A Novel Optimized QCA 4:1 Multiplexer Circuit Using Genetic Algorithm

Som Banerjee
Dept. of Computer Science and Engineering
Modern Institute of Engineering and Technology, Hoogly, India
sombanerjeecse@gmail.com

Chandrama Dey
Dept. of Electronics and Communication Engineering
Pailan College of Management and Technology, Kolkata, India
chandramadey1992@gmail.com

Abstract—With the rapid development of very large scale integration (VLSI) technology, it has became necessitate to design circuits with high operational speed with area efficiency and high device density. The quantum dot cellular automata (QCA) technology can be a very promising alternative to CMOS technology to maintain the progression of exponential Moore's law in the field of microelectronics. Majority gates and inverters are the fundamental blocks in quantum dot cellular automata circuits. In QCA, most important step is to reduce number of required majority gate and inverter for implementing a given Boolean function. This manuscript demonstrates a new method for reduction of number of majority gates and inverters in 4:1 multiplexer by implementing methodology on the basis of genetic algorithm. It has been proved that by implementing this proposed method less cell count, total area, cellular area and clock cycle are needed. Applying this proposed genetic algorithm and fitness function, any QCA based digital logic circuit can be optimized to obtain improved and efficient result.

Keywords—Quantum-Dot Cellular Automata, Majority gate, QCA clock, Genetic Algorithm, Fitness Function, Optimization, Multiplexer.

I. INTRODUCTION

Quantum-Dot Cellular Automata (QCA) has evolved as a significant alternative in nano scale computing also known as quantum computing [1]. It was first proposed by C. S. Lent in 1993 [2]. QCA takes advantage of operation speed at Tera Hz frequencies, low power consumption and high device density [2-8]. Traditionally circuit is optimized using karnaugh map by creating the expression into two forms, sum of products (SOP) and product of sums (POS) but this form of majority expression is difficult to make due to multilevel majority gates [9-10]. For overcoming this scenario, a new optimization technique is proposed which follows the principle of genetic algorithm (GA). Genetic algorithm works on the theory of natural evolution and its main concern is to find optimal solution for any circuit [11-13].

This manuscript shows a technique for optimization of 4:1 multiplexer circuit using genetic algorithm. Parameters such as gate count, clock cycle are considered for the systematic action of optimization. The objectives of this manuscript are given below.

- A new fitness function calculation for 4:1 multiplexer is done which is used for calculating the best optimal solution for multiplexer circuits.
- A new algorithm and methods for optimizing 4:1 multiplexer is done by following the previously proposed method of multiple output genetic algorithms (MOGA).
- Finally the presented technique is applied and simulation is executed and comparison is done between the presented technique and the previously done techniques.

II. BACKGROUND

A. Quantum-Dot Cellular Automata (QCA)

QCA cell is a square unit block consisting mainly of a charge container that contains four quantum dots. Each dots interact among each other through a metallic tunneling .When two extra free electrons are inserted in the cell, electrons start to repel because of their mutual electrostatic repulsion and occupies opposite corners of the cell i.e. the antipodal site [16-17]. This whole process is known as cell polarization and is expressed as P. To represent logic '1' P= +1 is used and to represent logic '0' P= -1 is used. Structure of QCA cell polarization is shown in Figure 1(a).

1) QCA wire: QCA is an array representation where binary information is passed from input to output. The polarization of a cell is influenced by that of its neighboring cells. Two types of wires are 90 and 45 degree as shown in Figure 1(b) and Figure 1(c) respectively.

2) QCA inverter: When QCA cells are diagonally placed it behaves as NOT gate i.e. inverter (inv). Because of the property of electrostatic repulsion between cells, this conversion of logic '0' to logic '1' and logic '1' to logic '0' takes place as described in Figure 1(d).

3) QCA majority gate: Majority Gate (maj) is the basic gate used in QCA for the formation of any logic circuits. The majority logic gate consists of an arrangement of five standard cells that is shown in the Figure 1(e). It generally contains 3 inputs and by taking into account the inputs present in majority, output is calculated. By fixing the value of polarization, AND, OR gates are formed. Equation to represent majority gate is shown in (1).

