An Evolutionary Approach to Reversible Logic Synthesis using Output Permutation

Kamalika Datta*, Indranil Sengupta †, Hafizur Rahaman *, and Rolf Drechsler‡
*Department of Information Technology, Bengal Engineering & Science University, Shibpur, Howrah 711103, India
Email: kdatta.iitkgp@gmail.com, rahaman_h@it.becs.ac.in
†Department of Computer Science & Engineering, Indian Institute of Technology, Kharagpur 721302, India
Email: isg@iitkgp.ac.in
‡Institute of Computer Science, University of Bremen / DFKI Bremen, Germany
Email: drechsler@uni-bremen.de

***Abstract*—The area of reversible circuit synthesis has become very important in recent years with the growing emphasis on low-power design and quantum computation. Many synthesis approaches have been reported over the last two decades. For small functions exact solutions can be computed. Otherwise, heuristics have to be applied that are either based on transformations or a direct mapping from a given data structure. Recently, it was shown that significant reduction in the cost of the synthesized circuits can be obtained, if the ordering of the output lines is changed. The drawback of the approach was that it can only be applied to smaller sized circuits.**

In this paper, an evolutionary approach for obtaining a good ordering of the output variables is proposed, which can be used for larger sized circuits as well. The method does not require explicit synthesis of the reversible circuit netlist. Experimental results are shown with respect to a transformation based synthesis tool. Reductions of up to 98% can be observed with an average reduction of 64.4% for larger circuits.

***Index Terms*—Reversible logic, evolutionary algorithm, output permutation**

I. Introduction

With the prospect of having quantum computers in a not-so-distant future, and the growing need of applications that demand ultra-low power consumption, research in the area of synthesis and testing of reversible logic circuits has drawn the attention of various researchers. A reversible circuit consists of reversible gates. A reversible circuit must also have equal number of inputs and outputs, and it should not have any fan-outs. Traditional gates such as AND, OR, EXOR are not reversible, but the NOT gate is reversible. There exist various reversible gates in the literature, such as Controlled NOT (CNOT) [4] and Toffoli [19], which are used by various synthesis methods. Many approaches to the synthesis of reversible logic circuits have been reported. While some of them are based on solving a state-space search problem, others are based on heuristics or randomized search. Approaches based on exact or constructive approaches have been proposed, while some are based on innovative ways of representing the set of functions as *Binary Decision Diagrams* (BDD) or *Exclusive Sum-Of-Products* (ESOP). While the exact approaches give optimal results only for smaller sized circuits, others give semi-optimal solutions but possibly run faster and can handle significantly large sized circuits.

It has been shown in [21] that the cost of the synthesized netlist can be significantly reduced if the order of the output lines is changed, which is often acceptable in practice. In other words, a permutation is imposed on the output lines. An optimization scheme based on a formulation as a *Satisfiability* (SAT) problem was proposed in [21] that can search through all feasible output permutations, and arrive at the one that leads to the least cost gate netlist. A formulation for incompletely specified functions was also presented, where some of the permutations that correspond to garbage outputs were skipped in the search process. Heuristics were presented to speed up the search process. Results have only been shown for smaller number of inputs, since the complexity of the algorithm increases rapidly with the size of the instance. As such, no methods have been reported so far that can handle circuits with larger number of inputs, while using output permutation for optimization. An excellent set of resources relating to tools for synthesis and analysis of reversible logic circuits are available in [16]. The tools, though capable of handling larger circuits, are not equipped with optimization mechanism using output permutation.

In this paper, a fast *Evolutionary Algorithm* (EA) is presented to address this specific problem. Given a reversible specification, based on a properly selected cost function, the algorithm searches for an output permutation that leads to the smallest cost. To avoid long runtimes the EA does not synthesize the circuit after each evolutionary operation. Both the original and the *least cost* permutations are synthesized using a standard synthesis tool, and the improvement in cost are compared. Experimental results demonstrate the effectiveness of the scheme.

II. Related Works

The various approaches to synthesis of reversible logic circuits can be broadly classified into *exact*, *heuristic*, and

based on higher-level *function representation*.

Exact synthesis approaches generate a minimal reversible logic realization of a given specification [7] [9] [15]. The problem with these methods is that since they tend to search the entire solution space for the optimum solution, they have high time complexity and can only be used for small sized circuit specifications. These methods are often used as benchmarks to evaluate the quality of solutions obtained by other synthesis approaches.

