

A New Hybrid Genetic-Based Reduction Method in Nanoelectronic Circuits

Mohammad Reza Bonyadi

Department of Computer Engineering, ECE Faculty, Shahid Beheshti University G.C., Tehran, Iran

Abstract: In this paper, a new hybrid genetic algorithm is proposed to reduce the gate count in majority-based circuits. To initialize the GA, a method based on Quine-McCluskey is used which accelerates the reduction procedure. Also, a new approach for finding the constituent function set for an N variable logical function is proposed which works based on the hamming distance. This approach enables us to find the constituent functions among all feasible functions and helps to reduce the space of reduction problem. The improvement of the proposed method for reducing the gate counts was satisfactory (20% improvement regards to related works) while the number of needed iterations for the reduction procedure was highly decreased according to other GA methods.

Key words: Majority · Genetic Algorithm · Logic Reduction

INTRODUCTION

There are several candidates for post-CMOS IC designs such as quantum cellular automata (QCA), tunneling phase logic (TPL) and single electron tunneling (SET) [1, 2]. The basic building block of QCA circuits is majority gate, hence, efficiently constructing QCA circuits using majority gates has received significant attentions by researchers [2, 3]. This efficiency back to the number of needed gates to construct a given function, i.e., uses less number of majority gates and inverters to manipulate a function. Related study goes back to 1960s [4, 5]. In [5], A reduced-unitized-table-based synthesis method was given and a Karnaugh-map (K-map)- based method was presented in [5] which resulted in an exponential number of majority gates. These methods are very useful and suitable for synthesizing small networks by hand and their high complexity does not allow them to be exerted in large instances. Some majority logic reduction methods have been proposed recently [6, 3], which are based on a three-cube method developed from a geometric interpretation of the Boolean function. These methods are still manual. Also, some other works which are automatic multilevel multi-output synthesis tool has been developed based on these methods [7, 8]. The first attempt to synthesis the QCA based circuits based on the Genetic Algorithm refers to 2007 [9]. This work was the pioneer one which exerts the GA to simplify Boolean

functions based on majority and inverters. The Genetic Algorithm (GA) is an optimization method that has been used widely in many optimization problems like logic reduction. In designing a combinational logic circuit, one of important objective is to minimize the number of logical gates used. Many researchers had come up with various methods to improve designing process including Genetic Algorithm (GA). As an example, in 1999, Carlos et.al uses a genetic algorithm to design digital logic circuit with number of gate counts constraint [10]. By using the Genetic Algorithm (GA), a reduction method is presented in this paper that minimizes the number of both majority gates and inverters. The optimizer is implemented in C#.net (the visual studio 2005 environment) and the experimental results are obtained and compared with recent available results in [3, 9, 6, 7]. The proposed method can be easily extended to minority-based circuit such as TPL and SET. Also, a new method to generate the standard functions [3] is proposed which exploits the Hamming distance between the min-terms to prepare the standard functions. For more information on standard functions see [3, 11, 12].

The Paper Organized as Follows: The next section contains backgrounds about Quantum Dot Cellular (QCA), Genetic Algorithm and Three-Cube Representation. Afterward, the proposed approach has been presented in section III. In this section, at first,

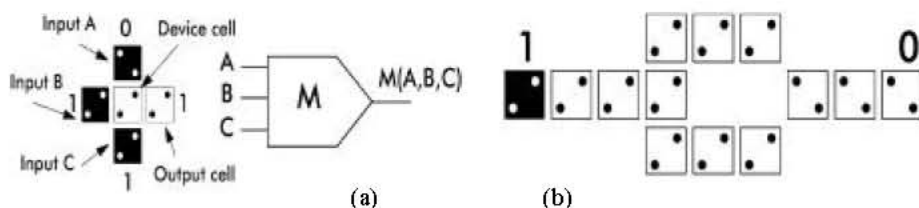


Fig. 1: (a) A QCA majority gate (b) A QCA inverter

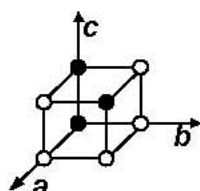


Fig. 2: $F = a'b'c + abc + a'b'c$, the black points are on-set points and the white points are the off-set points

a new approach to construct the standard functions has been explained. The section will be continued with the proposed reduction method based on GA and describes its fundamental operations. The simulation results have been compared with the previous works in section IV. Finally, the last section summarizes our conclusion remarks.

