig. 3. SimE Selection Procedure

The selection procedure makes use of a selection bias which is a fraction in the range of $[-0.2, 0.2]$. Basically, a negative bias means that the goodness measure is loose and more components are needed to be selected every time for reallocation. A well designed goodness measure can normally use a selection bias of zero.

C. *Allocation*

Finally, the allocation procedure which has the most impact on the solution quality obtained by SimE. Members of P_s are mutated according to a function suitable for the problem objective. A number of trial mutations are performed. The mutations that constitute a better solution are made permanent. The post processor is a handy tool that is used to tune the population for the next iteration of the SimE algorithm. For example, it could be used to get rid of the redundant components.

III. Brief Literature Review

There are two existing techniques for logic design. These techniques are divided into conventional logic synthesis and evolutionary logic synthesis. These techniques are briefly described. References to these techniques are provided.

A. *Conventional Logic Synthesis*

Conventional logic synthesis is mainly classified as two-level logic and multi-level logic. The former deals with PLAs and their minimization. A number of heuristics exist for such problems; The Quine-McClusky algorithm [6], [7]. An effective technique used is **ESPRESSO** [8]. The later makes use of factoring and decomposition into sub-functions. **SIS** is a sophisticated tool developed at University of Berkeley for multi-level minimization [9].

RM-Muller XOR logic is a work done for representing functions using only XOR gates and un-complemented variables [10]. XOR based functions are not easily visulized by human designers that is why the XOR gate is of interest to the evolutionary design of logic. It is believed that utilizing XOR gates can lead to new more efficient designs [1].

B. *Evolutionary Logic Design*

A number of researchers have worked on evolutionary logic design. To name a few, Miller [11], Colleo [12], [13], Sarif [14] and Uthman [15]. They have used different existing heuristic such as: Genetic Algorithm, Ant Colony and Simulated Evolution. In this subsection, several logic design specific terminology is explained.

1) Input Redundancy: Inputs which are not used in the operation of the circuit.

2) Functional Redundancy: occurs when number of cells of an evolved circuit is higher then optimal number needed to implement the circuit.

3) Cell Redundancy: when outputs are not connected to any other cells in an evolving circuit.

4) Logic Equivalency: when a sub circuit can be substituted by another equivalent.

5) Phenotype Equivalency: Possibility of having different encoding for a given sub-circuit.

IV. SimE For Circuit Design

The proposed modifications to the current implementation of SimE are discussed in this section. The data structure used and the programming implications are also discussed. SimE operators are explained in details.

A. Problem Formulation

The combinatorial optimization problem is mapped to an assignment problem where given a finite set M of k distinct movable elements and a finite set L of $|L|$ locations. A state is defined as an assignment function $S: M \rightarrow L$

M is said to be the possible gate types of different input configurations. The objective is to start with $|L|$ locations and then iteratively select modules from M to produce a logically correct circuit.The evolutionary algorithm is guided through the search space using a goodness function that is custom made. Table I lists all possible gate types allowed in this experiment. The table should be rich in gate types but yet limited to be able to obtain different structures [14].

TABLE I
CELL OPTIONS

#	Gate	Function
1	AND	$A \cdot B$
2	OR	$A + B$
3	NAND	NOT($A \cdot B$)
4	NOR	NOT($A + B$)
5	XOR	$A \oplus B$
6	XNOR	$A \odot B$
7	Buffer1	Input 1
8	Buffer2	Input 2
9	NOT1	NOT(Input 1)
10	NOT2	NOT(Input 2)

B. Circuit Encoding

Most of current implementations of the evolutionary design of digital circuit adopt a 2-D matrix representation which has been suggested by Coello [12]. This structure has been proven to be capable of evolving functionally correct circuits in number of research papers so far [12], [13], [14], [15].

In this experiment a tree data structure is used as it is our belief that a logical circuit is relatively similar to a tree in its structure. Most researchers start with a matrix of size $(n \times n)$ where n is the number of inputs of the required circuit.

Now, having a digital circuit mapped onto a tree introduces several differences as opposed to a 2-D matrix. Rows and columns are used in a matrix while a tree has a parent-child relationship and a node level. Each node has left and right children.

