Improving ESOP-based Synthesis of Reversible Logic Using Evolutionary Algorithms

Rolf Drechsler Alexander Finder Robert Wille

Institute of Computer Science, University of Bremen, Bremen, Germany
{drechsle,final,rwille}@informatik.uni-bremen.de,
http://www.informatik.uni-bremen.de/agra/ger/index.php

Abstract. Reversible circuits, i.e. circuits which map each possible input vector to a unique output vector, build the basis for emerging applications e.g. in the domain of low-power design or quantum computation. As a result, researchers developed various approaches for synthesis of this kind of logic. In this paper, we consider the ESOP-based synthesis method. Here, functions given as *Exclusive Sum of Products* (ESOPs) are realized. In contrast to conventional circuit optimization, the quality of the resulting circuits depends thereby not only on the number of product terms, but on further criteria as well. In this paper, we present an approach based on an evolutionary algorithm which optimizes the function description with respect to these criteria. Instead of ESOPs, *Pseudo Kronecker Expression* (PSDKRO) are thereby utilized enabling minimization within reasonable time bounds. Experimental results confirm that the proposed approach enables the realization of circuits with significantly less cost.

Keywords: Evolutionary Algorithms, Reversible Logic, Synthesis, Exclusive Sum of Products, Pseudo Kronecker Expressions, Optimization

1 Introduction

Reversible logic [11, 1, 21] realizes n-input n-output functions that map each possible input vector to a unique output vector (i.e. bijections). Although reversible logic significantly differs from traditional (irreversible) logic (e.g. fan-out and feedback are not allowed), it has become an intensely studied research area in recent years. In particular, this is caused by the fact that reversible logic is the basis for several emerging technologies, while traditional methods suffer from the increasing miniaturization and the exponential growth of the number of transistors in integrated circuits. Researchers expect that in 10-20 years duplication of transistor density every 18 months (according to *Moore's Law*) will come to a halt (see e.g. [24]). Then, alternatives are needed. Reversible logic offers such an alternative as the following applications show:

- *Reversible Logic for Low-Power Design*
  Power dissipation and therewith heat generation is a serious problem for today's computer chips. Landauer and Bennett showed in [11, 1] that (1) using traditional (irreversible) logic gates always leads to energy dissipation regardless of the underlying technology and (2) that circuits with zero power dissipation must be information-lossless. This holds for reversible logic, since data is bijectively transformed without losing any of the original information. Even if today energy dissipation is mainly caused by non-ideal behaviors of transistors and materials, the theoretically possible zero power dissipation

makes reversible logic quite interesting for the future. Moreover, in 2002 first reversible circuits have been physically implemented [5] that exploit these observations in the sense that they are powered by their input signals only and did not need additional power supplies.

– *Reversible Logic as Basis for Quantum Computation*
Quantum circuits [14] offer a new kind of computation. Here, qubits instead of traditional bits are used that allow to represent not only 0 and 1 but also a superposition of both. As a result, qubits can represent multiple states at the same time enabling enormous speed-ups in computations. Even if research in the domain of quantum circuits is still at the beginning, first quantum circuits have already been built. Reversible logic is important in this area, because every quantum operation is inherently reversible. Thus, progress in the domain of reversible logic can directly be applied to quantum logic.

Further applications of reversible logic can be found in the domain of optical computing [3], DNA computing [1], and nanotechnologies [12].

Motivated by these promising applications, various synthesis approaches for reversible logic have been introduced in the past. They rely on different function representations like truth-tables [13], permutations [16], BDDs [23], or positive-polarity Reed-Muller expansion [9].

In the following, we focus on a method based on *Exclusive Sum of Products* (ESOPs) representations [6]. Here, the fact is exploited that a single product of an ESOP description directly corresponds to an appropriate reversible gate. The cost of the respective gates strongly depends thereby on the properties of the products. Accordingly, the quality of the resulting circuits relies not only on the number of product terms of the ESOP, but on further criteria as well. This is different to conventional logic optimization and, thus, requires an appropriate treatment.