Principles and Fundamentals

Basic Operators in Quantum Dot Cellular: Any QCA circuit can be built using only majority gates and inverters. The Fig. 1 (a) shows a QCA gate that implements the majority function.

$$M(A,B,C) = AB + BC + AC \tag{1}$$

As it is seen in Fig. 1 (a and b), each QCA majority gate requires only five QCA cells and every QCA inverter gate can be implemented by 13 quantum cells. As it is shown in Eq. (2a) and (2b), the AND gate and an OR gate can be constructed using majority function by fixing the polarization of one input to a constant logic '0' or logic '1'. Hence, QCA circuit is based on majority gate-based circuits instead of AND/OR/Inverter gate-based circuits [1,2].

$$M(A,B,0) = AB \tag{2a}$$

$$M(A,B,1) = A+B \tag{2b}$$

Therefore, reducing the number of both majority gates and inverters is to be considered in the logic synthesis of QCA circuits.

Logic reduction using the majority and inverters is much more complex than that of Boolean logic [5]. In fact, the classical methods are not adequate to be used for reducing the number of basic operators in majority/minority logic. There for, new reduction methods are needed to do so [1, 2]. Some recent works are [2, 5, 9].

Genetic Algorithm: Genetic Algorithms [7, 8] (GAs) is as a possible solution for many search and optimization problems. In fact, GAs are evolutionary algorithms which imitate the behavior of the nature to solve problems. Each solution (individual) is represented as a string (chromosome) of elements (genes) which has a fitness value based on the value given by an evaluation function. The fitness value measures how close the individual is to the optimum solution. A set of individuals considered as a population that evolves from one generation to the next through the creation of new and the deletion of some old individuals. The process starts with an initial population created in some way, e.g., randomly.

Evolution Can Take Two Forms

Cross-Over: The chromosomes of two individuals are combined to gain a new individual for insertion in the population to replace another individual. The individuals are selected by techniques ensuring that the selection probability of each individual is proportional to its fitness. New individuals are thus likely to have a higher fitness than those they replace.

Mutation: A gene of a selected individual is randomly changed. This provides additional chances of entering unexplored sub-regions and also holding the method from getting stock on local optima. Evolution is stopped when either the goal is reached or a maximum CPU time has been spent.

Three-Cube Representation of Three-Variable Boolean Functions: According to [1, 2], a Boolean function of variables can be represented by a binary n-dimensional hyper cube. Each dimension of the hyper cube is corresponds with exactly one literal and the coordinates

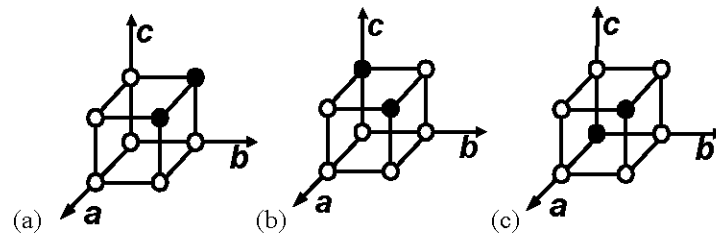


Fig. 3: Three possible 3 variable 2 min-terms functions

are corresponds with the Boolean value of that literal. In [3], the method in [1, 2] is applied to three-variable Boolean functions in the three-cube domain. For Boolean functions of three binary variables (a, b, c), there are 2^3 distinct min-terms each correspond to a vertex (point) of the three cube. Figure 2 illustrates an example of cube presentation for a three variable 5 min-terms Boolean function.

It is obvious that there are 256 Boolean functions for three variables, but they can be mapped to only 80 equivalence classes by permuting their inputs [11, 121] to produce the same functionality. In [3], a symbolic mapping in combination with discarding the functions with more than 4 min-terms was considered that could simplify the mentioned 80 functions to 13 standard functions. The introduced mapping was based on changing the name of variables which means the authors considered that a three-variable Boolean function is defined by three variables called A, B and C each of them can be mapped into any of a, b, c, a', b' and c' (a' is the inverse of a). In this case, they truly claim that there is no difference between the function $F=abc'+ab'c'+a'bc$ and $F=a'bc+a'bc'+ab'c$ (the mapping is $a \rightarrow b, b \rightarrow c$ and $c \rightarrow a$) in terms of simplification. They use the cube representation to show their definitions. As an example, all the three-variable Boolean functions that contain 2 min-terms are in one of the following groups: 1) two adjacent points (one edge); 2) two nonadjacent points, but in one face; and 3) two nonadjacent points and not in one face. (Figure 3).