Inspired from the 2-D matrix used by other researchers, the root level of the tree is set to n where n is the number of inputs. None the less, in order to facilitate the search in the design space a number of redundant cells are also needed. Therefore, $(n \times n)$ is the initial number of cells, the evolutionary algorithm starts with. Figure 4 pictures an example of a circuit of 4 inputs.

In this experiment, a node is allowed to have any other node as one of its children provided that it is at a lower level. Hence, a root node for example might have one or both of its children as an input pad.

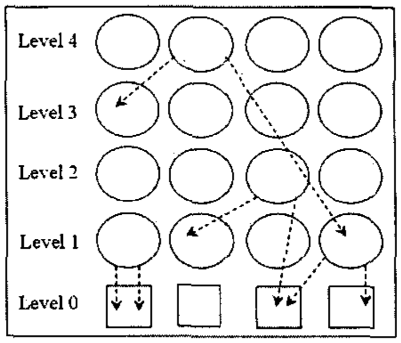

Fig. 4. Tree Data Structure of 4 Inputs Circuit

The proposed tree data structure allows the search in by far a larger design space. The beauty of the tree data structure is the ability to evaluate the circuit according to an in-order-traversal mechanism, i.e., recursively evaluating the goodness of each component. Therefore, reducing the execution time by a great deal as it is going to be shown later. Figure 5 shows how in-order-traversal has been used to speed up the evaluation of each node.

```
**Begin** *InOrderTraversal(node n)*
    **If** (Left.Type($n$) != Input) **Then**
        InOrderTraversal(Left($n$));
    **EndIf**
    **If** (Right.Type($n$) != Input) **Then**
        InOrderTraversal(Right($n$));
    **EndIf**
    Evaluate($n$)
**End** InOrderTraversal;
```

Fig. 5. In Order Traversal Procedure

C. Cell Encoding

Cells constitute the building blocks of the circuit. The encoding of each cell is crucial as cells (genotypes) are used to form the circuit (phenotype). Due to the use of a tree data structure the triplet suggested by Coello in [12] is modified to Figure 6. In other words, instead of using input1 and input2 as part of the triplet, references to the left and right children are used in order to form the tree.

Left	Right	Gate Type

Fig. 6. Cell Encoding

Not to violate the proposed tree data structure, gate types 7-10 as shown in Table I have been suggested. Buffer 1, for example, will have the cost of a WIRE from the MOSIS [16] library while it has two inputs. Hence, a post-processor that understands such peculiarities of the problem is used.

D. Fitness Calculation

In this section the pattern based goodness measures used in [14], [15] are presented. A modified goodness measure used in this experiment is also discussed.

In this experiment only single output circuits have been studied. Area, power and delay optimization has not been considered yet. Two functional goodness measures have been proposed. Each goodness measure is between 0 and 1.

A cell whose goodness is 1 produces all possible minterms. A cell relays on other cells in order to produce more minterms. Therefore, the functional goodness measure is divided into interior and exterior.

1) Interior Goodness Measure: The interior goodness measure poses the question: how good is this cell? The interior goodness measure is a function of the truth table length and the number of minterms generated by the cell. The truth table length equals to 2^n where n is the number of inputs. The number of matching minterms -output patterns- is denoted by P.

$$g_i = \frac{P_i}{2^n}$$

Table II shows possible interior goodness measure values for a 3 inputs circuit. At most, a 3 inputs circuit will have 8 minterms at its output. Consider the case where it produces 0 minterms, it means that by inverting the output signal the cell's goodness will jump to 1.

TABLE II
DIFFERENT INTERIOR GOODNESS MEASURES

#	Minterms	0	1	2	3	4	5	6	7	8
1	Goodness	0	1/8	2/8	3/8	4/8	5/8	6/8	7/8	1
2	[14], [15]	1	7/8	6/8	5/8	4/8	5/8	6/8	7/8	1
3	MI SimE	1	6/8	4/8	2/8	0	2/8	4/8	6/8	1

When the functional goodness is less than 0.5, the cell is inverted [14], [15]. It is denoted by the term normalized functional fitness as in [14], [15]. Row 2 enumerates possible goodness values using the normalized functional fitness.

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

Note that the normalized function fitness proposed by [14], [15] has 0.5 as its worst goodness. Due to this scaling of the fitness function, it is more difficult to evolve large circuits as cells would have less variation in their goodness making it harder for the SimE algorithm.

