Automated Synthesis of Generalized Reversible Cascades using Genetic Algorithms

Martin Lukac, Mihail Pivtoraiko, Alan Mishchenko, Marek Perkowski
Portland Quantum Logic Group
Department of Electrical and Computer Engineering
Portland State University
lukacm@ece.pdx.edu, mikhail@ece.pdx.edu, alanmi@ece.pdx.edu,
mperkows@ece.pdx.edu

Abstract.
We propose an automated synthesis of Reversible logic (RL) circuits using Darwinian and Lamarckian Genetic Algorithms (GA). Our designs are in a form of cascades of generalized gates which generalize factorized Exclusive-Or-Sum-of-Products (ESOP) circuits. GA can be used to explore the problem space of combinational functions and here it is used to evolve reversible logic circuits. We emphasize the role of problem encoding - a well-designed encoding leads to improved results. Our method with well-encoded circuits is compared to standard method on classical benchmarks in GA, and shows good results for synthesis of both random functions and benchmark functions with practical meaning, such as adders.

1. Introduction

Reversible logic synthesis is an area of high interest because it provides theoretical support for constructing circuits without the loss of information, and consequently, without energy loss [2,3,4,5]. The necessity for energy loss minimization raises with the increasing number of gates per chip. The energy lost in a logic circuit has two components: one is related to non-ideality of switches and other technological factors, the other to the information loss. While the first component is decreased with time by inventing new technologies and design principles such as adiabatic design, the second is related to information and can be decreased (to zero) only using the reversible design methodologies.

Implementation of reversible logic can increase the limits on power imposed by current CMOS technology at the price of increased area and reduced speed. It has been shown [3] that reversible circuits built using future logically and physically reversible technologies will minimize the loss of energy. With the increase of the number of gates per chip, according to Moore's Law, current standard CMOS technology will not efficiently reduce the switching energy. Consequently, the real advantage of RL over alternative existing approaches is the fact that we are able to build reversible circuits in today's technologies (contrary to quantum logic). Designing an optimized reversible gate or circuit is a highly challenging area of practical interest when high speed is traded off for low power. It is ideal for applications such as cellular telephones, pacemakers, etc. In addition, reversible logic synthesis is used in quantum logic synthesis.

2. Reversible Logic Gates and Circuits Synthesis Using Evolutionary Approaches

In the simplest approach, a Reversible Circuit (RC) is built from Reversible Gates (RG), such as Toffoli, Fredkin or Feynman. For example, Feynman is a gate with two inputs **A** and **B**, and outputs **P = A** and **Q = P ⊕ B,** where ⊕ is an exor functor. Its reversibility is evident because from each pair of output values one can uniquely determine the pair of input values. A Reversible Circuit will in general have an equal number of inputs and outputs and will be described by a one-to-one mapping (a permutation) [5,12].

In this paper we focus on the automated synthesis of RG's and of RC's, which are designed from a starting set of given RG's. This is similar to building classical logic circuits from cell libraries. The goal is to find an arbitrary function by composing a limited set of gates while minimizing the total complexity of the circuit by a heuristic method. The goal of approach is to find the best or optimal combination of gates that satisfies all care minterms of the specified function. By best we mean both minimizing the number/cost of gates and minimizing the total delay. We are confronted with a logic synthesis/optimization combinatorial problem. Different approaches have been used to synthesize reversible logic functions in order to minimize the complexity (see entire literature given below).

Here the focus is on Evolutionary Algorithms (EA), especially we use a Genetic Algorithm (GA)[17]. A GA is based on the principle of evolving a solution, by searching the problem space using evolutionary heuristics, similar to genetic information transmission known from the Nature. It uses a Population of Individuals, where each individual is a possible solution to the studied problem. In other words, each Individual in the population is in this case an RC. Each Individual is then evolved (mutated, crossed-over, or replicated) to the new generation of individuals in order to find the optimal solution. One of important arguments to use evolutionary algorithms and GA in particular for solving problems such as circuit synthesis, is the fact that a GA is only globally optimized by a Fitness function, while all operators on individuals are performed randomly or proportionally to the fitness value. Consequently, a Genetic Algorithm is well situated for usage in high noise problems. RC's and RG's can be well mapped into individuals, because all gates have the same number of inputs and outputs, and the encoding makes the evolutionary operators efficient and easy to apply, as described below. Our primary goal, as already mentioned, is to test a classical ***Darwinian GA***, without any specific problem-dependant configuration, so that in a later stage we will be able to select more efficient operators for an optimal convergence. (A similar research has been already shown to apply to classical binary logic [4] and quantum logic [8,14,15]. There is however a significant difference between RL and Quantum Logic (QL).) Our next goal is to add reversible logic specific optimization operators leading to a ***Lamarckian GA*** algorithm and compare the Darwinian and Lamarckian approaches.

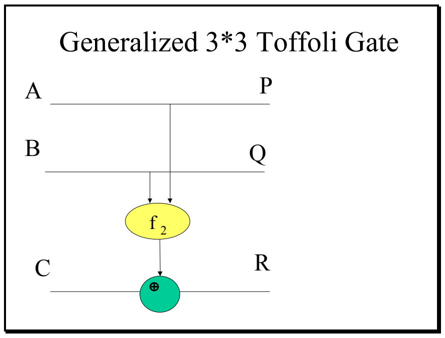
The specification of our GA encoding described below, comes from the RL itself but there are restrictions specific to RL, making it different from both classical Boolean logic synthesis and quantum logic synthesis. The main differences of synthesizing a circuit with reversible gates, as compared to synthesizing a standard binary circuit, are the following:

1. In reversible logic, fan-out of any gate output is not allowed, but such a gate as Feynman can be used to "copy" a signal
2. Several authors assume that there should be ***no loops of gates*** and we follow this assumption here. This is especially true for pseudo-binary circuits of quantum logic where intersection of signals is also not possible – circuits must be planar and use swap gates.