Using this method, they could find 13 standard three-variable Boolean functions that covered all the three-variable Boolean functions, i.e., these standard functions can map to all feasible functions in the space of three-variable Boolean functions.

Proposed Method: In this section, an approach for finding the N-Variable Boolean standard functions is presented. Then, a Genetic based simplification algorithm is proposed and its various parts consists of chromosome

structure, reproduction operators, fitness function and initialization method are introduced.

N-Variable Standard Functions (CFSs for NVKMBF):

As it was mentioned in section II.C, the cube representation could help to find 13 standard functions (in this paper we call them as Constituent Functions Set (CFS)) which can cover all the space of three-variable Boolean functions. Consider the CFS for N-variable K Min-term Boolean Functions (in this paper we call them NVKMBF: N-Variable K Min-terms Boolean Functions) is required. In this case, the cube base approach is not useful because the corresponding hyper-cube cannot be imagined when N is more than three. Here, we propose an approach based on Hamming distance to find the CFS in NVKMBF space.

As it was put, there are three groups of functions on a cube in cube representation for 3V2MBF: 1) both min-terms are adjacent, 2) The min-terms are in one face of the cube and non adjacent and 3) the min-terms are non adjacent nor in one face. On the other hand, the min-terms of a 3V2MBF that the Hamming distance between its min-terms is one are adjacent. If the Hamming distance between its min-terms is two, its min-terms are in one face of the cube and if this distance is three, its min-terms are not adjacent nor on one face of the cube. This definition can help to implement an algorithm to find CFS in each multi-variable function.

Here, we introduce a definition called Hamming Distance Set (HDS). An HDS for an NVKMBF is a set of hamming distances between its min-terms. As an instance, the HDS for the function $F=a'bc+ab'c'+abc$ is {3, 2, 1}. This set shows that the hamming distance between the first and second min-terms in Fig. 3, between the second and third is 2 and between the third and first is 1. If two functions with equal number of variables and min-terms have the equal HDS, they correspond to each other in the cube representation. Note that the permutation in their HDS is not important.

Table 1 number of all possible functions with 4 variables

# of Variables	# of Min-terms	All possible functions	# of CFS
4	1	$C(2^4, 1)=8$	1
4	2	$C(2^4, 2)=120$	4
4	3	$C(2^4, 3)=560$	6
4	4	$C(2^4, 4)=1820$	15
4	5	$C(2^4, 5)=4368$	19
4	6	$C(2^4, 6)=8008$	26
4	7	$C(2^4, 7)=11440$	27
4	8	$C(2^4, 8)=12870$	32
Total	39194	130	

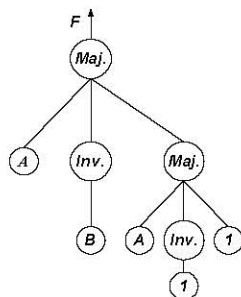


Fig. 4: a chromosome which indicates the function: $M(A, B', M(A, 0, 1))$

As an example, the number of all feasible functions with 3 variables and 3 min-terms is 56 ($C(2^3, 3)=56$ where C is combination operator). In sum, one can attain the CFS for NVKMBF by choosing the functions which their HDS are not equal. A program in C# was implemented which provides CFS for NVKMBF using the presented method. As an instance, Table 1 illustrates the CFS's for 4VKMBF:

It is obvious that the number of 4V3MBF is $C(2^4, 3) = 560$ which just 6 of them are needed to build all other 554 functions using a simple mapping. Hence, simplifying these 6 functions result in simplifying all possible functions in the form 4V3MBF.

Proposed Hybrid Genetic Based Reduction Algorithm:

In this section, a hybrid genetic based algorithm is proposed for reduction in nano-electronic majority based circuits. At first, the various parts of the genetic algorithm are introduced which consist of the chromosome structure, proposed reproduction operators, fitness function and proposed population initialization. Then, the algorithm is presented step by step.