In this experiment, an improvement was made to the goodness measure. Cells that generate half of the truth table are given a goodness of zero. Row 3 in Table II demonstrates this principle. Below is a formulation of the new proposed goodness measure. The newly proposed goodness measure is more sensitive to changes in the circuit utilizing a larger scale. This feature allows the evolution of larger circuits.

$$FF_n = [(Max\{FF, 1 - FF\}) - 0.5] \times 2$$

The interior goodness measure is used by the allocation operator to favor cells with respect to others when performing mutations.

2) Exterior Goodness Measure: The exterior goodness measure poses the question: how good is a cell to other cells in its current location? The exterior goodness measure is denoted by G. Figure 7 shows the process of calculating G for a cell i whose input is cell j. The exterior goodness is divided into two instances: when a cell depends on its inputs or when other cells depend on it to generate more functional patterns.

Exterior goodness calculation is done breadth-first. The exterior goodness measure is used by the selection operator to favor cells with respect to others when partitioning. In other terms, when a cell has low exterior goodness, it means that this particular cell is not desired by other neighboring cells in the current solution.

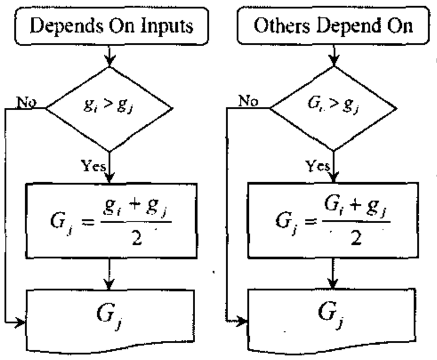

Fig. 7. Exterior Goodness Calculation

E. Allocation and Mutations

The allocation procedure is the most vital process of the evolutionary algorithm. Allocations are split into local and global allocations -mutations-. A number of trial mutations are carried on the cells selected by the selection operator; this number is denoted by W. Moreover, W is relative to the number of gate types m and circuit size. In addition, the gates used in this experiment make use of 2 inputs. Therefore, W is determined using the following formula

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

F. Effect of Trial Mutations

Since the allocation operator performs W trial mutations per cell every time the allocation operator is invoked, a study of W is needed. Table III lists the W values for different circuits based on the number of inputs.

Throughout this experiment m was set to 6 as this will cover all of the possible gate types, i.e., AND, OR, NAND, NOR, XOR and XNOR, refer to Table I. More gate types increases the complexity of the problem.

The total number of trials per mutation call grows almost exponentionally. In Figure 8, the number of possible gates is

TABLE III
W TRIAL MUTATIONS VS. CIRCUIT SIZE

n	m	W	Cells	Total
2	6	6	4	24
3	6	18	9	162
4	6	36	16	576
5	6	60	25	1500
6	6	90	36	3240
7	6	126	49	6174
8	6	168	64	10752
9	2	72	81	17496
10	6	270	100	27000

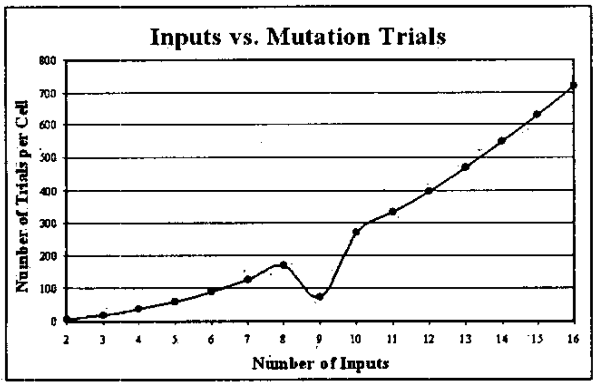

Fig. 8. Trials with Only 2 Gate Options at 9 Inputs Circuit

reduced to 2 for circuits of 9 inputs. The effect is a great reduction in the complexity of the problem.

To sum up, the number of cells allowed in this experiment has been set to a fixed number 6 while it is better to adaptively change that number during execution. In order to be able to intelligently change the number of allowed cells, the algorithm has to have a prior knowledge of the required circuit. This suggests the use of a constructive algorithm prior to the use of the evolutionary algorithm for SimE to be structure-aware. Another problem is which gates to allow and not to allow. It is believed that Tabu search could be utilized for this matter [4]. Tabu search makes use of short, intermediate and long-term memories which if properly tailored to the problem; a great reduction in complexity is possible.

