Reversible Logic Circuit Synthesis and Optimization using Adaptive Genetic Algorithm

Hanumanthu Bhookya
Dept. of Electronics & Communication Engg,
Christu Jyothi Institute of Technology & Science,
Jangaon, Telangana, India.
bhookya.hanmanthu471@gmail.com

B Santosh Kumar
Dept. of Electronics & Communication Engg,
Christu Jyothi Institute of Technology & Science,
Jangaon, Telangana, India.
santoshcjits@gmail.com

Abstract:

In the recent years reversible logic circuits have gained a remarkable interest in the light of advancements made in quantum computation. The promise of theoretically zero power consumption is a major driving force for researchers to develop circuits using this logic. Synthesis of reversible logic and generating reversible logic circuit automatically with lower cost always has been a challenging task as reducing the search space is one of the major issues in the synthesis process using permutation. In this work, we proposed an Adaptive Genetic Algorithm (AGA) for synthesizing reversible logic circuits. NCT (NOT, CNOT, Toffoli) gate based library has been considered for synthesis of reversible logic circuits (RLC). The proposed algorithm produces a cascade of Toffoli gates for a given reversible specification of a circuit. Comparison of experimental results for several benchmark circuits shows that proposed evolutionary algorithm enables optimal or near-optimal solutions with lesser Gate Counts.

Keywords: Reversible Logic; Genetic Algorithm;Optimization.

Introduction:

With higher levels of integration and increasing scaling; Moore's law seems to be valid yet, but in traditional (irreversible) technologies heat produced by each IC doubles. Work by Landauer showed that, regardless of the underlying technology, conventional logic circuits dissipate heat in an order of kTLn 2 joules for every bit of information that is lost, where k is the Boltzmann constant and T is the operating temperature. Lossless computing offers an alternative, where a logical operation does not yield information loss called reversible operation. In a reversible logic, output patterns can be used to recover the input pattern due to one to one mapping, so no information is lost during computation. Later Bennett (1973) demonstrated logically zero power dissipation is achievable, if the computation is carried in a reversible way, i.e. entropy of the system does not decrease. Therefore, circuits are required which theoretically dissipates zero power, and in this regard reversible logic is the viable choice.In this paper, we propose a new method for synthesizing reversible logic circuits based on an Adaptive Genetic algorithm which decreases the search space and finds near optimal solution for the desired specification. The algorithm has been made adaptive by changing the mutation rate and the selective rate of genetic algorithm over an exponential function. The rest of the paper is discussed as follows: Section 2 discusses related work. Section 3 describes briefly the various concepts and preliminaries of reversible gate and reversible circuits. Section 4 presents the implementation of the proposed algorithm step by step. Section 5 discusses the experiment results performed on various benchmark circuits and section 6 finally concludes the

paper.

Related Work:

Reversible logic has applications in various domains such as Low power VLSI, Fault tolerant designs, quantum computing, nanotechnology, DNA computing, optical computing, cryptography and informatics. Study of reversible logic gates and their physical realization has gained much interest, but theories for its synthesis are not well developed. Toffoli[4], Feynman[5], Fredkin[6] and Peres[7] are the most commonly used fundamental reversible gates. Various methods used by the researchers for synthesis of reversible logic circuits that includes: Search method using graph theory (DFS and BFS)[8], Exclusive-Or Sum of Product (ESOP) method[9], Cycle based[10], Decision Diagram based[11] etc. The synthesis of Reversible Logic Circuit (RLC) using DFS/BFS and EOSP are not optimum in terms of quantum cost (QC) and/or gate count (GC) as they stick into local minima. Shende et al.[12] proposed an optimal synthesis method for reversible circuit synthesis, which works only for small circuits. For larger circuits this algorithm fails to provide optimal solutions. Agrawal et al.[13] presented an algorithm that makes use of positive- polarity Reed–Muller (PPRM) expansion to synthesize a reversible function and generates a network of Toffoli gates.