Chromosome Structure: In this paper, a tree structure for the chromosomes is considered. This structure was firstly used for majority based circuits in [6].

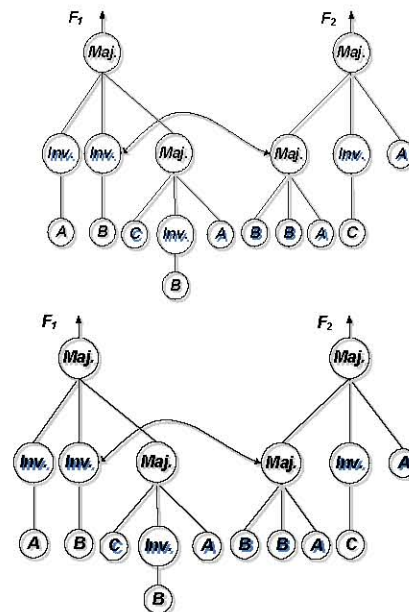


Fig. 5: Crossover operator changes the sub-trees of selected chromosomes

In this structure, each chromosome is a tree which implements a circuit. Each node of the tree is a literal or function. The internal nodes are functions consist of Inverters and Majorities while the external nodes are the function variables or constants ("0" and "1"). Figure 4 illustrates this structure for a circuit.

Reproduction Operators (Crossover and Mutation):

There are two major operators for GA: Crossover and Mutation. The crossover operator for the mentioned structure works as follow: Two parents are selected from the population via a selection process. These parents should combine to each other and produce two new chromosomes (children). In this paper, a sub tree of each parent is selected and replaced with each other. The sub trees are selected using a random process. Figure 5 illustrates this process.

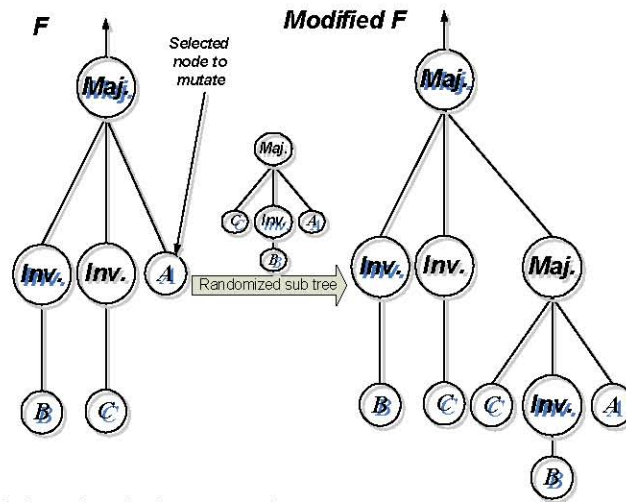


Fig. 6: Mutation operator which randomly change a sub-tree

In each iteration, the crossover operator is performed according to the predefined probability P_c . The pseudo code for cross over is presented in Algorithm 1.

Algorithm 1: Crossover

Input: two chromosomes C_1 , C_2 , Output: two modified chromosomes C_1 , C_2

- Select node N_1 in C_1 and node N_2 in C_2 randomly
- Replace the sub tree of node N_1 in C_1 with N_2 and the sub tree of node N_2 in C_2 with N_1
- Update the information in C_1 and C_2
- Return the Modified C_1 and C_2 as two new chromosomes

The Mutation Operator Is as Follow: A sub tree of a chromosome is discarded and a new generated sub tree is replaced with that. This process is performed in each iteration and each chromosome is mutated via the mentioned process with probability P_m . Figure 6 illustrates this process.