$$M(A, B, C) = AB + BC + CA \quad (1)$$

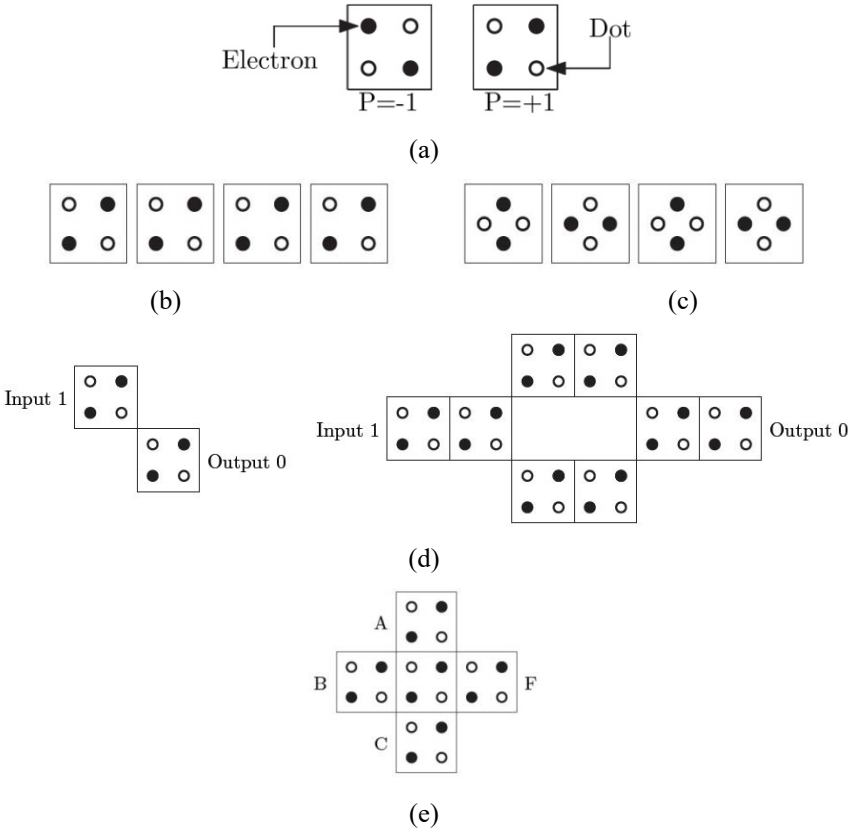


Figure 1.(a) QCA cell with polarization P (b) QCA 90° wire (c) QCA 45° wire (d) QCA inverter (e) QCA Majority Gate

B. Genetic algorithm

Genetic algorithm (GA) can be described as an evolutionary algorithm in which different types of computer program of dynamic variance of sizes and shapes are evolved to solve a particular problem. By using the Darwinian principle for natural selection, the populations of a computer program is genetically evolved. The genetic algorithm highlights the process of natural selection in which the fittest individual is selected for reproduction so that it can produce a new offspring for the next generation to continue. Natural selection begins by selecting the fittest individual from a population. They produce a new offspring which inherits the parent characteristics. If parent is having a better fitness, their offspring will be better than their parents having better chance of survival [11-13]. An iteration process keep on going and at the end the fittest individual is found. The six main phases of genetic algorithm are discussed below.

1) Initial Population: In genetic algorithm, initial population is the selection of random individual in the form of chromosomes. Tree is represented here as chromosome as shown in Figure 2. In this structure, majority gates and inverters are taken as internal node and the nodes where inputs and logical one '1' are present are considered as external node [12].

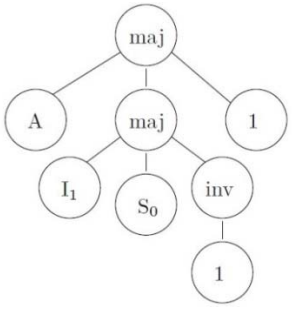


Figure 2. Chromosome [M(M(I_1,S_0,0),1,A)]

2) Fitness Function: In genetic algorithm, fitness function examines that how fit the individual is i.e. the potential of a particular individual to compete with another individual. It generates fitness score value to every individual. The probability of an individual to be selected for reproduction is completely based upon its fitness score value.

3) Selection: In genetic algorithm, selection is the procedure to select a particular chromosome before applying genetic algorithm over it. Selection is one of the most important factors because it determines the probability of getting the expected output.

4) Crossover: In genetic algorithm, crossover is the procedure of exchanging a sub-tree of a chromosome with a sub-tree of another chromosome. Crossover application in genetic algorithm makes the chromosomes a situation to go for a combination logic thereafter reducing the complexity of the circuit [13]. Crossover takes place with the probabilities P_c.

5) Combine: In genetic algorithm, after crossover operation takes place the combine procedure starts. In this procedure, two different chromosomes with similar sub branches are taken common and a new tree is formed consisting of both the chromosomes [13]. Combination techniques reduce the total number of gates required to make the expression thereby optimizing the total circuit.

6) Mutation: In genetic algorithm, mutation is the procedure in which the worst fitness factor valued chromosome within the population is replaced with a new randomly generated chromosome [13]. Mutation is mainly done to restore the genetic factor or diversity that might have been lost from the iterated application of selection and crossover. Mutation takes place with the probabilities P_m.

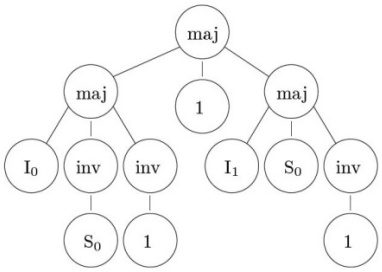
III. RELATED WORKS

Different works have been done for optimizing circuits based on AND/OR logic but reduction of majority gate was not their main concern. Zhang [14] proposed an automatic synthesis of optimal QCA circuit using Boolean algebra and that proposed technique reduced number of majority gates of three variable Boolean functions in QCA. In 2007, Bonyadi [11] made attempt to optimize majority gate based design using genetic algorithm considering single output circuit. In this method a chromosome is considered and presented in the form of tree consisting of internal nodes i.e. majority gates, inverter gates and leaf nodes. Houshmand [12] took forward this work and presented new technique to reduce number of gate count in the function having multiple output. Razieh [13] showed the minimization of the gates for particular Boolean truth tables of an arbitrary number of outputs by using genetic algorithm showing reduced delay of the considered implemented circuit.