A technique like evolutionary algorithm is an alternative way to address the optimization problem. Currently, few works have been done to synthesize reversible logic circuits using evolutionary algorithms. Lukac et al.[14] proposed an automated synthesis of reversible circuit using Genetic Algorithm is proposed. Here main emphasis is given on problem encoding which results in providing better solutions. Datta et al.[15] introduced a PSO (Particle Swarm Optimization) based search techniques for synthesis of a reversible logic circuit. The algorithm works to obtain a near optimal solution without exploring the entire search space. Li et al.[16] proposed a best-path search algorithm based on ACO (Ant Colony Optimization) for reversible logic synthesis. This method is suitable for handling large reversible functions and efficiently generates either optimal or near-optimal circuit with less gate count. Much work has been done to synthesize reversible logic circuits and the preliminary results are encouraging, but there is a huge scope of improvement as the number of gates to realize a reversible circuit are still large. Hence, to improve effectiveness and efficiency, this paper used an Adaptive Genetic Algorithm (GA) based approach to synthesize a reversible logic circuit, with the objective of providing less Gate Count within a reasonable time.

Preliminaries:

Reversible Function: An *n*-input, *n*-output function $f(x_1, x_2, \ldots, x_n)$ of *n* variable is a reversible function if it maps distinct input to distinct output, i.e. a *n* * *n* bijective function mapping between input and output. For *n* inputs $2^n!$ (n × n) reversible gates are possible. In general a truth table or a permutation can be used to represent a reversible function

Generalized Toffoli gate: The Toffoli gate is widely used universal reversible gate. For the set of input variables $X = \{x_1, x_2, \ldots\ldots, x_{n+1}\}$ the generalized Toffoli gate can be represented as TOF(C;T) or C^n NOT($x_1, x_2, \ldots\ldots, x_{n+1}$). A past control Toffoli gate inverts the value of target line if all the control lines feed to 1, and passes the values on control lines unchanged. A negative control Toffoli gate inverts the value of the target line if all the control lines feed to 0, and passes the values on control lines unchanged. For n □ 0 , a

generalized Toffoli gate is known as NOT gate which has no control lines. For $n \geq 1$, the CNOT gate also known as Feynman gate with single control line, is a 2- input reversible gate which works on the following principle; first input (control input) is passed to the output directly while the second input (target input) is inverted only if the control input is '1'. For $n = 2$, the C^2NOT gate commonly known as Toffoli gate with two control lines. Here the first two inputs are passed to the outputs directly while the third input is inverted to the output only if the first two inputs are set, i.e. '1'. These three gates compose the universal NCT library. The three gates are illustrated in Fig.1

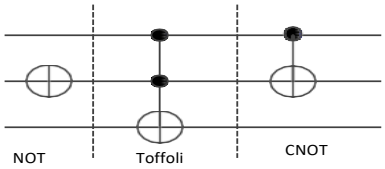


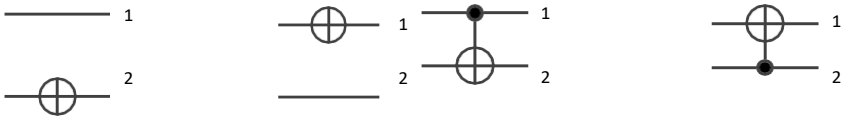
Fig. 1. An example of RLC

Gate Count (GC): The number of reversible or quantum gates needed to realize a circuit is the GC of that circuit. GC is used to compare the designs where all the gates are similar in size and type. The reversible logic circuit shown in Fig.1 has a Gate Count of 3

Genetic Algorithm (GA): GA is a search heuristic and an optimization algorithm that follows the mechanism of natural evolution. GAs belongs to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution. Inheritance, mutation, selection, and crossover are the basic building block of GA. In each generation epoch, every individual in the population is evaluated based on fitness value; multiple individuals are stochastically selected from the current population, based on their fitness, and modified to form a new population. The new population is then used in the next iteration of the algorithm.