The heuristic based approaches tend to use domain specific knowledge and simple rules, so as to restrict the search space. There are several methods that fall in this category, such as [1] [8] [10] [11] [22]. These methods take less computational time as compared to the exact methods, and can be used for larger circuits; however, there is no guarantee of an optimal solution.

To handle the synthesis of very large circuits, some methods have been proposed which rely on special compact functional realization of the reversible specification. There are methods based on *Binary Decision Diagrams* (BDD), which represent the multi-output function specification as a shared reduced BDD [20]. Using some simple linear time transformation methods, the nodes of the BDD are directly transformed into segments of the reversible logic netlist. There are methods based on *Exclusive Sum-Of-Products* (ESOP) [2] [3], which represent the functions as a linear sum of product terms (similar to the Reed-Muller representation). Again using simple translation rules, the gate netlist can be directly generated from the ESOP expressions. Although these methods can handle larger functions, the main drawback is that the quality of the solutions obtained is often quite significantly suboptimal. Furthermore, additional lines might be required. Some methods use an alternate data structure called Quantum Multiple-valued Decision Diagrams (QMDDs) [17], that can synthesize larger input circuits with the minimal number of circuit lines.

Some works have been reported which try to improve upon the quality of a given solution. One approach is based on template matching [12], which gives significant reduction in the number of gates. Yet another approach tries to find some good ordering (permutation) of the output lines, such that the size of the synthesized netlist is minimized [21]. In [21], the problem of finding the optimum output permutation was integrated with the original synthesis problem, and a SAT solver was used to obtain the solution. Since the SAT solver was not able to handle very large instances, the method was applicable to smaller circuits only. This method was called *Exact SWOP*. A heuristic was proposed which searched for the best position of the output lines (taken one at a time), which reduces the problem complexity from $O(k!)$ to $O(k^2)$, where k is the number of inputs.

The present work also uses the concept of permuting the output lines such that the cost of synthesis is minimized. The main emphasis of the work is to obtain a good permutation, and not to propose a synthesis algorithm. In fact, any existing synthesis algorithm can be used along with the proposed approach to reduce the cost of the synthesized circuits. The next section first explains the motivation for searching for a good output permutation, and then presents an evolutionary approach for obtaining good orderings based on some cost function. The effectiveness of the tool is validated by running the same on various reversible benchmark circuits, using tools available in [16].

III. Proposed Synthesis Approach

In this section we first explain with the help of an example how the concept of output permutation can help in reducing the cost of the synthesized netlist. Then the formulation of the exact problem that is being addressed is presented, along with a brief discussion of its complexity. Finally, the details of the evolutionary algorithm to solve the problem are presented. The following subsections discuss these issues in more detail.

A. Basic idea

In many of the synthesis approaches, the reversible function $f : B^k \rightarrow B^k$ to be synthesized is expressed as a truth table, in which the position of each output line is fixed. Consider the reversible function specification as shown in Table I. A minimum gate realization for this function specification is shown in Fig. 1. If we reorder the output lines to (x, z, y) and (z, x, y), the corresponding minimum gate realizations are shown in Fig. 2.

TABLE I
EXAMPLE FUNCTION SPECIFICATION

a	b	c	x	y	z
0	0	0	0	0	0
0	0	1	1	1	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	1	0	1
1	1	0	0	0	1
1	1	1	1	1	0

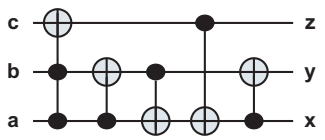

Fig. 1. Minimum realization for output order (x,y,z)

The above example clearly illustrates the fact that the cost of the synthesized netlist depends quite significantly on the output ordering. This observation forms the main motivation behind the proposed work.

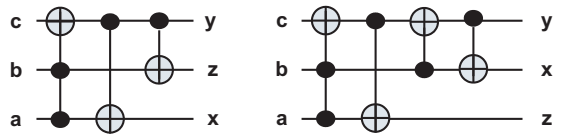

Fig. 2. Minimum realization for output orders (x,z,y) and (z,x,y)

B. Motivation and contribution

With the above motivation, this paper attempts to find a good output permutation, that results in a reversible logic netlist of small number of gates.

For a reversible logic specification with k qubits, since the number of possible output permutations is $k!$, it is computationally infeasible to explore all the possibilities and arrive at the best solution for larger values of k. In [21], the authors attempted to find the optimum solution by formulating the problem as a SAT instance, and use a solver to obtain the solution. As expected, the approach works for smaller values of k only. To reduce the search space from $k!$ to k^2, the authors also presented a heuristic where instead of trying out all possible output permutations, only a small subset was evaluated (based on the best positions of individual output lines).