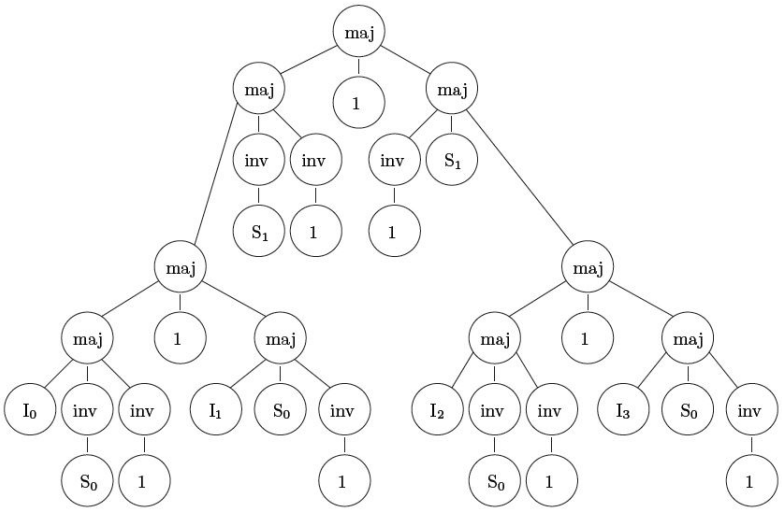
IV. PROPOSED WORK

A. Multiplexer circuit representation in genetic algorithm

In QCA based genetic algorithm approach, a logical circuit is represented by using a tree which is called chromosomes. Tree representation of a 2:1 multiplexer circuit and 4:1 multiplexer circuit is shown in Figure 3(a) and Figure 3(b) respectively. For representing a 2:1 multiplexer tree, 6 internal nodes are required out of which 3 are majority gates and 3 are inverters and for representing a 4:1 multiplexer tree, 18 internal nodes are required out of which 9 are majority gates and 9 are inverters.



(a)



(b)

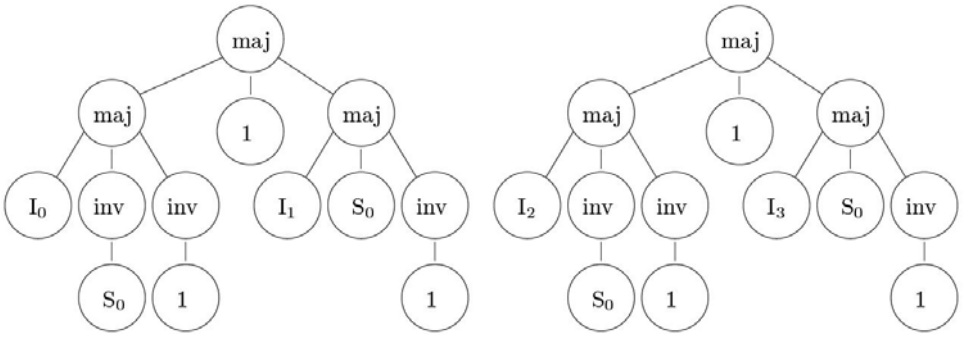
Figure 3. Tree Representation of (a) 2:1 multiplexer (b) 4:1 multiplexer

B. Optimization of 4:1 multiplexer using 2:1 multiplexer by applying genetic algorithm

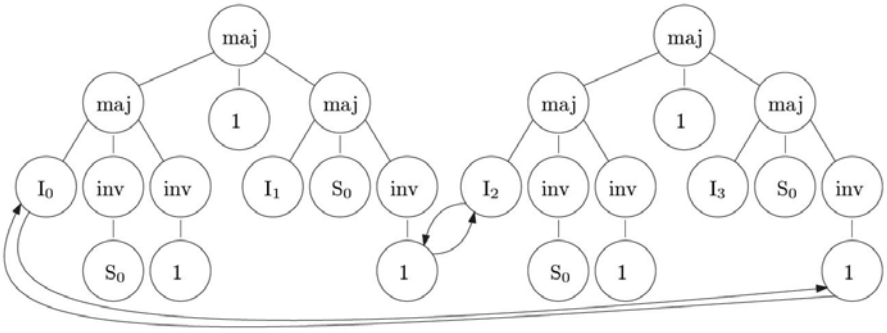
In genetic algorithm, logical circuit tree optimization is done by applying various steps of genetic algorithm like initial population, selection, mutation, cross over and combine. Optimization of 4:1 multiplexer is done using the logic of multiple output circuit. Here, two 2:1 multiplexer chromosomes of out-1 and out-2 shown in (2) and (3) having least fitness function value are taken and representation of this chromosomes are given by tree structure shown in Figure 4(a). Then the crossover stage starts to operate which actually selects the nodes which is likely to get crossed over between two trees based upon the crossover probability P_c. The nodes which will be crossed over are shown in Figure 4(b). Figure 4(c) represents the tree after crossover stage is complete. Then the final combination operation begins. The common parts which actually represents the identical properties between two trees are selected and combination in done to optimize the logical circuit as shown in Figure 4(d). At the end, another 2:1 multiplexer tree is put above the combined trees and another intra combination is performed between two inverter gates. Then the final optimized 4:1 multiplexer tree is created using three 2:1 multiplexer trees by applying genetic algorithm as shown in Figure 3(e).