The Proposed Algorithm: In this section, an algorithm to optimize a given reversible circuit using the GA is proposed. The Genetic Algorithm (GA) formulation of any problem must consider ways to represent the solution, compute the fitness of solutions, and carry out crossover and mutation operations.

Solution Representation: Before using the GA to optimize a reversible logic circuit, it has to be coded as a string, which is called a chromosome. To code a circuit, we have to select a representation scheme for it. In this research, cascade style for circuit representation is used. The approach used in the present work can be applied to solve the synthesis problem with respect to any reversible gate library. However, we have used generalized Toffoli gates as the target gate library in the implementation (NOT and CNOT gates are special cases of generalized Toffoli gates). For the sake of conciseness in representation, an n-input generalized Toffoli gate can be represented by an integer. First position in the string represents the location of target line, and the subsequent positions are to represent control inputs. For instance in Fig. 2(a), first integer in the string is 2 so line 2 represents a target line and second integer is also 2 represents no control line, which is nothing but a NOT gate. Similarly CNOT and Toffoli gates are represented in Fig.2 (c) and Fig.2 (e) respectively.



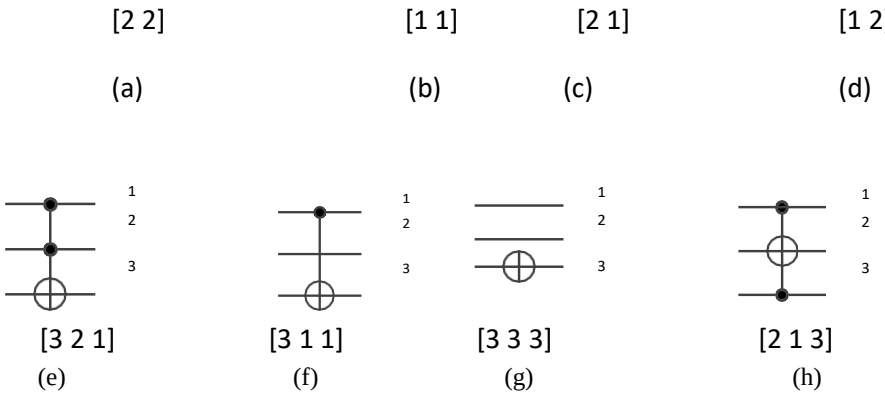


Fig. 2. Coding methods for NCT library gates.

Fitness function evaluation:

Fitness function evaluation in GA is the key factor for the convergence of the generation to the global solution and determines the effectiveness of our proposed algorithm. In AGA we divided the fitness function *f* into three parts, which are explained below:

- One of the major criteria of optimization is the maximization of the number of correct output pins. For this we define a fitness function,

$$f1 = 1 - \frac{hd}{hd_{\max}} \tag{1}$$

Where, *hd* is the Hamming distance and $hd_{max} = 2^{mgl} * mgl$

- RLC with lesser QC are more favourable,

$$f2 = 1 - \frac{Qc}{Qc_{\max}} \tag{2}$$

Where QCmax= $\left| \text{NoG}(2^{\text{mgl}} - 3) \right|$ (3)

mgl is the maximum gate length (Maximum value of the size of the gate).

- The RLC with lesser number of wired gates(*NWG*) out of Number of gates(*NoG*) is favourable, which gives us the third fitness function,

$$f3 = 1 - \frac{NWG}{NoG} \tag{4}$$

The overall objective function is the weighted sum of all these fitness functions given as,

$$f = \sum_{i=1}^{3} a_i(itr) * fi(itr) \tag{5}$$

where, $\alpha_i(itr)$ is the weight of each fitness function in the itr^{th} generatio

We have changed, α_i dynamically as,

$$\alpha_i(itr+1) = w\alpha_i(itr) * (1 - \overline{f_i(itr)}) \tag{6}$$

We have considered adaptive genetic algorithm (AGA), where the mutation and crossover rates are changed based upon the generation count and hamming distance respectively. With the increase in generation count (*itr*) and hamming distance (hd) the rate of crossover decreases. If the crossover rate is represented by r_c and the hamming distance of two individuals by *hd*, then r_c is given as:

$$r_c = \frac{r_{co} e^{\frac{-k_1 itr}{\max it}}}{1 + k_2 . hd / hd_{\max}} \tag{7}$$