In this paper, an approach is introduced which optimizes a given *Pseudo Kronecker Expression* (PSDKRO) with respect to these criteria. PSDKROs represent a subclass of ESOPs enabling minimization within reasonable time bounds. In order to optimize the PSDKROs, the evolutionary algorithm introduced in [7] is utilized. We describe how this algorithm can be extended to address the new cost models. Experimental results show that this leads to significant improvements in the costs of the resulting circuits. In fact, in most of the cases, the respective costs can be decreased by double-digit percentage points.

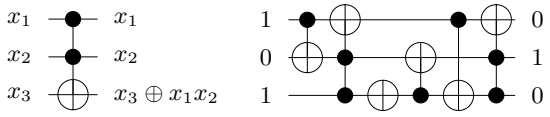
The remainder of this paper is structured as follows. The next section introduces the necessary background on reversible circuits as well as on ESOPs and PSDKROs. Afterwards, the ESOP-based synthesis method is briefly reviewed in Section 3. Section 4 describes the proposed optimization approach. Finally, experimental results are presented in Section 5 and conclusions are drawn in Section 6, respectively.

2 Background

To keep the paper self-contained, this section briefly reviews the basic concepts of reversible logic. Afterwards, ESOPs and PSDKROs are introduced.

2.1 Reversible Circuits

Reversible circuits are digital circuits with the same number of input signals and output signals. Furthermore, reversible circuits realize bijections only, i.e. each



(a) Toffoli gate (b) Toffoli circuit

Fig. 1: Toffoli gate and Toffoli circuit

input assignment maps to a unique output assignment. Accordingly, computations can be performed in both directions (from the inputs to the outputs and vice versa).

Reversible circuits are composed as cascades of reversible gates. The *Toffoli gate* [21] is widely used in the literature and also considered in this paper. A Toffoli gate over the inputs $X = \{x_1, \ldots, x_n\}$ consists of a (possibly empty) set of *control lines* $C = \{x_{i_1}, \ldots, x_{i_k}\} \subset X$ and a single *target line* $x_j \in X \setminus C$. The Toffoli gate inverts the value on the target line if all values on the control lines are assigned to 1 or if $C = \emptyset$, respectively. All remaining values are passed through unaltered.

Example 1. Fig. 1(a) shows a Toffoli gate drawn in standard notation, i.e. control lines are denoted by ●, while the target line is denoted by $\oplus$. A circuit composed of several Toffoli gates is depicted in Fig. 1(b). This circuit maps e.g. the input 101 to the output 010 and vice versa.

Since the number of gates in a cascade is a very poor measure of the cost of a reversible circuit, different metrics are applied (sometimes depending on the addressed technology). In this work, we consider quantum cost and transistor cost. While the quantum cost model estimates the cost of the circuit in terms of the number of elementary quantum gates [14], the transistor cost model estimates the cost of the circuit in terms of the number of CMOS transistors [20]. Both metrics define thereby the cost of a single Toffoli gate depending on the number of control lines. More precisely:

- *Quantum cost model:* The quantum cost of a Toffoli gate is given in Table 1(a) (using the calculations according to [17]), where c denotes the number of control lines for the gate and n denotes the number of circuit lines. Note that the quantum cost depend not only on the number c of control lines, but also on the number $(n - c + 1)$ of lines neither used as control line or target lines. The more lines are not in the set of control lines, the cheaper the respective gate can be realized.
- *Transistor cost model:* The transistor cost of Toffoli gate increases linearly with $8 \cdot s$ where s is the number of control lines in the gate (see Table 1(b)).

The cost of a circuit is the sum of the costs of the individual gates. For example, the gate shown in Fig. 1(b) has quantum cost of 14 and transistor cost of 56.