In this paper, we propose a cost based output ordering technique based on an *Evolutionary Algorithm* (EA), which will give a *good* output ordering, even for reversible functions with larger values of k. The overall schematic diagram of the approach is depicted in Fig. 3, where a backend synthesis engine is used along with the proposed tool (shown shaded) developed as part of the present work.

C. Evolutionary algorithm based solution approach

As mentioned earlier, we have used an EA to optimize a given output ordering with respect to synthesizability based on a chosen cost metric. The various aspects of the implementation are discussed in the following subsections.

1) Individual representation: Each individual in the EA formulation represents an ordering of the output variables, and is represented by a linear array. The initial population consists of a set of such individuals, containing randomly generated output permutations. For $k = 6$ outputs, a sample chromosome is shown in Fig. 4.

5	4	0	1	2	3

Fig. 4. An example chromosome

2) Fitness function: In any EA formulation, the fitness function plays a central role. The function, when applied to an individual, will return a numeric value providing a measure of its goodness. In the context of the present problem, the fitness function F when applied to a given output permutation P, should give a measure of the difficulty of synthesizing the given reversible function.

In this work, the fitness function is computed as the Hamming distance between the permutations realized by the reversible function under a given output ordering, and the identity permutation. An identity permutation corresponds to the case where the inputs are directly copied to the outputs without any gates in between. The intuitive justification is that closer a permutation is to the identity permutation, lesser will be the effort to synthesize it.

Algorithm *calculate_fitness (O, P, F)*

Input: Permutation P corresponding to the given reversible specification; output permutation O corresponding to a given individual; number of lines k
Output: Fitness function F of the individual O
begin
  F = 0;
  for $i = 0$ **to** $2^k - 1$ **do**
    begin
      x = decimal equivalent of the k-bit output vector for input i under output permutation O;
      if (*hamm_dist (i, x)* $<$ *threshold*)
        F = F + *hamm_dist (i, x)*;
      else
        F = F + *hamm_dist (i, x)* + *penalty*;
    end
end

In the above algorithm, the parameter *threshold* specifies a limit of Hamming distance between i and x, beyond which the problem of synthesis is regarded as difficult and a *penalty* is added to the function F.

3) Crossover and mutation: We start with an initial population consisting of individuals with randomly generated output permutations. The following procedures are used to generate the individuals of the *next generation* from the *present generation*.

a) The best t solutions of the present generation are copied to the next generation, so that the best solutions are retained. For the present problem, the value of t has been set to 4.
b) Using the roulette wheel method [5], pairs of solutions are selected from the present generation, and a process based on single-point crossover performed on them with some probability, and the resulting pair copied to the next generation. A random crossover point is selected,

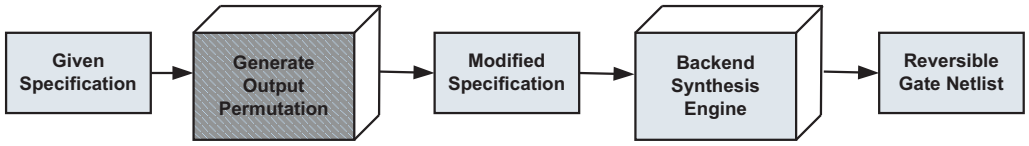

Fig. 3. Overall schematic diagram of the approach

and the new individuals are generated by taking the portion of one individual upto the crossover point and scanning the other from left to right filling up the missing numbers. The process is illustrated in Fig. 5. This is required to ensure that the individuals obtained after crossover are valid permutations. This crossover operator, though slightly different from the Partially mapped crossover (PMX) concept [6] as used in [2], is chosen because it was found to give better results.

c) With a certain probability, some individuals in the next generation are mutated. Several alternatives are followed to carry out mutation: select an output and insert it in at a randomly selected target position, swap two randomly selected outputs, and rotate all the outputs left by one position. One of the three alternatives is chosen randomly. The process is illustrated in Fig. 6 with the help of an example.