Concluding, the main rules for efficient reversible logic synthesis are the following: **(1)** use as many outputs of every gate as possible, and thus do not create garbage outputs**, (2)** do not create more constant inputs to gates than it is absolutely necessary. Observe that this is quite different from traditional logic design where only one output of gate is used in the synthesis process. Dealing with multi-output gates and the attempt to use all outputs of gate in the best possible way is the difficulty of reversible logic synthesis which causes that most of the current methods are exhaustive, either evolutionary or backtracking /search. Here these two approaches are combined for the first time.

In RL one can introduce constant values on some inputs in order to modify the functionality of the circuit. This is different from Quantum Logic. Even if both are reversible, only in RL one can tune a circuit by inserting constant inputs. Following then the Adiabatic synthesis, these constants can be (in principle) reinserted back into the power supply and they do not cause energy dissipation.

3. Families of Generalized Reversible Logic Gates for Cascades



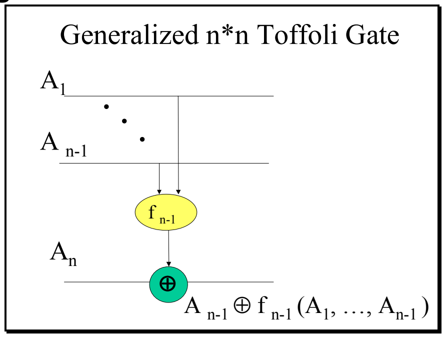
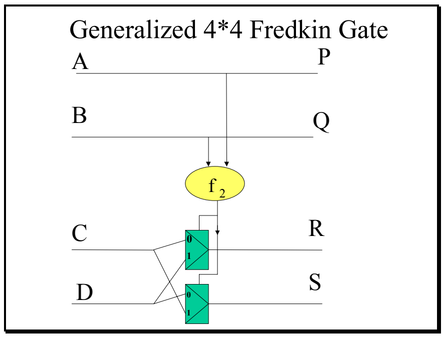
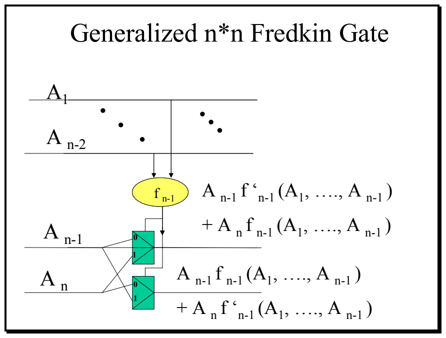


Fig1a,b





Generalized n*n Fredkin Gate
A1
A n-2
f n-1
A n-1 f ' n-1 (A1, …., A n-1)
+ A n f n-1 (A1, …., A n-1)
A n-1
A n
A n-1 f n-1 (A1, …., A n-1)
+ A n f ' n-1 (A1, …., A n-1)

Fig.2a,b

Standard Toffoli gate, called by him Quantum NAND realizes function **$P = A, Q = B, R = A*B \oplus C$**. Toffoli generalized it to a circuit in which the AND gate has an arbitrary number of inputs, called ***multi-input quantum NAND***. Figure 1a presents a generalization of standard Toffoli gate with three inputs and three outputs. We call these gates ***the Toffoli Family***. The symbol of gate at the bottom is Galois Addition (EXOR in binary) and **f_1** is an ***arbitrary*** *binary or multi-valued* function. Figure 1b presents a generalized Toffoli gate for arbitrary number of inputs. Such reversible gates called multi-input quantum NAND have function **f_{n-1}** being an multi-input AND [13,30]. They were generalized as in Fig.1b by De Vos [21,22] to function **f_{n-1}** being an arbitrary binary or multi-valued function. This family of gates was called by De Vos the simple control gates. The concept of Fredkin gate can be generalized to a ***Fredkin family*** with an arbitrary number **n** of inputs as follows: **$P_1 = A_1, P_2 = A_2, \ldots. P_{n-2} = A_{n-2}, P_{n-1} = MUX(f_{n-2}, A_{n-1}, A_n)$, $P_n = MUX(f_{n-2}, A_n, A_{n-1})$** where: **$f_{n-2}$** is arbitrary function of **$n - 2$** variables (in general, binary or multi-valued) being a control variable of the multiplexer, input **A_{n-1}** is a data input 0 and input **A_n** is a data input 1 of the multiplexer. This family has the same applications as Toffoli family, but may be easier to realize in some technologies in which realization of multiplexer is easier than EXOR. Figure 2a presents a 4*4 generalized Fredkin gate with **f_1** being an **arbitrary** binary function. All these gates have been also generalized to multi-valued logic. Figure 2b illustrates a family of n*n generalized Fredkin gates. Kerntopf introduced a family of 3*3 gates that have the maximum number of cofactors [6]. These gates were named Kerntopf gates and found useful for regular structures [7,10,11,12]. The concept of Kerntopf gate can be generalized to a ***Kerntopf family*** with an arbitrary number **n** of inputs as follows: **$P_1 = A_1, P_2 = A_2, \ldots. P_{n-2} = A_{n-2}, P_{n-1} = MUX(f_{n-2}, A_{n-1}, A_n)$, $P_n = DAVIO(f_{n-2}, A_n, A_{n-1})$** where: **$MUX(x,y,z) = x'y + x z, DAVIO(x,y,z) = x'z + y$, f_{n-2}** is arbitrary function of **$n - 2$** variables (in general, binary or multi-valued) being a control variable of the multiplexer, input **A_{n-1}** is a data input 0 and input **A_n** is a data input 1. The simplest generalized Kerntopf gate is shown in Figure 3a. Observe that by Kerntopf family we call all gates that have a mixture of Davio and Shannon gates as they individual output functions. Thus, functions that have only Davio type outputs are Toffoli family, those that have only Shannon type outputs are Fredkin family and those that have mixtures of Davio and Shannon outputs are called Kerntopf family. Observe that these three families do not exhaust of possible reversible gates, because such gates can include other balanced functions as their single outputs, for instance a majority or EXOR functions. An example of a cascade of Kerntopf, Toffoli and Fredkin Family gates is shown in Figure 3b.