The pseudo code for Mutation is denoted in Algorithm 2:

Algorithm 2: Mutation

Input: A chromosome C , Output: Modified Chromosome C

- Select a node N randomly in chromosome C
- Build a sub tree S using the valid operators (Majority, Inverter, Variables,...) via a random process
- Replace N with S and update the parent information in C for new sub tree
- Return C as the modified chromosome

Fitness Function: The fitness function is a function that shows the virtue of a chromosome. Here, the fitness in [6] is used.

$$Fit(C_i) = \frac{n(C_i, F)}{2^k} \quad (3)$$

Where C_i is the i^{th} chromosome and F is the expected logical function. $n(C_i, F)$ is the function that counts the number of equivalent min-terms between the implemented logical function by C_i and F . If the fitness of i^{th} chromosome is equal to 1.00 (that means the chromosome implements the desired circuit completely), we use:

$$Fit(C_i) = Fit(C_i) + \frac{1}{\text{Number of nodes in } C_i} \quad (4)$$

This function can classify the chromosomes according to both similarity to the expected logical function and number of nodes. The value of fitness function will be increased when the number of nodes is decreased. Also, further works on the fitness function is possible to reduce the number of levels and other primitive operators.

Population Initialization: In the proposed method, a heuristic approach is considered for initializing the population. Here, one of the chromosomes in initial population is produced via a process which implements an adequate circuit. In fact, the SOP of the given circuit is simplified using the Karnaugh-map and Quine-McCluskey procedure and then its corresponding majority expression is produced using Eq.(2a) and Eq.(2b). The remaining chromosomes are made via a random process. The process for producing the population is presented in Algorithm 3.

Algorithm 3

Input: the SOP of the given circuit, output: Population

- Simplify the SOP (Sum of Products) of the given function using one of the Karnaugh-map or Quine-McCluskey
- Convert the simplified result into the majority based expression using Eq (2a) and Eq (2b)
- Build the tree structure of the attained majority based structure and add this tree to the population as a new chromosome.
- Produce the remaining chromosomes via a random process.

Proposed Algorithm: Algorithm 4 illustrates the proposed hybrid genetic based algorithm for reducing a give SOP for a desired circuit *M*.

Algorithm 4: Simplify the circuit *M* with the given SOP

- Initialize the parameters *Pm*, *Pc*, Population size and iteration number.
- Initialize population according to section 3.iv
- While stopping criteria is not emerged
- Perform cross over and mutation and replace the best children with worst chromosomes from the population
- Calculate the fitness of chromosomes according to *M* and report the best fitness
- End while
- The best chromosome is the best circuit found by the algorithm

The algorithm is stopped according to predefined stopping criterion that is the number of iterations in this paper.

Simulation Results: In this section, the results are compared with some recently proposed algorithms. The proposed algorithm is implemented in C#.Net in visual studio 2005 environment. The implemented algorithm is run on a personal computer with 2 GB of RAM and 1.66 GHz Core 2 Duo mobile CPU. The results are compared with some new methods cited in [3, 8, 9]. In addition, some CFS's for 4VKMBF and 6VKMBF are reduced using the proposed algorithm. All experiments are performed under the following conditions:

Pm=0.2, *Pc*=0.9, Population size: 70, Max iterations: 500

The value of *Pm* and *Pc* was set according to several experiments. Table 2 shows the results in comparison with [3] according to 13 standard functions that were defined in that work.

The table shows that the proposed method find some majority expressions for 13 standard functions which are more simple than that cited in [3] in terms of number of majorities and inverters. For example in function 11, the proposed method found the majority expression which has one majority and one inverter less than that cited in [3]. The improvement in this case was around 6% in terms of number of majorities and 37% in terms of number of inverters. Table 3 shows the comparison results between the proposed method and [8].

Table 2 the comparison results between the proposed method and [3]