$$\text{out} - 1 \ = \text{M}(\text{M}(\text{I2}, \text{S0'}, 0), \text{M}(\text{I3}, \text{S0}, 0), 1) \tag{2}$$

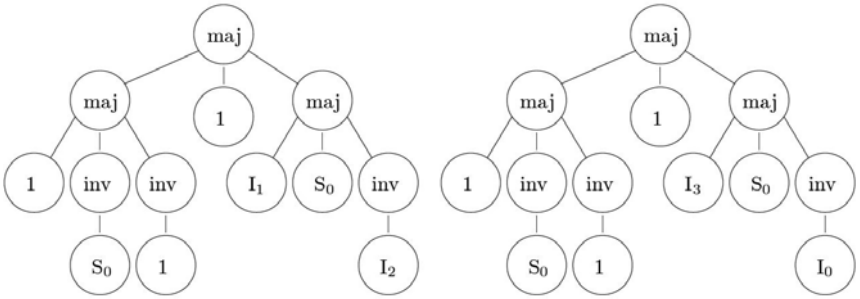
$$\text{out} - 2 \ = \text{M}(\text{M}(\text{I0}, \text{S0'}, 0), \text{M}(\text{I1}, \text{S0}, 0), 1) \tag{3}$$



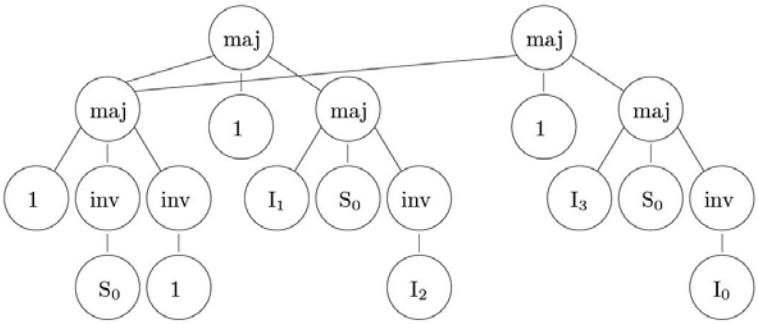
(a)



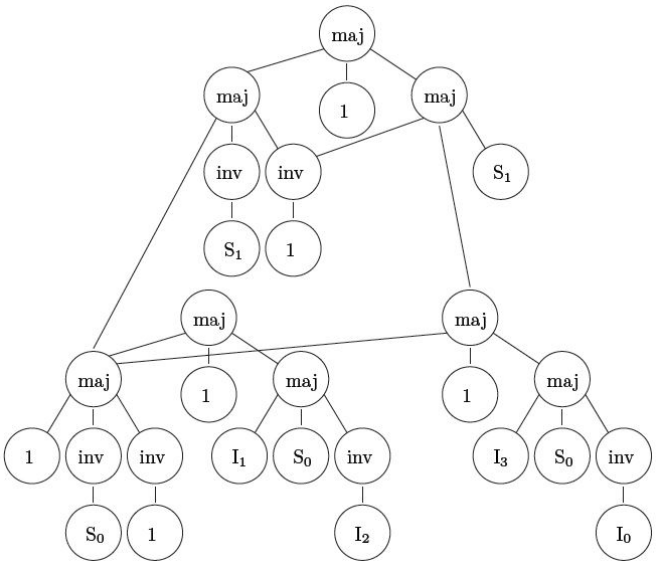
(b)



(c)



(d)



(e)

Figure 4. Tree representation (a) before crossover of two 2:1 multiplexer (b) at the time of crossover (c) after crossover (d) after combination (e) GA optimized 4:1 multiplexer

C. Genetic Algorithm of Optimized 4:1 Multiplexer

Genetic algorithm optimizes the total number of gates in a QCA logic circuit with a single output. Here, firstly, optimization of the lower level two 2:1 multiplexer is done by applying the genetic algorithm and then implementation of the algorithm in the upper level 2:1 multiplexer is done by applying the same algorithm. Then finally the optimized circuit of the 4:1 multiplexer is produced. This algorithm is an inductive approach i.e. the present output x is obtained noticing the output of the previous state output x-1. The algorithm is run until getting an expression synthesizing all of them. Crossover and mutation is done as explained earlier to get the best expression following the rule of fitness factor depicted in subsection IV-D.

The algorithm is executed for a large number of generations to get some of the chromosome with fitness factor having minimum value. These chromosomes are known as correct chromosome and is stored an array. The first loop *o,* denotes the total Number of Outputs. The second loop *i*, denotes the total Number of Generations of the genetic algorithm. Initial population is done here. The third loop *j*, denotes the Number of Chromosomes in Current Generation. The fourth loop, *k*, denotes the total Number of Cuts. The two randomly generated numbers M and N works as cutoff point for out-1[*j*] and out-2[*o*-1,*i*]. In this method, the out-1[*j*] obtains a common subtree with out-2[o-1,*i*] without changing its logic. In combined method, the common nodes of the subtree of the two chromosomes out-1[*j*] and out-2[*o*-1,*i*] are combined to get a common node part hence reducing or optimizing the complexity of the circuit. After combining, the fitness function value is calculated using (5) or (6) depicted in subsection IV-D and combined chromosome gate count is made using (4). Gates which become common are subtracted from the total number of gates of the two chromosomes. Then selection process is done. Crossover and mutation are done with their respective probabilities P_c and P_m in order to produce the next generations. The whole operation is run for number of generations to get the best chromosome in correct chromosomes. The pseudo code to demonstrate genetic algorithm is given in details in Pseudo code: 1 below.