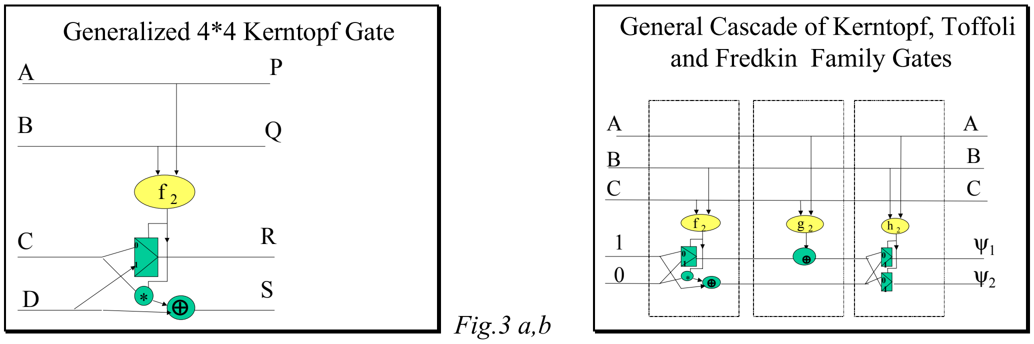

Generalized 4*4 Kerntopf Gate
A
P
B
Q
f 2
C
R
D
S
General Cascade of Kerntopf, Toffoli and Fredkin Family Gates
A
B
C
1
0
f 2
g 2
h 2
Ψ1
Ψ2

Fig.3 a,b

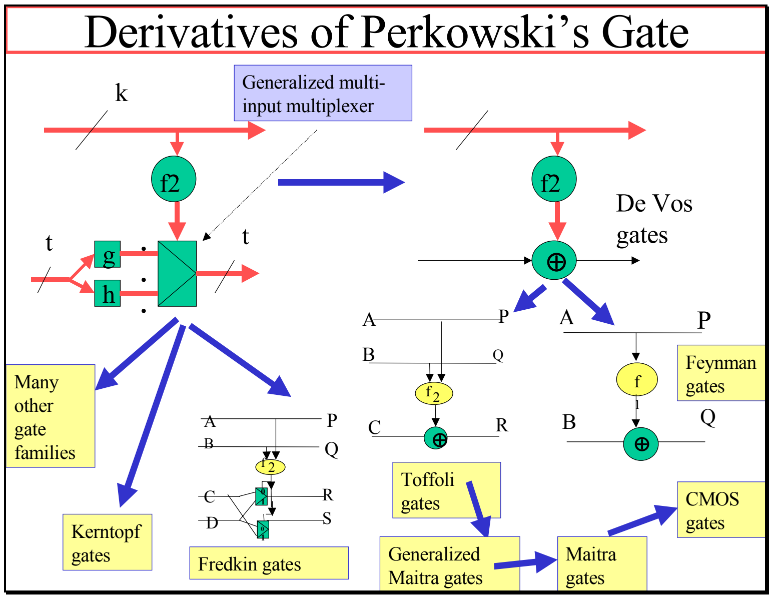
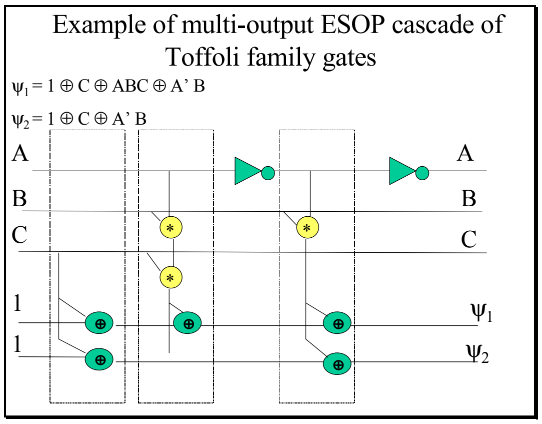
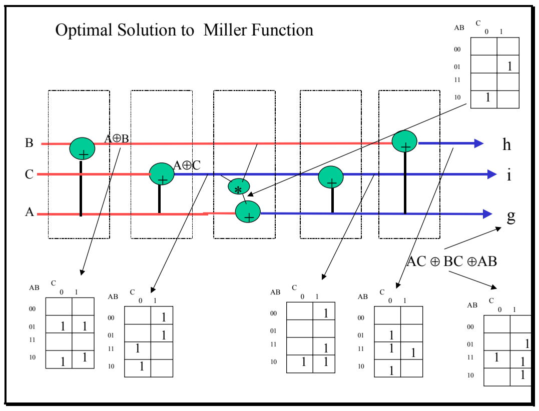


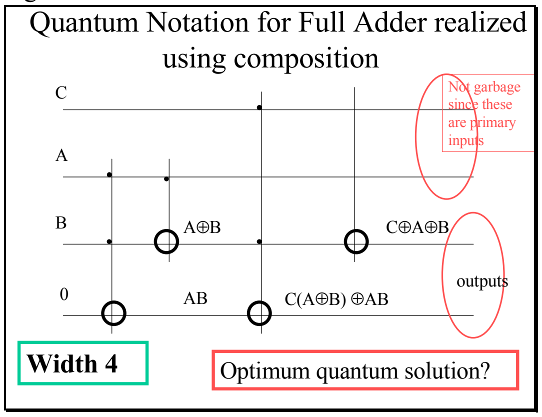
Fig.4.