2.2 Exclusive Sum of Products and Pseudo Kronecker Expressions

Exclusive Sum of Products (ESOPs) are two-level descriptions of Boolean functions. Each ESOP is composed of various conjunctions of literals (called *products*). A *literal* either is a propositional variable or its negation. To form the

Table 1: Cost of reversible circuits

(a) Quantum cost

c	$(n-c+1) \geq$	cost
0		1
1		1
2		5
3		13
4	2	26
4	0	29
5	3	38
5	1	52
5	0	61
6	4	50
6	1	80
6	0	125

c	$(n-c+1) \geq$	cost
7	5	62
7	1	100
7	0	253
8	6	74
8	1	128
8	0	509
9	7	86
9	1	152
9	0	1021
> 9	$c-2$	$12(c+1)-34$
> 9	1	$24(c+1)-88$
> 9	0	$2^{c+1}-3$

(b) Transistor cost

s	cost
0	0
1	8
2	16
3	24
4	32
5	40
6	48
7	56
8	64
9	72
10	80
> 10	$8 \cdot s$

ESOP, all products are combined by Exclusive ORs. That is, an ESOP is the most general form of two-level AND-EXOR expressions.

Since the minimization of general ESOPs is computationally expensive, several restricted subclasses have been considered in the past, e.g. *Fixed Polarity Reed-Muller Expressions* (FPRMs) [15] and *Kronecker Expressions* (KROs) [4]. As an interesting alternative, *Pseudo Kronecker Expressions* (PSDKROs) have been proposed, since the resulting forms are of moderate size, i.e. close to ESOPs, and the minimization process can be handled within reasonable time bounds. The following inclusion relationship can be stated for ESOPs and PSDKROs: $FPRM \subseteq KRO \subseteq PSDKRO \subseteq ESOP$.

Let f_i^0 (f_i^1) denote the *cofactor* of the Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ with $x_i = 0$ ($x_i = 1$) and $f_i^2 := f_i^0 \oplus f_i^1$, where $\oplus$ is the Exclusive OR operation. Then, f then can be represented by:

$$f = \overline{x}_i f_i^0 \oplus x_i f_i^1 \qquad \textit{(Shannon, abbr. S)} \tag{1}$$

$$f = f_i^0 \oplus x_i f_i^2 \qquad \textit{(positive Davio; abbr. pD)} \tag{2}$$

$$f = f_i^1 \oplus \overline{x}_i f_i^2 \qquad \textit{(negative Davio; abbr. nD)} \tag{3}$$

A PSDKRO is obtained by applying either S, pD, or nD to a function f and all subfunctions until constant functions are reached. If the resulting expressions are expanded, a two-level AND-EXOR form called PSDKRO results.

Example 2. Let $f(x_1, x_2, x_3) = x_1x_2 + x_3$. If f is decomposed using S, we get:

$$f_{x_1}^0 = x_3 \text{ and } f_{x_1}^1 = x_2 + x_3$$

Then, decomposing $f_{x_1}^0$ using pD and $f_{x_1}^1$ using nD, we get:

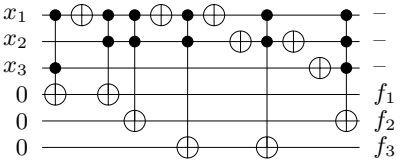
$$(f_{x_1}^0)_{x_3}^0 = 0 \textit{ and } (f_{x_1}^0)_{x_3}^2 = 1$$
$$(f_{x_1}^1)_{x_2}^1 = 1 \textit{ and } (f_{x_1}^1)_{x_2}^2 = 1 \oplus x_3$$

Finally, again pD is applied for $(f_{x_1}^1)_{x_2}^2$:

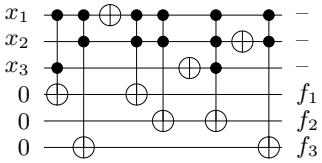
$$((f_{x_1}^1)_{x_2}^2)_{x_3}^0 = 1 \textit{ and } ((f_{x_1}^1)_{x_2}^2)_{x_3}^2 = 1$$

x_1	x_2	x_3	f_1	f_2	f_3
1	-	1	1	0	0
0	1	-	1	1	0
1	1	-	0	0	1
0	0	-	0	0	1
0	1	0	0	1	0