$$\text{Gates (Combined Chromosome[j])} = \text{(Gates(out} - 2[\text{o} - 1, \text{i}]) + \text{Gates (out} - 1[\text{j}]))\text{– Number of Common Gates} \quad (4)$$

1. Start
2. Save all the chromosomes having minimum fitness value in correct chromosome.
   for ***o*←2** to *Number of Outputs* **do**
   for ***i*←1** to *Number of Generation* **do**
   for ***j*←1** to *Number of Chromosomes in Current Generation* **do**
   for ***k*←1** to *Number of Cuts* **do**
   Create a random generated number named *M* between 0 and Gates (out-1[*j*]).
   Create a random generated number named *N* between 0 and Gates (out-2[*o*-1,*i*]).
   Replace the M^{th} node and its subtree in out-1[*j*] with the N^{th} node and its subtree in the out-2[*o*-1,*i*].
   Combined Chromosome (j) = Combine (out-2[*o*-1,*i*], out-1[*j*]).
   end for
   Calculate fitness factor on the basis of number of nodes and levels in each chromosomes of
   Combined Chromosome.
   Gates (Combined Chromosome[j]) = Gates (out-2[*o*-1,*i*]) + Gates (out-1[*j*]) – Number of Common Gates.
   Make selection, crossover with probability value P_c, and mutation with probability value P_m for out-1.
   end for
   Save the chromosomes in *Combined Chromosome* having minimum fitness value in correct chromosome.
   end for
   Return the best obtained chromosome in correct chromosomes[*o*].
   end for
3. **RETURN** : *Optimized Chromosome*
4. Termination:
5. End

Pseudo code of genetic algorithm

Pseudo code 1: Genetic algorithm

D. Fitness Function Calculation

The fitness function has two goals. The first goal is synthesizing a QCA logic circuit corresponding to the output function. To achieve that, fitness function assigns better fitness value to a chromosome, which is closer to the expected result. A fitness value of 0.50 (fitness = "0.50") indicates the chromosome output vector equals the expected result. A fitness value of 0.75 (fitness = "0.75") indicates the chromosome output vector differs from the expected result. The second goal of fitness function is to synthesis the optimal circuit. The fitness function tries to optimize the number of the gates and levels of the circuit. A dynamic fitness table is maintained in the process of fitness calculation. All the fitness factor value of the chromosomes are stored in the fitness table and if the same chromosome is used again, its fitness value is retrieved from the fitness table reducing complexity and computation cost. The fitness of smaller value is considered to be a better solution than the fitness with larger value. The chromosomes are generated randomly and checked if it behaves as the expected results. The majority (maj), inverter (inv), common majority (com maj), common inverter (com inv), levels are required to find the fitness factor of a chromosome. The number of gate is counted as the sum of the majority gate, common majority gate with the one-third of the sum of inverter gate and common inverter gate. The equation to find the fitness value of a chromosome is given in (5), (6). If the chromosome output vector equals the expected result, then (5) is used to calculate the fitness factor otherwise (6) is used to calculate the fitness factor. The pseudo code of fitness function is given in Pseudo code: 2 below.

$$\text{Fitness value} = 0.50 - \left(\left(\frac{1}{\text{majority}+\text{common majority}+\left(\frac{\text{inverter}+\text{common inverter}}{3}\right)}\right)+\left(\frac{1}{\text{level}}\right)\right) \quad (5)$$

$$\text{Fitness value} = 0.75 - \left(\left(\frac{1}{\text{majority}+\text{common majority}+\left(\frac{\text{inverter}+\text{common inverter}}{3}\right)}\right)+\left(\frac{1}{\text{level}}\right)\right) \quad (6)$$

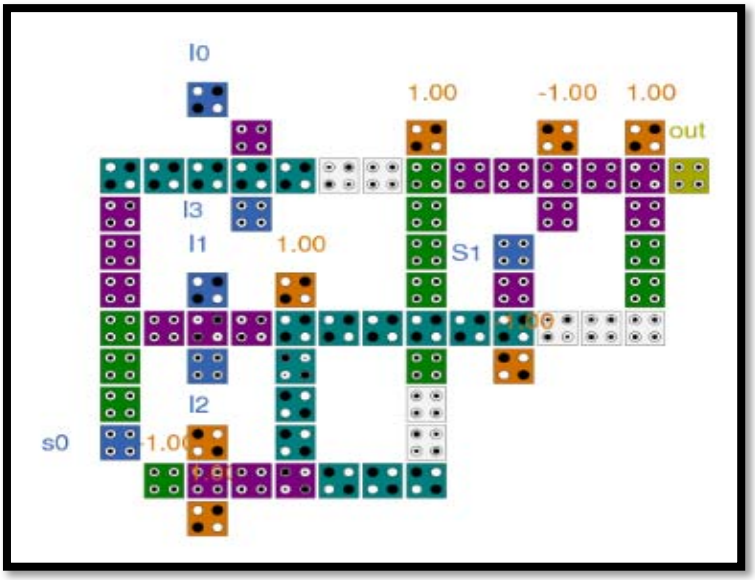
1. Start
2. **INPUT**: Initialization of chromosomes
   if chromosome *fitness_value* is present in *fitness_table* **then**
   fitness_value= *fitness_table*(chromosome);
   else
   majority ← total number of majority gates in the chromosomes
   inverter ← total number of inverter gates in the chromosomes
   common_majority ← total number of common majority gates in the chromosomes
   common_inverter ← total number of common inverter gates in the chromosomes
   level ← total number of layers in the chromosomes
3. Generate output vector of the chromosome
   if (output vector == expected result) **then**
   fitness_value = 0.50 - ((1/ (*majority* + *common_majority* + ((*inverter* + *common_inverter)* /3))) + (1/*level*))
   fitness_table(chromosome) ← *fitness_value*
   else
   fitness_value = 0.75 - ((1/ (*majority* + *common_majority* + ((*inverter* + *common_inverter)* /3))) + (1/*level*))
   *fitness_table(*chromosome) ← *fitness_value*
   end if
4. Optimize the circuit by crossover
5. Combine
6. Calculate *fitness_value* (Repeat Step 2 -5)
7. **Iteration:**
   end if
   RETURN: *fitness_value*
8. Termination:
9. End