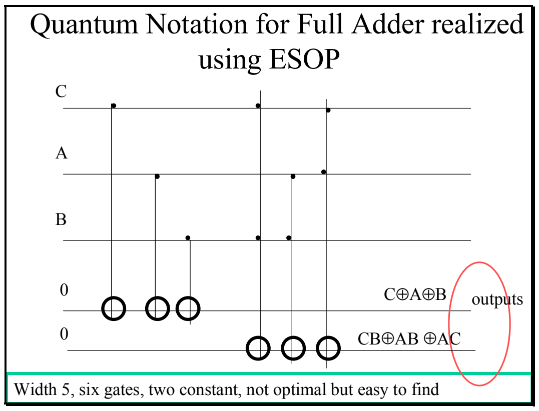
The generalization of Toffoli, Fredkin, De Vos, Kerntopf and many other families has been called the Perkowski's gate. Figure 4 shows how families of gates can be derived from such gate. This powerful generalized gate is thus a generator of families of reversible gates. Function ***f2*** in Figure 4 is an arbitrary binary or multi-valued function and multiplexer is a generalized binary or multi-valued multiplexer, it means it includes not only standard mux but also muxes with several data inputs and multi-valued address inputs for generalized Shannon expansions [1]. In binary case, the mux has **v** address inputs that come from function **f2**, and it has $\mathbf{2}^{\mathbf{v}}$ data inputs. The width of mux is **t** wires, which means that each of $\mathbf{2}^{\mathbf{v}}$ reversible functions at inputs to the mux have **t** inputs and **t** output, and mux has **t** wires in its output. The binary gate in Fig.4 has ***k*** controlling inputs and ***t*** data inputs. Functions ***g*** ,…, ***h*** are arbitrary reversible functions. The construction of gate from Figure 4 is thus recursive. Some MV generalizations of such mux are covered in [1] and in the literature referenced there. In a further generalization [ddd] the data inputs can become control inputs as well, we call them Perkowski's gates with ***mixed data-control***. They generalize what De Vos calls ***complex control gates*** [20,21] (De Vos was restricted to Toffoli-like generalizations only). Synthesis with some special cases of the generalized gates have been discussed in [7,9,10,11,12,18,19,20,21,22,25,26,27,28,29,30]. Interestingly, the synthesis methods from papers before year 2002 were for gates that were not known at this time to be reversible, but now all ESOP-based, factorized-ESOP based and Two-Dimensional Logic Array – based methods [18,19] methods, Multi-Valued Complex Terms and XOR family terms methods [20,25,26,28,29] can be easily adapted to reversible gates presented above. For instance, Figure 5a presents a multi-output ESOP cascade. Every column is a reversible gate. A circuit with gates in which control and data inputs are mixed is shown in Figure 5b. This is an optimal 5-gate solution to the so-called Miller function (based on majority with 3 inputs and 3 polarities). We found optimal solutions to both 3*3 and 5*5 Miller functions (for majority with 5 inputs and 5 polarities. Finally, Figure 6a presents the quantum notation of an adder realized with only Toffoli and Feynman gates. It shows that circuit is ***planar*** and no intersection of wires is allowed. When a general-purpose reversible circuit diagram is converted to quantum notation it is done using quasi-optimal planarization and folding algorithms is [31] which involve adding the (quasi)-minimum number of ***swap gates***. The same adder using ESOP-like circuit is shown in Figure 6b. As we see, ESOP is a good approximation of the cost of cascade from arbitrary gates, and this is why we use ESOP minimization in the feedback loop of genetic algorithms and other exhaustive synthesis methods for quantum and reversible cascades design, section 8 and [31].


Example of multi-output ESOP cascade of Toffoli family gates
$\psi_1 = 1 \oplus C \oplus ABC \oplus A' B$
$\psi_2 = 1 \oplus C \oplus A' B$
A
B
C
1
1
A
B
C
ψ_1
ψ_2



Figures 5a and 5b


Quantum Notation for Full Adder realized using composition
C
A
B
0
A⊕B
AB
C(A⊕B) ⊕AB
C⊕A⊕B
Not garbage since these are primary inputs
outputs
Width 4
Optimum quantum solution?


Quantum Notation for Full Adder realized using ESOP
C
A
B
0
0
C⊕A⊕B
CB⊕AB ⊕AC
outputs
Width 5, six gates, two constant, not optimal but easy to find

Figures 6a and 6b

4. Genetic Algorithm for Reversible Cascade Synthesis

To synthesize a reversible circuit (or a new reversible gate in particular) we have chosen two approaches: **(1)** use only classical gates such as: Feynman, Fredkin, Toffoli (all presented above), Margolus (there are many Margolus gates, all of them have three muxes, example is **P=MUX(A,B,C), Q=MUX(C,A,B), R=MUX(B,C,A)**), Swap (**P=B, Q=A**), Inverter (**P=not(A)**) , and Wire (**P=A**). **(2)** Use our generalized families of gates from previous section. Various gates are combined in order to obtain a circuit that optimizes certain synthesis goal. For this purpose we use similar approach as in [8], where we have obtained correct results that were comparable with previous results but generated more efficiently. Here we use at first a Genetic Algorithm with classical genetic operators (Darwinian GA). Representation of binary reversible logic is a special case of quantum logic representation, thus the automated synthesis of reversible logic circuits is well suited for parsing circuit into parallel blocks [8]. The GA is used to evolve reversible circuits, by searching the space of all* possible circuits defined by the initial set of gates (a starting set). The fitness-optimized recombination of positions of different gates can eventually lead to an optimal solution. Consequently, the convergence of the whole algorithm is based on the fitness function (1). Once again our goal is to test a generic approach to RL synthesis, hence we focus on benchmarks that we propose to be commonly accepted for comparing RL and QL synthesis algorithms.

A GA has a finite set of individuals, with each individual reversible circuit encoded as in Figure 7. Each circuit is coded into a chromosome, being a unit on which genetic operators act. Each individual is a potential solution to the searched problem. We use string/object representation and each individual is encoded in a sequence of parallel blocks, which can be easily manipulated. A good encoding is one of the major ingredients for fast convergence of evolutionary algorithms. An inadequate chromosome encoding can result in not finding the optimal solution or even in not finding a solution at all. The encoding used here is not optimal for minimal circuits but is good for an exploration of a large problem space. In logic synthesis that uses only a finite number of basic gates, the method for optimal solution could be an exhaustive search of the problem space. This results from a fact that since the solution has a limited number of **k** gates, the complete enumeration of possible gate connections from a given set of gates and with no more than a total of **k** gates leads to the optimal solution. For obvious reasons this method is not applicable to larger functions or to infinite gate sets of quantum logic, and more sophisticated searching strategies are required.

5. Fitness Function for Darwinian GA

Fitness Function is another major part of a GA. On one hand each individual is modified with pseudo random genetic operators (mutation, crossover) so as to explore its neighborhood, while on the other hand it is the Fitness Function that selects individuals that will survive to the next generation. It is very important to use a correct fitness function, because the convergence to the solution can be too fast or too slow. The fitness function used here is described by a single equation, however other variants of this function were explored in the presented research. The initial fitness function is defined as:

$$F_i = \frac{1}{1 + error_i} - \Lambda_i \qquad (1)$$

with ***F*** is the fitness for ***i***-th individual and error is the calculated error for this individual***.*** ***Λ*** is the penalization of the circuit, increasing with its length. Initially we were using this Fitness function. However to explore possibilities of recombination, in later experiments we dropped the penalization. This consequently leads to obtaining longer circuits, however this is useful for our exploration.