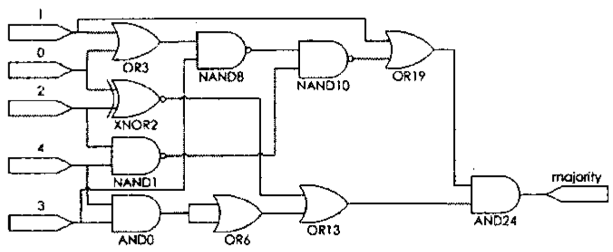

Fig. 9. Evolved Circuit Example (majority)

V. RESULTS AND COMPARISON

A number of techniques have been proposed in the literature. Genetic Algorithm has been used in [12]. Ant colony system has been used in [13] and later modified in [14]. Sait has proposed two SimE algorithms where his SimE-G2 has shown the best results so far [15]. 14 Single output circuits have been evolved and compared to the current results obtained by others. Area has been calculated for the output circuit yet it has not been optimized for. None the less, results in Table IV are very comparable to the current results by other researchers while some circuits have been shown to have less area. The gates library used in this experiment is CMOS 0.25μ obtained from MOSIS [16].

Run time has been shown to outperform the current implementations. It is owed to the sophisticated tree data structure used. The in-order-traversal procedure recognizes the redundant cells and only spends time evaluating the required portion of the tree. For run time comparison see Table V.

TABLE IV
AREA COMPARISON (MICRON)

Circuit	n	GA [12]	ACO [13]	MACO [14]	SimE-2 [15]	SimE MI
circuit1	4	18954	16767	14094	12879	15066
circuit2	4	21870	16767	14823	13122	15066
circuit3	4	19926	14823	10692	10752	11907
circuit4	2	1458	1458	1458	1458	1458
circuit5	4	27945	29889	11664	13870	12879
circuit6	4	40338	13365	10935	10935	12393
circuit7	4	83835	18954	12636	12393	8019
circuit8	4	75087	7776	7290	7776	7290
Sarif6	3	9477	9477	7776	-	7776
Sarif7	4	17496	19197	12393	-	15309
Sarif8	4	15552	18468	14337	14337	15066
majority	5	21141	-	13851	13851	19440
xor8	8	32805	-	20655	20655	20655
xor9	9	35266	-	23328	23814	24300

VI. FUTURE DIRECTIONS

There is more work to be done in order to improve the performance of the evolutionary algorithm. The future work involves the further investigation of:

1) Incorporating area, power and delay Multi-objective optimization along with the functional constraint.
2) New goodness measures that are structure-aware rather than pattern-based.
3) Evolving multiple output functions.
4) Use of a constructive algorithms for initialization.
5) Adaptive cell choices.
6) Adaptive number of levels.

TABLE V
TIME COMPARISON (TIME)

Circuit	n	GA [12]	ACO [13]	MACO [14]	SimE-2 [15]	SimE MI
circuit1	4	91.66	61.4	15.27	3.12	3.062
circuit2	4	102.3	61.4	15.17	3.77	3.062
circuit3	4	155.7	49.7	14.67	4.57	11.51
circuit4	2	275.1	0.4	3.5	0.48	0
circuit5	4	266.3	76	16.97	6.51	5.016
circuit6	4	-	50.7	14.83	6.02	0.125
circuit7	4	-	75.6	16.1	7.44	0.14
circuit8	4	-	50.4	12.7	5.02	0.109
Sarif6	3	-	-	-	-	0.125
Sarif7	4	-	-	-	-	7.93
Sarif8	4	91	-	-	11	1.75
majority	5	6290	-	-	15.32	318.2
xor8	8	7430	-	-	220.43	10.85
xor9	9	10857	-	-	231	6.546

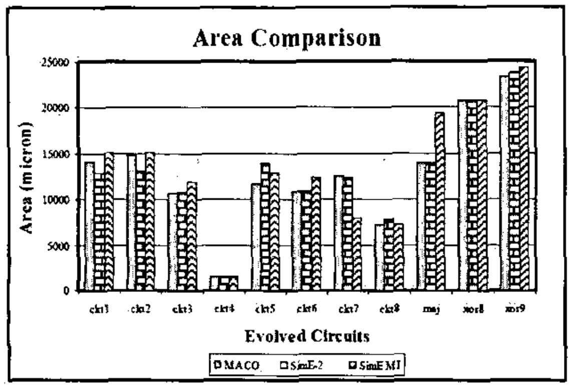

Fig. 10. Area Comparison (micron)

VII. CONCLUSION

The modified simulated evolution algorithm has the capability of searching a larger design space yet in a shorter execution time. Circuits having a more complex structure have been evolved. The tree data structure implementation mimics the interconnections used in a digital circuit. Recursively programming the evaluation procedure using in-order-traversal has successfully reduced the execution time. More work is needed for the algorithm to handle larger multiple output circuits.