(a) ESOP



(b) Basic approach



(c) W/ reordering

Fig. 2: ESOP-based synthesis

Thus, by expanding the respective expressions, the following PSDKRO description for f results:

$$f = \overline{x}_1 x_3 \oplus x_1 \oplus x_1 \overline{x}_2 \oplus x_1 \overline{x}_2 x_3$$

3 ESOP-based Synthesis

In this work, evolutionary algorithms are applied in order to improve the ESOP-based synthesis method originally introduced in [6]. For a given function $f : \mathbb{B}^n \to \mathbb{B}^m$, this approach generates a circuit with $n + m$ lines, whereby the first n lines also work as primary inputs. The last m circuit lines are respectively initialized to a constant 0 and work as primary outputs. Having that, gates are selected such that the desired function is realized. This selection exploits the fact that a single product $x_{i_1}, \ldots x_{i_k}$ of an ESOP description directly corresponds to a Toffoli gate with control lines $C = \{x_{i_1}, \ldots x_{i_k}\}$. In case of negative literals, NOT gates (i.e. Toffoli gates with $C = \emptyset$) are applied to generate the appropriate values. Based on these ideas, a circuit realizing a function given as ESOP can be derived as illustrated in following example.

Example 3. Consider the function f to be synthesized as depicted in Fig. 2(a)[1]. The first product $x_1 x_3$ affects f_1. Accordingly, a Toffoli gate with control lines $C = \{x_1 x_3\}$ and a target line representing the primary output f_1 is added (see Fig. 2(b)). The next product $\overline{x}_1 x_2$ includes a negative literal. Thus, a NOT gate is needed at line x_1 to generate the appropriate value for the next mappings. Since $\overline{x}_1 x_2$ affects both, f_1 and f_2, two Toffoli gates with control lines $C = \{x_1 x_2\}$ are added next. Afterwards, a further NOT gate is applied to restore the value of x_1 (needed again by the third product). This procedure is continued until all products have been considered. The resulting circuit is shown in Fig. 2(b).

Note that thereby the order in which the respective products are traversed may have a slight impact on the resulting circuit cost. For example, the line x_1 in the circuit from Example 3 is unnecessarily often be inverted. This can be avoided by treating the respective products in a different order as shown in Fig. 2(c). Here, the two product terms with positive literals only were considered first. Afterwards, the products including $\overline{x}_1$, $\overline{x}_1 \overline{x}_3$, and, finally, $\overline{x}_1 \overline{x}_2$ have been handled. This leads to a reduction in the number of NOT gates by 3. In the

[1] The column on the left-hand side gives the respective products, where a "1" on the i^{th} position denotes a positive literal (i.e. x_i) and a "0" denotes a negative literal (i.e. $\overline{x}_i$), respectively. A "-" denotes that the respective variable is not included in the product. The right-hand side gives the respective primary output patterns.

following, a reordering scheme as introduced in [6] is applied to generate the circuits.

Overall, having an ESOP description of the function f to be synthesized, a reversible circuit realizing f can easily be created using the reviewed approach. However, the quality of the resulting circuits strongly depends on the following properties of the given ESOP:

- The number of products (since for each product, a Toffoli gate is added to the circuit),
- the number of primary outputs affected by a product (since for each affected primary output, a Toffoli gate is added to the circuit), and
- the number of literals within a product (since for each literal, a control line needs to be added which causes additional cost as shown in Table 1).

These criteria contradict with the optimization goals applied in common Boolean optimization approaches (e.g. EXORCISM [19]), where usually only the number of product terms is reduced. In contrast, considering ESOP-based synthesis, a function description including more products might be better if instead the number of literals within these products is smaller. Then, although even more gates have to be added, these gates are of less cost. Determining a "good" ESOP representation trading-off these contradictory criteria is thereby a non-trivial task. The next section introduces an evolutionary algorithm addressing this problem.