A, B, D	A'	B'	C'
000	0	0	0
001	0	0	1
010	0	1	0
011	0	1	1
100	1	0	0
101	1	0	1
110	1	1	1
111	1	1	0

Table 1: *Example of a desired Function*

The Fitness of each individual is based on the evaluation error. This error is obtained by by comparing the desired output values of the circuit with the look-up table description of the function under optimization. It is important to note that here we evolve the circuits functionally verified for tautology with their specifications, and not only the closest possible circuits, as it is common in data mining and machine learning applications of GA by other authors. In this case using constants on circuit's input terminals is allowed and even necessary to be able to create the desired function. For example if Fredkin gate which is defined on three pairs of input/output by P,Q,R : P,PQ+¬PR,¬PQ+PR is used with P = 0, then it permutes the outputs on Q,R to R,Q. The error is then defined by the following equation, but only on the desired outputs:

$$error = \sum_{i=1}^{n}\sum_{j=1}^{2^n}\left|U_{ij} - S_{ij}\right| \quad S,U \in U(2^n) \quad (2)$$

with **U** is the correct output value and **S** is the value obtained from the circuit on the desired output **i** for desired pattern **j**. In other words for each possible input combination of the circuit we compare the correct output values with the one obtained from the circuit - Table 1. By normalizing this error by the number of wires and patterns we obtain unitary error for each individual, next used in the Fitness calculation as in (1) above. The Fitness function applied by us here is good for arbitrary number of inputs and outputs of the original circuit and applies also to incompletely specified functions.

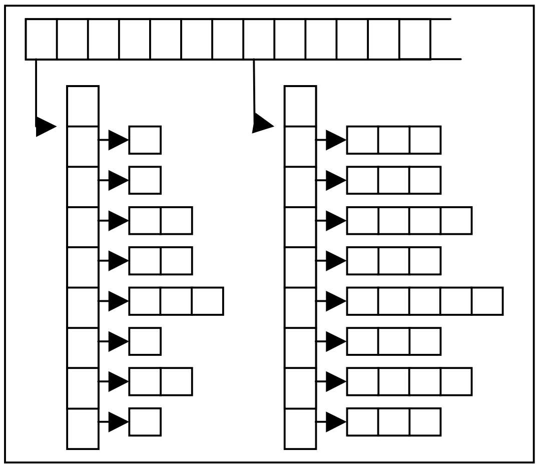


Figure 7. Global representation of an Individual and of a population in a GA. In the upper part the population is represented where each individual has a number. Each individual is represented by a string of objects representing an RL circuit. First place on the chromosome is assigned to the number of inputs/outputs of this circuit, followed by a sequence of parallel blocks, each containing gates with the sum of all wires equal to the number of wires in the circuit.

6. Role of Encoding in Exploring the Reversible Cascade Design Space

Each circuit is modified using evolutionary principles in order to explore the problem space. We use here only the usual genetic operators that are sufficient to operate on the individual circuits. The representation of the individuals, as shown in Figure 1, is quite similar to [8]. However, the global settings are different, so as the operators have to be adapted in a new context. The problem of encoding complex structures into GA is encountered when the elements of individuals are not simple strings. Moreover the encoding must provide for fair distribution of application of operators, by which we mean that each block in any chromosome needs to have the same probability as other block to be selected for a genetic operation. Different encodings were used [8] and most of them satisfy the property described above. Similarly, our encoding here is very simple, while all information necessary is present intrinsically. Figure 8 shows details of a possible chromosome representation. The representation used by the GA is an array of parallel blocks (ordered from input to output), and each of these blocks is composed from gates (in wire order). Figure 8 illustrates an encoding of a Toffoli gate AB ⊕ C with the EXOR gate on C wire. It is an extreme case and needs to be used only if constants are inserted, but in most of our experiments the SWAP gate as presented in Figure 8 was not used. As can be seen on both figures, each circuit is parsed into parallel blocks that determine the overall length of the circuit. The genetic operators will act locally upon these parallel blocks, and in effect globally on the whole circuit by reducing or increasing its size.

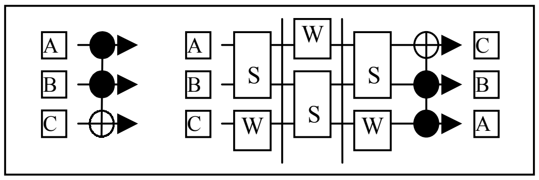


Figure 8. Example of encoding of a three input reversible gate, connected in inverse order. A,B,C are inputs, W is wire, S is Swap gate. The major gate here is CCnot or Toffoli, AB ⊕ C.

The operators used here are Mutation, Crossover and Replication. A GA is an algorithm that can be simulated on a parallel machine while evaluating individuals, however only one generation of individuals can be computed at a time. Steps of a GA are: evaluate fitness of all Individuals, replicate them according to selection rule to the new population, apply crossover and mutation, and replace parents by child population.

Mutation operator here is as simple as possible. However specific actions were defined in order to obtain a big diversity of results. Consequently the mutation can: change a gate to another one (with the same or different number of inputs), change one gate's position in a parallel block, remove a gate (replacing it by wire) and add a new gate (generally a new parallel block needs to be created to preserve the constant number of wires in the circuit).

***Table 2**: the complete set of gates used in a non restricted starting set. Wire gate was always present in all our simulations.*

Number of wires	Gates
1	Wire, Inverter
2	Feynman (Cnot), Swap
3	Fredkin, Toffoli
4	Margolus

The Crossover operator takes generally two individuals and interchanges parts of their chromosomes. This is done only in the case when both circuits have the same number of wires. The temptation to implement a crossover operator that would be able to interchange arbitrary parts from two different genes has been not yet explored, because for a high percentage of cases this is not possible without modifying the structure of both circuits. This technique would completely remove the possibility of preserving unmodified building blocks in the chromosome. Here we used a crossover randomly deciding whether to use one point crossover (cutting two chromosomes at one place, and interchanging their respective halves) or to make two-point crossover (better for building blocks preservation).