ACKNOWLEDGMENT

The authors would like to acknowledge the support received from King Fahd University of Petroluem and Minerals under the project entitled "Iterative Heuristics for the Design of Combinational Logic Circuit".

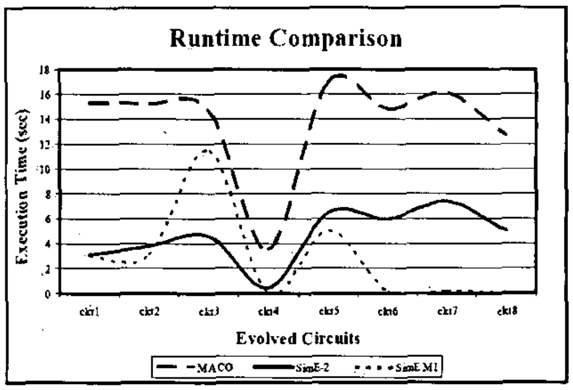

Fig. 11. Run Time Comparison (sec)

REFERENCES

[1] 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*, vol. 1, no. 1, pp. 8–35, 2000.
[2] Hugo de Garis, "Evolvable Hardware: Genetic Programming of a Darwin Machine," *Proceedings of the International Conference in Innsbruck, Austria*, vol. 2, no. 4, pp. 441–449, Springer-Verlag, 1993.
[3] J. F. Miller and P. Thomson, "Discovering Novel Digital Circuits Using Evolutionary Techniques," *IEE Colloquium on Evolvable Systems, Savoy Place, London*, March 1998.
[4] Sadiq M. Sait and H. Youssef, *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*, IEEE Computer Society Press, 1999.
[5] R. M. Kling and P. Banerjee, "ESP: A New Standard Cell Placement Package using Simulated Evolution," *Proceeding of 24th Design Automation Conference*, pp. 60–66, 1987.
[6] W. Quine, "The Problem of Simplifying Truth Functions," *American Mathematical Monthly*, vol. 59, pp. 521–531, 1952.
[7] E. McCluskey, "Minimisation of Boolean Function," *Bell System Technical Journal*, vol. 38, pp. 1417–1444, 1956.
[8] R. Brayton, G. D. Hachtel, C. T. McMullen, and A.L. Sangiovanni-Vincentelli, *Logic Minimisation Algorithms for VLSI Synthesis*, Kluwer Academic Publisher, 1984.
[9] 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.
[10] D. Green, *Modern Logic Design*, Addison-Wesley, Reading, MA, 1986.
[11] 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*, pp. 105–131, 1998.
[12] 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*, vol. 2, no. 4, pp. 299–314, June 2000.
[13] C. A. Coello, A. D. Christiansen, and A. H. Aguirre, "Ant Colony System for the Design of Combinational Logic Circuits," *Evolvable Systems: From Biology to Hardware, Edinburgh, Scotland*, pp. 21–30, April Springer Verlag, 2000.
[14] Sadiq M. Sait, Mostafa Abd-El-Barr, Uthman Al-Saiari, and Bambang A. B. Sarif, "A Modified Ant Colony Algorithm for Evolutionary Design of Digital Circuits," *IEEE Congress on Evolutionary Computation, Canberra*, pp. 708–715, July 2003.
[15] Sadiq M. Sait, Mostafa Abd-El-Barr, Uthman Al-Saiari, and Bambang A. B. Sarif, "Digital Circuit Design through Simulated Evolution (SimE)," *IEEE Congress on Evolutionary Computation, Canberra*, pp. 375–381, July 2003.
[16] "Standard Cell Library for MOSIS CMOS, http://www.mosis.org/Technical/Designsupport/std-cell-library-scmos.html," .