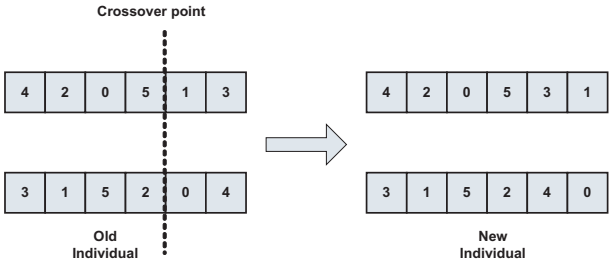

Fig. 5. Illustration of crossover operation

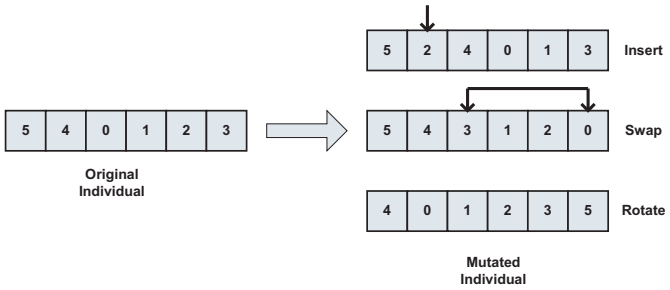

Fig. 6. Illustration of mutation operation

The overall flow of the synthesis algorithm is given below, which is self explanatory.

Algorithm *Selecting Output Ordering using EA*

Input: Permutation P corresponding to the given reversible specification
Output: Ordering of the output lines to minimize cost
begin
  currGen = initial_population();
  for *nGen* = 1 **to** *maxGen*
    begin
      forall chromosome $c \in currGen$ **do**
        calculate_fitness (c, P, F);
          // Calculate fitness and save in an array F
      copy_best (currGen, nextGen, t); // t = 4
      for i = 1 **to** *(pop_size - t)* **step** 2
        begin
          X = *select (currGen)*;
          Y = *select (currGen)*;
          crossover (X, Y, X_{new}, Y_{new});
          mutation (X_{new});
          mutation (Y_{new});
          Add X_{new} and Y_{new} to *nextGen*;
        end
      currGen = nextGen;
    end
end

In the above algorithm, the function *initial_population* generates a set of randomly generated initial output orderings, which constitutes the individuals of the initial generation. Function *copy_best* copies the best k individuals from the current generation *currGen* to the next generation *nextGen*. Function *select* picks up an individual from the current generation using the roulette wheel method. Function *crossover* performs single-point crossover on two individuals X and Y, and generates two new ones X_{new} and Y_{new}. Function *mutation* mutates a given individual using one of the three methods as discussed before.

IV. Experimental Setup and Results

The proposed evolutionary approach explained in the previous section takes a reversible logic specification as input and produces a set of *good* output permutations as the output, ranked by their fitness. Since the SAT based tool used in [21] was not available for experimentation, it was not possible to directly compare our work with [21] with regards to the quality of output permutations it generates for larger input circuits. But comparison have been done for smaller input circuits.

Comparing our method with [21] for smaller input benchmark circuits we observe that number of gates required is same in both the cases, only the time required is less in our method. Moreover, the approach of [21] can be used only for circuits with a smaller number of inputs k (7 or 8 maximum). In contrast the proposed approach can also be used for larger values of k. We have shown results up to $k = 18$. The major advantage of our method in contrast to other techniques based on output permutation is that we need not have to carry out synthesis during the generation of the required permutation which is required by other techniques. All the experiments were carried out on a dual-core Pentium processor with a clock frequency of 2.8 GHz, and running Ubuntu 11.04 version of Linux.

Table II shows the results for some benchmark circuits up to 6 inputs. The table depicts the value of k, the minimum gate count GC_{min} for the original benchmark, number of gates GC and time required for *Exact SWOP* [21] method and our method. The time $Time$ shown is the total time required for ordering and synthesis. We have used the *Exact synthesis* method provided in RevKit [16] for synthesis after generating the desired output permutation. The result clearly shows that the number of gates required is same for both the techniques, but the time required is less in our method. This is because of the fact that Exact SWOP searches for the best permutation among all the $(k!)$ permutations, whereas our method provides a good permutation using an EA based tool.

Since the *Exact synthesis* tool was unable to handle specifications with more than 5 or 6 inputs, for larger specifications we used another synthesis tool available under RevKit [16] as the backend synthesis engine (namely, the transformation based approach [13]). This is a constructive approach that scans the minterms of the given truth table sequentially, and goes on adding gates to a partial netlist. This tool gives results for reasonably larger values of k; however, the results are sub-optimal. Since the objective of the present experimentation is not to generate near optimal solutions, but rather to test the effectiveness of the output ordering created by the EA based tool, the choice is justified.