The selection operator selects individuals from the current population to the next one. Child from the next generation will replace the not replicated individuals. In this work we use only the Stochastic Universal Sampling operator for selection, because it provides a higher preservation of the population diversity and avoids a high number of local minima. These operators provided all necessary tools to manipulate chromosomes and did not require any specific adjustments.

7. Experimental results of Darwinian GA

Two major setting of the GA were used, especially the Mutation operator was explored. First the impact of this operator can be separated in two distinct categories of influences: local and global. As our representation allows parsing the circuit in blocks, the mutation operator can be used to modify only a single block, i.e. a small random circuit. In this first part, the following settings of the GA were used:

- Constant size of the population (100 – 150) individuals.
- Probability of mutation is from the interval [0.1, 1.0]
- One point crossover probability from [0.3, 0.8].
- Each time an individual was selected for mutation, a random decision selected whether a parallel block is inserted or removed. Each new added segment consists of random selected gates from the starting set.

For the experimentation we use simple benchmarks consisting of any unitary gates used in the starting set (Table 2). The gate to be found was present in the starting set.

# of inputs	Number of individuals	Number of generations	Real gate found	Real Time
2	10/50	10/1	*	< 1 Min
3	10/50	10/1	*	< 1 Min
4	10/50	10/1	*	< 1 Min

.

Table 3*: Global results on basic benchmarks. A * means the real model of the gate was found exactly.*

This set of tests was to verify our non-traditional approach to the RL synthesis. The approach we use here is to use a GA as a block manipulator. Contrary to classical GAs where flipping one element in the chromosome directly affects the result, the element here is a single parallel block randomly synthesized. Consequently, the algorithm is confronted with two types of modifications: one selected by the mutation operator and one completely stochastic (the parallel block). Consequently each circuit can change its structure in one mutation step, but only by one block. This limits a too large exploration in one step. That explains the reason for using a high probability of mutation justified by a local quantified search. This type of exploration is another to the random mutation. In classical mutation the operators directly modify the chromosome (by flipping a bit) while here the circuit is modified by a larger block which completely random and placed over all wires in the circuit. Two consequences are implied from previous statement: first a higher number of generations will be required in the case the solution is not found randomly and second the modification of the circuit is normalized. This type of approach can be used especially if one defines basic parallel blocks instead of using single gates. It can be argued that the high mutation probability in our approach changes the GA to a random search. This criticism is, however, only partially true which will be discussed in the last part of the paper.

Another test was to synthesize a random function. The result is shown in Figure 11. The fitness of the first individual demonstrates that its outputs correspond to the target function in all minterms. As it can be seen, 200,000 generations were required to find this solution. This shows that our approach (using block mutation) does not work well for bigger circuits in which parallel blocks have to be randomly synthesized. However, once again if one has optimal blocks for some particular function, they can be efficiently used to evolve a successfully optimized circuit. An important part in this section of experiments was a synthesis of a full adder taken from [7]. We were only able to find an alternative with one constant inserted. Number of generations was 450. The circuit found is presented in Figure 9.

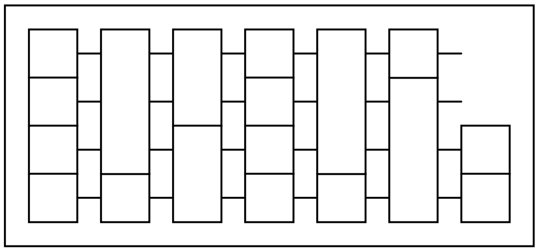


Figure9. Adder found by the GA. F is Feynman, I is Inverter, W is wire and T is Toffoli gate. The optimal design of this circuit has two Toffoli and two Feynman gates [7].

This has a consequence that some gates remain locked in certain particular configurations. The problem then is to know on which wire to apply the constant input. As our only possibility to evaluate a circuit (with constants allowed) is to determine the difference between circuit's output and the searched function, the search of correct output is computationally very requiring on resources because of all possible combinations of constants present in the circuit.

The next experiment set treated mutation in the opposite way: the mutation was used at all levels. For each mutation, the operator selects what type of mutation will be performed (adding/removing segment, adding removing a gate, flipping gates on wires). The mutation probability is set to small values (0.01 to 0.1) and the crossover is maintained in the same range as previously.

Main goal in this section was to look at the ability of the GA under classic constraints to build similar or better circuits for a non-trivial function (compare with the case when the GA was searching for a single gate). For this purpose we first explore the possibility of synthesizing any unitary gate with the number of inputs higher or equal to 3. For this we have selected to synthesize the Toffoli, Fredkin gates, and the adder. The summary of the results is given in Table 4. The number of individuals in the population was set up either to 10 or 100 individuals.

Circuit/Gate	Number of generations	Real Time	Exact/ Similar
Toffoli	5/1	0	*/*
Fredkin	5/1	0	*/*
Adder	?/40000	10 min	0/*

***Table 4**: Summarized results for the synthesis of circuits having three or more inputs.*

There are two types of results in the table. As in the previous experiment sets, one can introduce constants on some inputs in Table 4, the number of generations required to find a solution is indicated respectively for a population size of 10 and 100 individuals, respectively. The Exact/Similar column indicates whether a solution was found (0 – not found, * - found). In the case of the adder we were looking for one proposed by [7] but we only found the one from Figure 10, which has one Feynman gate more than the exact minimum solution from Figure 6a. Moreover the mutation operator has the power to turn gates upside down (Feynman gate with control bit on the second input and XOR on the first one). The number of generations necessary to find the result was 200,000, that is approximately 2 minutes real time. This result indicates the necessity to improve the quality of our genetic operators. However even if a very big number of iterations were required, the real time was practically acceptable.