Pseudo code 2: Fitness function

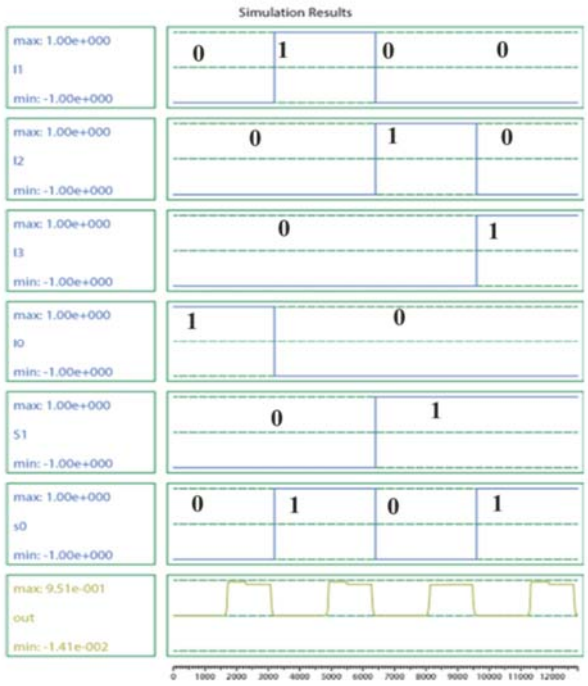
V. EXPERIMENTAL RESULT AND ANALYSIS

A. QCA implementations

The genetic algorithm based optimized 4:1 multiplexer tree by using three 2:1 multiplexer trees evaluated in subsection IV-B and represented by Figure 4(e) is simulated using QCA designer version 2.0.3. In Figure 5(a), layout of optimized 4:1 QCA based multiplexer circuit is created by implementing genetic algorithm. Simulation result of signal diagram of optimized QCA 4:1 multiplexer circuit is shown in Figure 5(b).



(a)



(b)

Figure 5. 4:1 Multiplexer (GA optimized) (a) QCA layout (b) Signal diagram

B. Evaluation and comparative analysis of fitness factor value

The fitness factor evaluation and comparison between the proposed algorithm for fitness function by using (5), (6) and the existing algorithm in [15] for fitness function is shown in TABLE I. After being implemented by using Matlab, it is graphically plotted and compared as shown in Figure 6. Fitness function of smaller value is considered to be a better solution than the fitness function with larger value. So from the graph it can be seen that the proposed algorithm gives better solution than [15] in terms of fitness factor. 15 chromosomes are randomly taken among

which two chromosome of smallest fitness values are considered for performing further operations (crossover, recombination, mutation) for forming 4:1 multiplexer and fitness value at each stage is calculated and stored in table. If the output vector and the expected results become equal, then required 4:1 multiplexer logic circuit is generated but the objective is to optimize the circuit so the 4:1 multiplexer having least fitness value is considered as the required optimized circuit.

TABLE I. ANALYSIS OF FITNESS FUNCTION COMPARISON (*MAJORITY GATE, #COMMON MAJORITY GATE, $INVERTER, %COMMON INVERTER, &LEVEL)