Where, r_{cO} is the initial mutation rate; *itr* and *maxit* are the current and maximum generation value respectively.hd_{max} is the maximum hamming distance among chromosomes and k_1, k_2 are constants. Similarly the mutation rate is decreased exponentially with the generation (*itr*) value as:

$$m_t = m_{t0} e^{\frac{-\beta . itr}{\max it}} \tag{8}$$

Where, m_{t0} is the initial crossover rate and β=2.

Example 1: Consider a reversible specification f = [1032] of circuit size mgl = 2.

Let's consider *NoG*=6, α_1=0.4, α_2=0.3, α_3=0.3, *w*=0.9

For itr=0,input vector [0 1 2 3] use chromosome1= [2 2, 2 1, 1 1, 1 2] as shown in Fig.3, and generates output vector= [1 2 0 3].

The fitness function f_1=1- (hd/hd_{max})=1-(4/8)=0.5

f_2=1-(4/6)=0.33 f_3=1-(4/6)=0.33

Overall fitness function f= ($\alpha_1 * f_1$) + ($\alpha_2 * f_2$) + ($\alpha_3 * f_3$) =(0.4*0.5)+(0.3*0.33)+(0.3*0.33)=0.398

$\square_{i(itr+1)}$dynamically updated as $\square_i(1)=(0.9*\square_i(0))*(1-0.386)=0.613*(0.9*\square_i(0))$ For k_1=2; k_2=1, r_{cO}=0.9, *maxit*=400

Crossover rate is updated as $r_c = \frac{0.9 * e^{-\frac{2*1}{400}}}{1 + 1 * \frac{4}{8}}$

Mutation rate updated as $m_t = 0.2 * e^{-2*1/400}$

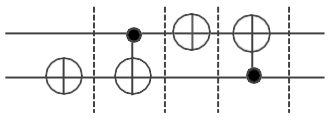


Fig. 3. Representation of chromosome1= [2 2, 2 1, 1 1, 1 2].

For a given reversible specification and size, the evolution process starts with a randomly generated individual called as a population. Each Individual/chromosome represented by a sequence of integers as

shown in Fig.3. For the each generation fitness value of the individual has been calculated as illustrated in example 1 and depending upon the fitness values two chromosomes get selected, and then evolved (mutated, crossed-over, or replicated) to the next generation of individuals in the search of optimal solution. Generally, the algorithm terminates on two conditions: 1) maximum number of generations has been reached; 2) Suitable fitness value has been achieved in the population. The basic structure of the has been reached; algorithm is described in Fig.4

```
Input: The desired permutation vector Output: A sequence
of Toffoli gates begin
    generate_ initial_population P(itr=0);
    for itr=0 to maxit begin calculate_fitness(itr);
        for k=1 to popsize
            begin
                Select two individual;
                Adjust crossover rate based on hamming distance and itr (iteration) value; Crossover on the
                population P(itr) & generate the new population P'(itr); Mutate the population P'(itr) and generate the
                new population P(itr+1);
            end
        itr=itr+1;
    Check for optimal solution in P(itr) & terminate;
    end
```

Fig.4. Basic structure of the algorithm.

Fig.4. Overall flow of the synthesis algorithm

Experimental Results:

The algorithm is implemented in MATLAB R2012a (7.14.0.739) on a core i5 based PC with 2.5GHz and 4GB RAM. The program receives desire permutation as an input and output a sequence of gates. The parameters used in this algorithm were tuned through a series of experimentation simulations. Values of various parameters used are

- Population Size (popsize): 20
- Initial Crossover rate (r_{c0}):0.9
- Initial mutation rate (m_{t0}):0.2
- Number of generations (**m**axit): 400

The input to the program is the array of desired output (des_out) and output is the array of gates needed for the synthesis. For example, when [1 0 3 2] is given as the des_out, the output gate array is:

[1 2 1 2 1 1 1 1 1 2 1 1 1 2 1]

Where each gate is a two input gate, and after removal of redundant gates the output gate array is:

[1 1 2 1 1 1 2 1]

The gate array is shown in Fig. 5.