Table III shows the results of experimentation on various reversible benchmark circuits. The table depicts the value of k, the number of gates g required in the original function specification as well as the quantum cost q and equivalent transistor cost t, and those required in the best out of top three output orderings generated by the proposed tool. The table also depicts the runtime of the EA based tool to generate the output ordering. The quantum cost gives an estimate of the circuit cost in terms of the number of elementary gates [14], while the transistor cost gives an estimate of the circuit cost in terms of the number of CMOS transistors [18]. For circuits with inputs up to 9 from Table III we observe that reductions of up to 78.86% is observed with an average reduction of 18.52%.

Table IV shows the results on larger randomly generated reversible circuit specifications. These specifications were generated synthetically by randomly generating a gate netlist consisting of 50 to 100 gates, and computing the equivalent permutations. The results show that reductions of up to 98% is observed with some of the larger input circuits with an average reduction of 64.4%. As expected the runtime of the EA based tool gets doubled with unit increase in number of inputs, because of a similar increase in the size of the permutations.

TABLE II
SYNTHESIS RESULTS FOR BENCHMARK CIRCUITS

Bench	k	GC_{min}	Exact SWOP		Our Method	
			GC	$Time$ (sec)	GC	$Time$ (sec)
4mod5	5	5	5	7.4	5	0.04
decode24	4	6	5	0.1	5	0.1
gt4	4	4	3	<0.1	3	<0.1
gt5	4	3	1	<0.1	1	<0.1
low-high	4	5	4	0.4	4	0.2
3_17	3	6	5	<0.1	5	0.02
ham3	3	5	3	0.01	3	< 0.01
graycode6	6	5	5	13.5	5	0.1
mod5mils	5	5	5	1.7	5	0.1
rand0	4	8	7	26.4	7	15.64
rand1	4	8	7	28.3	7	3.74
rand2	4	9	8	150.3	8	22.27
rand3	4	9	9	1895.6	9	231.5
rand4	4	9	9	569.9	9	26.99

V. CONCLUSION

An evolutionary approach for reversible logic synthesis with output permutation has been investigated in this paper. The results demonstrate the fact that, using output permutation prior to any synthesis process, the number of gates can be reduced significantly. The main advantage of this method is that it does not require explicit synthesis of the circuits and can scale up to larger number of inputs. This work can be extended in the future to incorporate incompletely specified functions and other methods of functional representations.

REFERENCES

[1] K. Datta, G. Rathi, I. Sengupta, and H. Rahaman. Synthesis of reversible circuits using heuristic search method. In *Proceedings of* 25^{th} *International Conference on VLSI Design*, pages 328–333, 2012.

[2] R. Drechsler, A. Finder, and R. Wille. Improving ESOP-based synthesis of reversible logic using evolutionary algorithms. In *EvoApplications (2)*, pages 151–161, 2011.

[3] K. Fazel and M. A. Thornton. ESOP-based Toffoli gate cascade generation. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 206–209, 2007.

[4] R. Feynman. Quantum mechanical computers. *Optic News*, 11:11–20, 1985.

[5] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Professional, 1989.

[6] D. Goldberg and R. Lingle. Alleles, loci and the travelling salesman problem. In *Intl. Conference on Genetic Algorithms*, pages 154–159, 1985.

[7] D. Grosse, R. Wille, G. W. Dueck, and R. Drechsler. Exact multiple control Toffoli network synthesis with SAT techniques. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 28(5):703–715, May 2009.

TABLE III

SYNTHESIS RESULTS FOR BENCHMARK CIRCUITS (g: GATE COUNT, q: QUANTUM COST, t: TRANSISTOR COST, $time_{GA}$: EA ORDERING TIME (IN SEC))

Benchmark	No. of lines (k)	Original ordering			Proposed EA based ordering				% reduction in gates
		g	q	t	g	q	t	$time_{EA}$	
4_49	4	22	138	352	16	104	248	0.00	27.27
nthprime5	5	48	700	1080	46	614	1008	0.00	4.16
nthprime6	6	133	2753	3432	116	2224	2864	0.00	12.78
nthprime7	7	317	11325	9904	284	10583	8960	0.02	10.41
nthprime8	8	772	40922	27888	730	34569	25440	0.95	5.44
ham7	7	246	9712	8000	52	865	1120	0.04	78.86
hwb4	4	19	115	296	11	31	128	0.00	42.10
hwb5	5	54	570	1064	49	509	968	0.00	9.26
hwb6	6	144	3364	3808	126	2954	3376	0.00	12.50
hwb7	7	343	11710	10608	304	10164	9376	0.03	11.37
hwb8	8	765	39044	27232	739	39477	26592	1.00	3.39
hwb9	9	1808	118792	73272	1722	121917	70896	1.24	4.75