Chromosome	*	#	$	%	&	Fitness Factor		Output Vector = Expected Result
						Proposed	In [15]	
M(M(M(I_0,I_1,I_3),I_3,1)',M(M(1,I_2,I_1'),I_1',I_2),I_1)	5	0	3	0	5	0.38	0.67	NO
M(M(I_0,I_1,1)',M(I_1,I_2,1),I_0)'	3	0	2	0	5	0.28	0.60	NO
M(M(I_0,I_1,1)',0,M(I_1,I_0',I_2))'	3	0	4	0	5	0.31	0.65	NO
M(M(M(I_1,I_0',I_2),I_1,M(I_0,I_1,1)'),M(I_0,I_1,1)',0)'	4	0	3	1	6	0.39	0.70	NO
M(M(M(I_1,I_0',I_2),I_2',1)',I_2',M(I_1,I_0',I_2))	4	0	5	0	6	0.40	0.72	NO
M(M(M(I_1,I_0',I_3),I_0,I_3'),I_3',1)'	3	0	2	1	5	0.33	0.63	NO
M(M(M(M(S_0,I_2,0),S_0,S_0'),S_0,I_1'),S_0,0)	4	0	4	0	6	0.39	0.70	NO
M(M(M(I_1,I_2,1)',I_1,I_3'),I_1,M(I_2',I_3,0))	4	0	4	0	5	0.36	0.67	NO
M(M(M(I_0,I_1',0),I_0,M(I_1,I_2,0),1,M(I_0,0,M(I_1',I_0,M(I_1,I_2,0)))	5	1	5	0	6	0.45	0.74	NO
M(M(I_1,I_0',0)',M(I_0,I_1,1)',I_1)'	3	0	5	0	6	0.36	0.70	NO
M(M(M(I_0',I_1,I_2),I_2',1)',I_0,M(I_1,I_0',I_2))	4	0	4	0	6	0.39	0.70	NO
M(M(M(M(I_1,I_0',I_2),I_2',1)',I_2,I_3),I_0,I_3')	4	0	4	0	7	0.41	0.72	NO
M(M(M(I_1,I_0',I_2),I_0,I_2'),I_2',M(I_1,I_0',I_2)))'	4	0	4	0	6	0.39	0.70	NO
M(M(I_1,I_0',I_2'),',M(I_0,I_1',0),M(I_2,I_1,0))	4	0	6	0	5	0.38	0.70	NO
M(M(I_0',S_0',0),M(I_1,S_0,0),0)	3	0	5	0	4	0.28	0.62	NO
M(M(I_2,S_0',0),M(I_3,S_0,0),1)	3	0	3	0	4	0.25	0.58	NO
M(M(I_0,S_0',0),M(I_1,S_0,0),1)	3	0	3	0	4	0.25	0.58	NO
M(M(M(1,S_0',0),M(I_1,S_0,I_2'),1),M(M(1,S_0',0),M(I_3,S_0,I_0'),1),1)	5	1	4	0	5	0.41	0.70	NO
M(M((M(M(I_0,S_0',0),M(I_0,S_0,0),1)),S_1',0),M((M(M(I_2,S_0',0),M(I_3,S_0,0),1)),S_1,0),1)	9	0	9	0	6	0.25	0.77	YES
M(M((M(M(1,S_0',0),M(I_1,S_0,I_2'),1)),S_1'0),M(0,(M(M(1,S_0',0),M(I_3,S_0,I_0'),1)),S_1),1)	7	1	5	1	6	0.23	0.76	YES

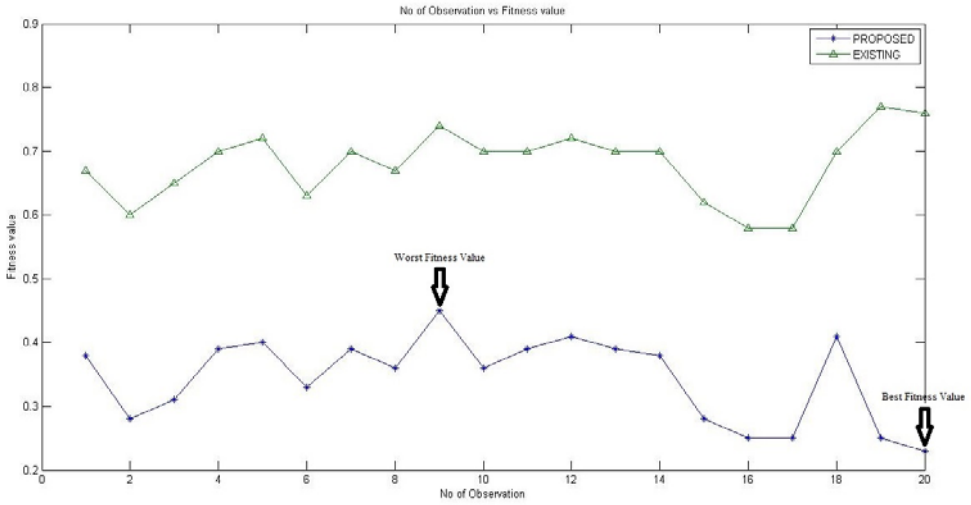


Figure 6. Graphical comparison among fitness function

C. Comparative analysis of gate requirement

From the QCA simulation, a comparative analysis is done upon requirement of majority gate, inverter and total gate. Comparative TABLE II shows that the normal QCA based 4:1 multiplexer design consumes more gate requirement count than the proposed genetically optimized QCA based 4:1 multiplexer design.

TABLE II. COMPARISON TABLE OF 4:1 MULTIPLEXER USING GENETIC ALGORITHM AND NORMAL QCA

4:1 Multiplexer	Required Majority gates	Required inverter	Common gates	Total number of gates
Normal QCA	9	9	0	9+9=18
Proposed (using GA)	8	6	1	8+7-1=14

D. Comparative analysis of cells, area, clock zone

From the QCA simulation result, a comparative analysis is done upon different parameters like cell requirement, area, number of clock zones. Comparative TABLE III shows that the other recent QCA based 4:1 multiplexer design consumes more cell, area and clock zones than the proposed method.