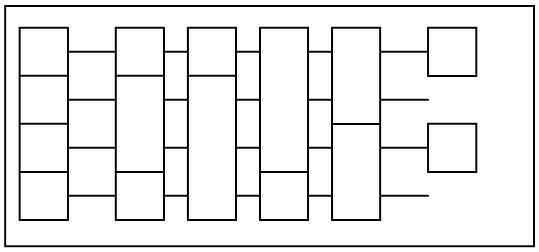


Figure 10. Another Synthesized Adder, which has only one extra gate, IF.

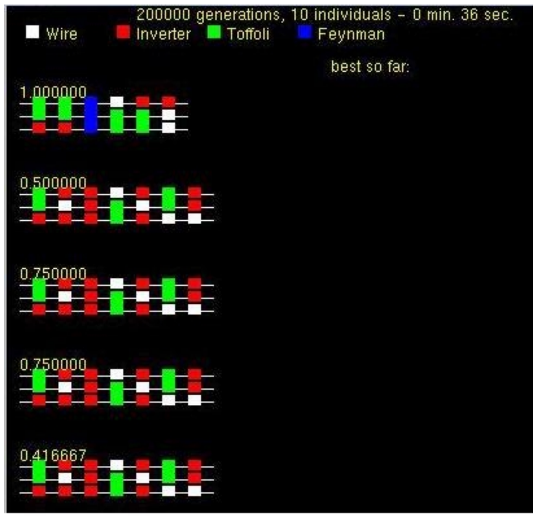


Figure 11. Solution to a random problem search. Here number of individuals in the population is 10. Probability of mutation is 0.8 and of crossover 0.8.

8. Lamarckian Genetic Algorithm for Design of a Cascade with Generalized Reversible Gates

As presented above, the role of fitness function is fundamental to the success of any evolutionary algorithm (or any search algorithm). While ***entropy*** and ***coincidence count*** methods were tested, as well as the method presented here, they did not give a good evaluation of the distance of two functions, given the individual function **F** represented by a chromosome and the goal function **G** to be synthesized (these functions can be both multi-output). It was found that the best function to evaluate the distance of functions **F** and **G** is *min*(**ESOP(F ⊕ G)),** which means that an EXOR of the functions **F** and **G** is calculated and the resulting function **F ⊕ G** is minimized by an ESOP minimizer [20,26,27,29]. The number of terms, number of literals or weighted sum of numbers terms and literals in the minimized ESOP expression can be calculated as the distance evaluation for cascade. Such cascade is designed from gates that are a superset of the quantum NAND gates used in standard ESOP (see Figure 5a). Thus, the term-cost of ESOP is the upper bound of the number of gates of the „remainder circuit“ **F ⊕ G** of the cascade under synthesis. The Lamarckian GA is similar to the Darwinian GA. It uses all operators and strategies of Darwinian GA presented above, together with ESOP minimizer Exorcism-4 [27] as its component algorithms. Main difference is the method to evaluate the fitness function, and Exorcism-4 is used in this process. It is also used to complete the cascade**: FULL_CASCADE (G) = CASCADE(F**) **⊕ ESPOP (F⊕G)** being a new chromosome of an Individual. Now the cascades that have best values of fitness function are treated as new chromosomes and returned back to the population of mixed general/ESOP cascades that are next subject to Darwinian operators on chromosomes and the above presented Lamarckian operator based on ESOP. Observe that a mixed cascade has arbitrary gates in its left and multi-input, multi-output quantum NANDs at its right, but during the run of the Lamarckian GA the part composed of arbitrary gates grows higher as a percentage of all gates in the cascade, while the total length of the solution cascade shortens.

Observe that Lamarckian GA is always convergent, because EXOR of any cascade for **F** designed so far by the Darwinian GA and the minimized ESOP of the remainder function **F ⊕ G** is a correct realization of the given goal function **G**, [31]. The algorithm creates thus the sequence of the cascades of the decreasing cost, with smaller and smaller ESOP component. The experimental results of Lamarckian GA will be available during the conference.

9. Conclusions

The paper introduced four new ideas:

(1) The concept of "Perkowski's Gate" - a very general reversible gate that serves as a generator of families of reversible gates. These families include the family of Toffoli-like gates from De Vos [21,22], Fredkin-like and Kerntopf-like gates from [7,9], and others [6,25,26,27,31]. The early variants of such gates were introduced and used for synthesis in an earlier work of Portland Quantum Logic Group.

(2) The idea of using ESOP minimizer as a part of the fitness function for genetic algorithm. This way, a good quality evaluation of the chromosome-circuit can be found.

(3) The concept of Lamarckian genetic algorithm that uses remainder logic which is an ESOP-like circuit from k*k "quantum NAND gates", synthesized by Exorcism-4 to complete the cascade from arbitrary gates found by the standard GA. This way a sort of Lamarckian evolution is simulated, where the new chromosome is created partly by GA and partly by ESOP minimization, thus the influence of environment (ESOP minimization) is inherited in the chromosomes returned to the chromosome pool.

(4) We have shown a successful experiment in RL synthesis. Design of high quality reversible circuits is a difficult problem. In past, several methods have been tried, based on group theory, function decomposition, gate composition, regular and symmetric structures, decision diagrams, wave cascades and backtracking [1,6,7,8,9,10,11,12]. None of them is so far very successful for larger functions. In present paper we implemented an evolutionary approach to be evaluated and compared with the previous approaches.

Current and future research involves:

(1) characterizing systematically properties of the new families of n-input n-output reversible gates for being used in regular structures and developing logic synthesis methods for them (section 3);

(2) design of reversible/adiabatic CMOS circuits for these families;

(3) improving software presented here to work on larger circuits by investigating various evolutionary paradigms and mixing evolutionary and backtracking/search algorithms;

(4) comparing Darwinian and Lamarckian GAs, and various heuristic search algorithms for reversible benchmarks.