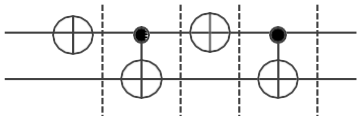


Fig.5. Reversible function [1 0 3 2].

Experimental results obtained are shown in Table 1. The metric chosen for comparison is the gate count (GC), which is equal to the number of gates in the synthesized netlist. For the benchmarks we have considered same sixteen circuits from recent works MOSAIC[18], PPRM[13], PSO[15], ACO[16]. The synthesis results obtained within a fraction of a second. Results suggest that our proposed algorithm shows optimal results for most of the benchmarks reported and near optimal for few

Table 1: Experimental Results

Function name	Gate count				
	MOSAIC[18]	PPRM[13]	PSO[15]	ACO[16]	Proposed AGA
[1,0,3,2,5,7,4,6]	4	4	5	3	4
[7,0,1,2,3,4,5,6]	3	3	3	3	3
[0,1,2,3,4,6,5,7]	3	3	3	3	3
[0,1,2,4,3,5,6,7]	7	5	5	4	4
[1,2,3,4,5,6,7,0]	3	3	3	3	4
[3,6,2,5,7,1,0,4]	8	7	8	6	6
[1,2,7,5,6,3,0,4]	8	7	6	6	6
[4,3,0,2,7,5,6,1]	6	6	6	5	5
[7,5,2,4,6,1,0,3]	6	7	-	5	6
[0,1,2,3,4,5,6,8,7,9,10,11,12,13,14,15]	9	7	-	7	7
[0,7,6,9,4,11,10,13,8,11,12,13,14,15,0]	4	4	-	4	4
[6,2,14,13,3,11,10,7,0,5,8,1,15,12,4,9]	19	14	-	11	12
[9,7,13,10,4,2,14,3,0,12,6,8,15,11,1,5]	23	14	-	11	10
[6,4,11,0,9,8,12,2,15,5,3,7,10,13,14,1]	21	17	-	13	11
[13,1,14,0,9,2,15,6,12,8,11,3,4,5,7,10]	29	14	-	10	10
[1,2,3,4,5,6,8,7,9,10,11,12,13,14,15,0]	4	4	-	3	3

Conclusion:

This work focuses on synthesis of reversible logic circuits. This paper proposed a novel method of reversible logic synthesis based on an Adaptive GA. For a given specification the proposed algorithm generates a sequence of generalized Toffoli gate. We also proposed a new encoding method for a generalized nxn Toffili gate. This approach is flexible as the weight of each fitness function can be changed dynamically; mutation and crossover rate becomes more adaptive depend upon the generation count and hamming distance. The proposed work has been compared with existing works for smaller input function. Result shows, proposed method gives better performance in fewer Gate Count. The reversible logic circuit application has a number of future scopes.

The scalability and searching can be improved by reducing search space, which is achieved by adding a number of features such as: Fuzzy-connection based crossover, Hybrid GA etc. for a quicker and better results.

References:

1. Moore GE. Cramming more components onto integrated circuits. *Electronics* 1965; 38(8): 114–117.
2. Landauer R. Irreversibility and heat generation in the computing process. *IBM Research and Development* 1961; 5:183-191.
3. Bennett H. Logical reversibility of computation. *IBM J. Research and Development* 1973; 17:525-532.
4. Toffoli T. Reversible Computing. *Technical Report MIT/LCS/TM-151*;19
5. Feynman R. Quantum mechanical computers Optic News. *Foundations of Physics* 1985; 11:11-20.
6. Fredkin E, Toffoli T. Conservative Logic. *International Journal of Theoretical Physics* 1980; 21: 219-53.