TABLE III. COMPARISON TABLE OF DIFFERENT 4:1 MULTIPLEXERS

Design	Complexity (Cell Count)	Area (μm^2)	Delay (Clock Zones)
[4]	271	0.39	19
[3]	215	0.25	6
[5]	155	0.24	5
[18]	251	0.20	5
[19]	124	0.25	8
[20]	107	0.15	4
Proposed	67	0.08	2.5

VI. CONCLUSION

This manuscript demonstrates a novel genetically optimized QCA based 4:1 multiplexer circuit. New calculation of fitness function and genetic algorithm is proposed and is applied in this 4:1 multiplexer. Optimization of 4:1 multiplexer is done by reducing required number of majority gate and inverter and this optimization is done by implementing new proposed fitness function and genetic algorithm. The proposed multiplexer architecture is simulated by using QCA designer version 2.0.3. From the simulation results, it is proved that the proposed genetic algorithm based QCA 4:1 multiplexer architecture provides an improved result in terms of complexity (cell count), area (μm^2) and delay (clock zone) when compared with other QCA 4:1 multiplexer architecture in [3, 4, 5, 18-20].

REFERENCES

[1] M. Wilson, K. Kannangara, G. Smith, M. Simmons, B. Raguse, "Nanotechnology: basic science and emerging technologies", CRC press, 2002.

[2] C. S. Lent, P. D. Tougah, W. Porod, G. H. Bernstein, "Quantum cellular automata", Nanotechnology, vol.4, pp.49–57, 1993.

[3] V. A. Mardiris, I. G. Karafyllidis, "Design and simulation of modular 2^n to 1 quantum-dot cellular automata (QCA) multiplexers", Int. J. Circ. Theor. Appl, vol.38, pp.771–785, 2010.

[4] R. S. Nadooshan, M Kianpour, "A novel QCA implementation of MUX-based universal shift register", J. Comput Electron, vol.13, pp.198–210, 2013.

[5] B. Sen, M. Goswami, S. Mazumdar, B. K. Sikdar, "Towards modular design of reliable quantum-dot cellular automata logic circuit using multiplexers", In Computers and Electrical Engineering, vol.45, pp.42–54, 2015.

[6] K. Kim, K. Wu, R. Karri, "The robust QCA adder designs using composable QCA building blocks", IEEE Trans. Comp. Aid. Des. Integ. Circ. Syst, vol.26, pp.176–183, 2007.

[7] S. Hashemi, M. R. Azghadi, A. Zakerolhosseini, "A novel QCA multiplexer design", International Symposium on Telecommunications, pp.692–696, 2008.

[8] A. Roohi, H. Khademolhosseini, S. Sayedsalehi, K. Navi, "A novel architecture for quantum-dot cellular automata multiplexer" IJCSI, vol.8, issue.6, pp.55–60, 2011.

[9] C. S. Lent, P. D. Tougaw, "Lines of interacting quantum-dot cells-a binary wire", Journal of applied physics, vol.74, 1993.

[10] I. Amlani, A. O. Orlov, G. Toth, G. H. Bernstein, C. S. Lent, G. L. Snider, "Digital logic gate using quantum-dot cellular automata", Science, vol.284, pp.289-291, 1999.

[11] M. R. Bonyadi, S. M. R. Azghadi, N. M. Rad, K. Navi and E. Afjei, "Logic optimization for majority gate-based nanoelectronic circuits based on genetic algorithm", International Conference on Electrical Engineering, ICEE, 2007.

[12] M. Houshmand, S. H. Khayat, R. Rezaei, "Genetic algorithm based logic optimization for multi-output majority gate-based nano-electronic circuits", IEEE, pp.84-88, 2009.

[13] R. Rezaee, M. Houshmand, M. Houshmand, "Multi-objective optimization of QCA circuits with multiple outputs using genetic programming", Genetic Programming and Evolvable Machines-Springer, vol.14, pp.95-118, 2013.

[14] R. Zhang, K. Walus, W. Wang, G. A. Jullien, "A method of majority logic reduction for quantum cellular automata", IEEE Transactions on Nanotechnology, vol.3, no.4, pp.443–450, 2004.

[15] M. H. Mahalat, M. Goswami, A. Mondal, B. Sen, "Synthesis and Optimization of multi-objective multi-output QCA circuit using genetic algorithm", In arXiv, pp.1-10, 2017.
[16] G. L. Snider, A. O. Orlov, I. Amlani, G. H. Bernstein, C. S. Lent, J. L. Merz, W. Porod, "Quantum dot cellular automata: line and majority logic gate", Japn. J. Appl. Physics, vol. 38, pp.7227-7229, 1999.
[17] P.D. Tougaw, C. S. Lent, "Logical devices implemented using quantum cellular automata", J. Appl. Phys, American Institute of Physics, vol.75, issue.3, pp.1818-1825, 1994.
[18] G. Cocorullo, P. Corsonello, F. Frustaci, S. Perri, "Design of efficient QCA multiplexers", Int. J. Circ. Theor. Appl, vol.44, pp.602–615, 2016.
[19] M. Askari, M. Taghizadeh, K. Fardad, "Digital Design Using Quantum-Dot Cellular automata (A Nanotechnology Method)", Proceedings of the International Conference on Computer and Communication Engineering, pp.952-955, 2008.
[20] H. Rashidi, A. Rezai, S Soltany, "High-performance multiplexer architecture for quantum-dot cellular automata", J. Comp. Electr, vol.15, pp.968–981, 2016.