10. References

1. Anas Al.-Rabadi and M. Perkowski, "New Classes of Multi-Valued Reversible Decompositions for Three-Dimensional Layout", *Proceedings of the 5th International Workshop on Applications of Reed-Muller Expansion in Circuit Design*, Starkville, Mississippi, USA, August 10-11, 2001, pp. 185-204.
2. W.C. Athas and L."J." Svensson , "Reversible Logic Issues in Adiabatic CMOS", USC, *Information Sciences Institute*.
3. C. Bennett, "Logical reversibility of computation", *I.B.M. J. Res. Dev.*, **17** (1973), pp. 525-532.
4. K. Dill and M. Perkowski**,** ``Baldwinian Learning utilizing Genetic and Heuristic Algorithms for Logic Synthesis and Minimization of Incompletely Specified Data with Generalized Reed-Muller (AND-EXOR) Forms", *Journal of Systems Architecture,* Vol. 47, Issue 6, pp. 477 - 489, June 2001.
5. E. Fredkin and T. Toffoli, "Conservative Logic", *Int. Journal of Theor. Phys.,* **21** (1982), pp. 219-253.
6. P. Kerntopf, "A Comparison of Logical Efficiency of Reversible and Conventional Gates," *Proc. 3rd LDL,* Portland, Oregon, May 31, 2000
7. A. Khlopotine, M. Perkowski and P. Kerntopf, "Reversible Logic Synthesis by Gate Composition," *Proc. 11th International Workshop on Logic and Synthesis*, IEEE and ACM, New Orleans, June 2002.
8. M. Lukac and M. Perkowski, "Evolving Quantum Circuits Using Genetic Algorithm," Proc. of *NASA/DOD Workshop on Evolvable Hardware*, Washington, D.C. July 2002.
9. A. Mishchenko and M. Perkowski, "Logic Synthesis of Reversible Wave Cascades*," Proc. 11th International Workshop on Logic and Synthesis*, IEEE and ACM, New Orleans, June 2002.

10. M. Perkowski, P. Kerntopf, A. Buller, M. Chrzanowska-Jeske, A. Mishchenko, X. Song, A. Al.-Rabadi, L. Jozwiak, A. Coppola and B. Massey, "Regular Realization of Symmetric Functions Using Reversible Logic", *Proceedings of the EUROMICRO Symposium on Digital Systems Design*, Warsaw, Poland, September 4-6, 2001, pp. 245-252.
11. M. Perkowski and P. Kerntopf, "Fundamentals of Reversible Logic and Computing" (Tutorial), *Proceedings of the EUROMICRO Symposium on Digital Systems Design*, Warsaw, Poland, September 4-6, 2001, pp. 254
12. M. Perkowski, L. Jozwiak, P. Kerntopf, A. Mishchenko, A. Al.-Rabadi, A. Coppola, A. Buller, X. Song, Md. M. H. Azad Khan, S.N. Yanushkevich, V.P. Shmerko and M. Chrzanowska-Jeske, "A General Decomposition for Reversible Logic", *Proc. 5th International Workshop on Applications of Reed-Muller Expansion in Circuit Design,* Starkville, Mississippi, USA, August 10-11, 2001, pp. 119-138.
13. J.Preskill, Lecture notes in quantum computing: http://www. Theory.caltech.edu/~preskill/ph229
14. B.I.P. Rubinstein, "Evolving quantum circuits using genetic programming", *Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001),* pp. 144-151 (2001)
15. C.W. Williams, and G.G. Alexander, "Automated Design of Quantum Circuits", *QCQC '98,* Springer-Verlag, pp. 113-125 (1999)
16. S.G. Younis, "Asymptotically Zero Energy Computing Using Split-Level Charge Recovery Logic, *Ph.D. Thesis*, MIT, June 1994.
17. D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning *Addison Wesley*, 1989
18. N. Song and M. A. Perkowski, "A New Design Methodology for Two-Dimensional Logic Arrays," *Proc. of IEEE International Workshop on Logic Synthesis, IWLS '93,* Tahoe City, CA, pp. 1 - 17, May 1993.
19. N. Song, M. Perkowski, M. Chrzanowska-Jeske and A. Sarabi, "A New Design Methodology for Two-Dimensional Logic Arrays," *VLSI Design,* 1995, Vol. 3., Nos. 3-4, pp. 315-332.
20. N. Song and M. Perkowski, "Minimization of Exclusive Sum of Products Expressions for Multi-Output Multiple-Valued Input, Incompletely Specified Functions*," IEEE Transactions on Computer Aided Design*, " Vol. 15, No. 4, April 1996, pp. 385-395.
21. A. De Vos, ``Design of reversible logic circuits by means of control gates", *Proceedings Patmos 2000 Conference, Goettinge,* (Springer Lecture Notes in Computer Science No. 1918) pp. 255-264, 15 September 2000.
22. A. De Vos , ``Control gates as building blocks for reversible computers", *Proc. Patmos 2001 Conference*, Yverdon. paper 9.2. 28 September 2001.
23. V.V. Shende, A.K. Prasad, I.L. Markov and J.P. Hayes, "Synthesis of Optimal Reversible Logic Circuits'', *Proc. IWLS 2002.*
24. K. Iwama , Y. Kambayashi and S. Yamashita, "Transformation Rules for Designing CNOT-based Quantum Circuits," *Proc. of DAC 2002.*
25. M. A. Perkowski, A. Sarabi and F. R. Beyl. Universal XOR Canonical Forms of Switching Functions. *Proc. of IFIP W.G. 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design.* Hamburg, Germany, September 16-17, pp. 27 - 32, 1993.
26. N. Song and M. Perkowski, „Minimization of Exclusive Sums of Multi-Valued Complex Terms for Logic Cell Arrays," *Proc. ISMVL 98,* Fukuoka, Japan, May 1998.
27. A. Mishchenko and M. Perkowski, ``Fast Heuristic Minimization of Exclusive Sums-of-Products," *Proc. RM'2001 Workshop,* August 2001.
28. A. Sarabi, N. Song, M. Chrzanowska-Jeske and M. A. Perkowski. "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays," *Proc. DAC'94,* pp. 321 – 326
29. N. Song and M. Perkowski, "Minimization of Exclusive Sum of Products Expressions for Multi-Output Multiple-Valued Input, Incompletely Specified Functions," *IEEE Trans. CAD,* Vol. 15, No. 4, April 1996, pp. 385-395.
30. T. Toffoli, "Reversible Computing," In *Automata, Languages and Programming*, Springer Verlag, 1980, pp. 632- 644.
31. M. Perkowski, "Generalized reversible gate families and their use in cascade synthesis," report PSU, May 21, 2002.