Rank*	Function	Previous Works [3]		Our Approach			
		Majority Expression	M	I	Majority Expression	M	I
8	A	M(A,0,1)	1	0	A	0	0
2	AB	M(A,B,0)	1	0	M(A,B,0)	1	0
1	ABC	M(M(A,B,0),C,0)	2	0	M(M(A,C,0),B,0)	2	0
12	AB+A'B'	M(M(A,B,0),M(A',B',0),1)	3	2	M(1,M(B,A,0),M(A,B,1)')	3	1
5	AB+BC	M(0,B,M(A,C,1))	2	0	M(0,B,M(A,C,1))	2	0
10	AB+B'C	M(M(A,B,0),M(B',C,0),1)	3	1	M(A,M(B,C,1),M(A,B,1)')	3	1
6	AB+A'B'C	M(M(A',B,1),M(A,B',C),M(A,B,0))	4	2	M(M(A,B,0),M(A,B,1)',M(A,C,1))	4	1
3	ABC+AB'C'	M(M(A,B,C'),M(A,B',C),0)	3	2	M(A,M(0,B,C),M(B,C,1)')	3	1
4	ABC+A'B'C'	M(M(A',B,1),M(B',C',0),M(A,C,0))	4	3	M(M(C',A,1),M(C,B,0),M(A,B,1)')	4	2
7	ABC+A'BC'+AB'C'	M(M(A,B,C'),M(A,B',C),M(A',B,0))	4	3	M(M(M(A,B,C),1,A)',C,M(A,B,C'))	4	2
13	ABC+A'B'C'+AB'C'+A'BC'	M(M(A',B,C),M(A,B',C),C')	3	3	M(C,M(A,B,C'),M(A,B,C)')	3	2
11	AB+BC+A'B'C'	M(M(B,M(A,C,1),0,M(A',M(B',C',0),0),1)	5	3	M(M(1,B,C'),M(B,1,A)',M(A,C,B))	4	2
9	AB+BC+AC	M(A, B, C)	1	0	M(A, B, C)	1	0
Total number of gates			36	19	Total number of gates	34	12

*The Rank is the number that were used in [3] for the standard functions, M is number of Majorities and I shows number of Inverters

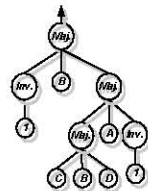
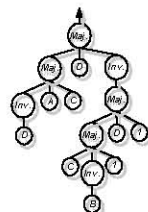
Table 3: Comparison results between the proposed method and [7]

Function	Previous Works[7]		Our Approach		Improvement			
	Majority Expression	M	I	Majority Expression	M	I	M	I
A	M(A,0,1)	1	0	A	0	0	1	0
AB	M(A,B,0)	1	0	M(A,B,0)	1	0	0	0
ABC	M(M(A,B,0),C,0)	2	0	M(M(A,C,0),B,0)	2	0	0	0
A+B	M(A,B,1)	1	0	M(A,B,1)	1	0	0	0
A+BC	M(M(B,C,0),A,1)	2	0	M(M(1,A,C),A,B)	2	0	0	0
AB+A'B'	M(M(A,B,0),M(A',B',0),1)	3	2	M(1,M(B,A,0),M(A,B,1)')	3	1	0	1
AB+BC	M(0,B,M(A,C,1))	2	0	M(0,B,M(A,C,1))	2	0	0	0
AB+B'C	M(M(A,B,0),M(B',C,0),1)	3	1	M(A,M(B,C,1),M(A,B,1)')	3	1	0	0
AB+A'B'C	M(M(A',B,1),M(A,B',C),M(A,B,0))	4	2	M(M(A,B,0),M(A,B,1)',M(A,C,1))	4	1	0	1
ABC+AB'C'	M(M(A,B,C'),M(A,B',C),0)	3	2	M(A,M(0,B,C),M(B,C,1)')	3	1	0	1
ABC+A'B'C'	M(M(A',B,1),M(B',C',0),M(A,C,0))	4	3	M(M(C',A,1),M(C,B,0),M(A,B,1)')	4	2	0	1
A+B+C	M(M(A,B,1),C,1)	2	0	M(1,B,M(A,C,1))	2	0	0	0
A+BC'+B'C	M(M(A,B,C'),M(A,B',C),1)	3	2	M(M(A,B,C)',A,M(B,C,1))	3	1	0	1
AB+A'B'+AC	M(M(A,B',1),M(A',C,1),M(A,B,0))	4	2	M(1,M(A,B,1)',M(A,0,M(B,1,C)))	4	1	0	1
AB+BC+A'B'C	M(M(A,B,C),M(A',B,1),M(B',C',0))	4	3	M(M(A,B,1)',B,M(A,C,1))				

Table 4: Comparison results in terms of number of needed iterations between the proposed method and [6]

Function	GA based in [6] (maximum needed iterations)	Proposed method (maximum needed iterations)
A	100	80
AB	100	75
ABC	100	85
A+B	100	100
A+BC	100	100
AB+A'B'	500	250
AB+BC	500	150
AB+B'C	1000	650
AB+A'B'C	3000	550
ABC+AB'C'	1000	450
ABC+A'B'C'	5000	500
A+B+C	100	100
A+BC'+B'C	1000	550
AB+A'B'+AC	4000	900
AB+BC+A'B'C	1000	750
AB'+BC'+A'C	5000	750
ABC+AB'C'+A'BC'	5000	650
B+BC+AC+A'B'C'	5000	900
ABC+AB'C'+A'B'C+A'BC'	2000	850