TABLE IV

SYNTHESIS RESULTS FOR RANDOMLY GENERATED CIRCUITS (g: GATE COUNT, q: QUANTUM COST, t: TRANSISTOR COST, $time_{GA}$: EA ORDERING TIME (IN SEC))

Random Specification	No. of lines (k)	Original ordering			Proposed EA based ordering				% reduction in gates
		g	q	t	g	q	t	$time_{EA}$	
rand10_1	10	2250	228110	108776	692	132756	39312	2.37	69.20
rand10_2	10	3085	317015	148872	1668	248892	89680	2.40	45.90
rand11_1	11	1401	340489	84824	549	168547	34840	6.22	60.80
rand11_2	11	5778	776480	314960	1971	378358	118104	6.14	65.90
rand12_1	12	9180	1471859	537600	1393	541397	96680	12.66	84.80
rand12_2	12	10460	1834983	623632	3006	846708	201912	12.50	71.30
rand13_1	13	2482	974450	181712	1770	982851	134456	25.40	28.70
rand13_2	13	21005	4672148	1382664	6236	2973730	472096	25.44	70.30
rand14_1	14	24220	9460213	1794392	2041	2545109	172528	58.90	91.60
rand14_2	14	11103	8231635	912920	9092	6916571	755808	58.80	18.10
rand15_1	15	89349	24305682	6522496	1407	3528477	133024	114.20	98.40
rand15_2	15	99696	32598743	7416192	9738	13642619	881576	114.00	90.20
rand16_1	16	209233	75744459	16528816	3991	24763262	408336	256.20	98.09
rand17_1	17	30798	9708764	3181896	19794	89313670	2100600	550.90	35.73
rand18_1	18	14480	66351028	1592280	9052	70861118	1017800	1201.00	37.49

[8] P. Gupta, A. Agrawal, and N. K. Jha. An algorithm for synthesis of reversible logic circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(11):2317–2329, 2006.

[9] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski. Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(9):1652–1663, September 2006.

[10] R. Khanom, T. Kamal, and M. H. A. Khan. Genetic algorithm based synthesis of ternary reversible quantum circuit. In *Proceedings of* 11^{th} *International Conference on Computer and Information Technology*, pages 270–275, 2008.

[11] M. Li, Y. Zheng, M. S. Hsiao, and C. Huang. Reversible logic synthesis through ant colony optimization. In *Proceedings of Design Automation Test in Europe*, pages 208–212, 2010.

[12] D. Maslov, G. W. Dueck, and D. M. Miller. Toffoli network synthesis with templates. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(6):807–817, 2005.

[13] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings of Design Automation Conference*, pages 318–323, 2003.

[14] M. Nielsen and I. Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2000.

[15] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(6):710–722, June 2003.

[16] M. Soeken, S. Frehse, R. Wille, and R. Drechsler. Revkit: An open source toolkit for the design of reversible circuits. *LNCS*, 7165:64–76, 2012. Selected Papers from the Third International Workshop on Reversible Computation.

[17] M. Soeken, R. Wille, C. Hilken, N. Przigoda, and R. Drechsler. Synthesis of reversible circuits with minimal lines for large functions. In *Asia and South Pacific Design Automation Conference*, pages 85–92, 2012.

[18] M. Thomson and R. Gluck. Optimized reversible binary-coded decimal adders. *Journal of Systems Architecture*, 54:697–706, 2008.

[19] T. Tofolli. Reversible computing. *Tech. Memo-MIT/LCS/TM-151, MIT Lab for Comp. Sci*, 1980.

[20] R. Wille and R. Drechsler. BDD-based synthesis of reversible logic for large functions. In *Proceedings of Design Automation Conference*, pages 270–275, 2009.

[21] R. Wille, D. Grosse, G. W. Dueck, and R. Drechsler. Reversible logic synthesis with output permutation. In *Proceedings of* 22^{nd} *International Conference on VLSI Design*, pages 189–194, 2009.

[22] M. Zhang, S. Zhao, and X. Wang. Automatic synthesis of reversible logic circuit based on genetic algorithm. In *Proceedings of IEEE International Conference on Intelligent Computing and Intelligent Systems*, pages 542–546, 2009.