Table 5: Reduction results for two 4 variables functions

Function	Majority expression produced by the proposed method	Output file of the program
abc'd+ abcd'+ abcd	M(M(M(B,C,D),A,0),B,0)	
ab'cd+ abc'd'+ abcd	M(M(A,C,D'),M(M(C,B',1),1,D)',D)	

In this case, the improvement was around 4% in terms of number of majorities and 55% in terms of number of inverters. The following table shows the comparison results between the proposed method and the method cited in [9]. The method in [9] is a genetic based method and its results are the same as the proposed method in terms of majority expression but the number of iterations has been improved.

It is obvious from Table 4 that the proposed algorithm could find adequate solutions in less iterations in comparison with [9]. Table 5 illustrates the results of simplification using the proposed algorithm for some CFS's for 4V3MBF.

CONCLUSION

In this paper a hybrid method based on the Genetic Algorithm to reduce the number of gates in majority-based circuits was proposed. A method for initializing the chromosomes was proposed which accelerate the convergence properties of the GA. Also, a new approach based on the hamming distance was introduced that helped to find the constitutive functions among the possible functions with N variables and K min-terms. The results of the proposed reduction method were compared to other new methods. The results showed that the proposed initialization method can accelerate the convergence of the method. Moreover, the proposed method excels other methods in terms of the needed gate count for the attained reduced circuits (around 20% improvement). The method can be readily used in other reduction problems such as minority based functions. Also, the proposed approach for finding the constitutive functions can make the simplification problem easier.

REFERENCES

1. Wolf, W., 2002. Modern VLSI Design: System-on-Chip Design, ser. Modern Semiconduct. Des. Upper Saddle River, NJ: Prentice-Hall.
2. Brayton, R.K., C. McMullen, G.D. Hachtel and A. Sangiovanni-Vincentelli, 1984. Logic Minimization Algorithms for VLSI Synthesis. Norwell, MA: Kluwer.
3. Zhang, R., K. Walus, Wei Wang and G. Jullien, 2004. A method of majority logic reduction for quantum-dot cellular automata, IEEE Transaction on Nanotechnology, 3(4): 443-450.
4. Akers, S.B., 1962. Synthesis of combinational logic using three-input majority gates, in Proc. 3rd Annu. Symp. Switching Circuit Theory and Logical Des., pp: 149-157.
5. Miller, H.S. and R.O. Winder, 1962. Majority logic synthesis by geometric methods, IRE Trans. Electron. Comput., EC-11(1): 89-90.
6. Walus, K., G. Schulhof, G.A. Jullien, R. Zhang and W. Wang, 2004. Circuit design based on majority gates for applications with quantum-dot cellular automata, in Proc. IEEE Asilomar Conf. Signals, Syst., Comput., pp: 1354-1357.
7. Zhang *et al.*, 2007. Majority and Minority Network Synthesis with Application to QCA, SET and TPL, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 26(7).
8. Zhi Huo, Qishan Zhang, Sansiri Haruehanroengra and Wei Wang, 2006. Logic Optimization for Majority Gate-Based Nanoelectronic Circuits, IEEE International Symposium on Circuits and Systems (ISCAS'06), pp: 1307-1310.
9. Mohammad Reza Bonyadi, Mostafa Rahimi Azghadi, Nzanin Malakooti Rad, Keian Navi and Ebrahim Afjei, 2007. Logic Optimization for Majority Gate-Based Nanoelectronic Circuits Based on Genetic Algorithm, ICEE ISSN: pp: 1-5.
10. Carlos, A. Coello Coello and Alan D. Christiansen, 1999. Arturo Hernandez Aguirre: "Use of Evolutionary Techniques to Automate the Design of Combinational Circuits, International J. Smart Engineering System Design.
11. Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Natural Selection. Cambridge, MA: MIT Press.
12. Input Boolean program spaces. Univ. College London, London, U.K. [Online]. Available: <http://www.cs.ucl.ac.uk/staff/W.Langdon/csrp-98-16/node4.html>.