4 EA-based Optimization

In order to optimize a given ESOP description with respect to the criteria outlined in the previous section, the approach introduced in [7] is utilized. This approach optimizes PSDKRO descriptions – as mentioned above, a subclass of ESOPs enabling efficient optimization. In this section, we briefly review the essential parts of the algorithm and describe the extensions to address the new cost models.

4.1 General Flow

In [7], PSDKROs are optimized using *Reduced Ordered Binary Decision Diagrams (ROBDDs)* [2]. Having a BDD representing the function to be optimized, a depth-first traversal over all nodes is performed. Then, a PSDKRO is derived exploiting the fact that for each decomposition (i.e. for each S, pD, and nD) two out of three possible successors f_i^0, f_i^1, and f_i^2 are needed. That is, in order to generate the PSDKRO, for each node the costs of these three sub-functions are determined. Since ROBDDs are applied, f_i^0 and f_i^1 already are available. In case of f_i^2, the respective function representation is explicitly created. Having the respective costs, the two cheapest sub-functions are applied leading to the respective decomposition type for the PSDKRO.

Using this algorithm, a PSDKRO results which is optimal with respect to a given ordering of the ROBDD. However, modifying the variable ordering likely has an effect on the cost of the PSDKRO. Thus, determining a variable ordering leading to the best as possible PSDKRO representation remains as optimization task. Therefore, an evolutionary algorithm described as follows is applied.

4.2 Individual Representation

The ordering of the input variables in the expansion influences the cost of the PSDKRO. To obtain the minimum cost for all orderings, $n!$ different combinations have to be considered, where n denotes the number of input variables. That is, a permutation problem is considered. This can easily be encoded in EAs by means of vectors over n integers. Each vector represents a permutation, i.e. a valid ordering for the ROBDD, and works as individual in the EA. The population is a set of these elements.

4.3 Operators

In the proposed EA-approach, several procedures for recombination, mutation, and selection are applied. Due to page limitations, they are introduced in a brief manner. For a more detailed treatment, references to further readings are provided.

Crossover and Mutation To create an offspring of a current population two crossover operators and three mutation operators are used alternately.

For recombination *Partially Matched Crossover* (PMX) [8] and *Edge Recombination Crossover* (ERX) [22] are applied equally. In our application, both operators create two children from two parents.

PMX: Choose two cut positions randomly. Exchange the parts between the cut positions in the parent individuals to create two children. Validate the new individuals in preserving the position and order of as many variables as possible.

ERX: Create an adjacency matrix which lists the neighbors of each variable in both parents. Beginning with an arbitrary variable, next the variable with the smallest neighbor set is chosen iteratively. Already chosen variables are removed from all neighbor sets.

For mutation three operators are used as follows:

SWAP: Randomly choose two positions of a parent and exchange the values of these positions.

NEIGHBOR: Select one position $i < n$ randomly and apply SWAP with positions i and $i + 1$.

INVERSION: Randomly select two positions i and j and invert all variables within i and j.

Selection During the experimental evaluation, several selection procedures have been applied and the following turned out to be usually advantageous. As parent selection a deterministic tournament between q uniformly chosen individuals is carried out. The best individual is chosen as a parent used for recombination or mutation, respectively.

To determine the population for the next generation, PLUS-selection $(\mu + \lambda)$ is applied. Here, the best individuals of both, the current population μ and the offspring λ, are chosen equally. By this, the best individual never gets lost and a fast convergency is obtained.

4.4 Termination Criterions

The optimization process is aborted if no improvement is obtained for $20 * ln(n)$ generations or a maximum number of 500 generations, respectively. The default termination criterions are chosen based on experiments in a way that the EA provides a compromise between acceptable runtime and high quality results.

4.5 Parameter Settings

In general, all parameters and operators described above are parameterizable by the user. However, by default the size of the population is chosen two times larger than the number of primary inputs of the considered circuit. For the creation of the offspring, recombination is applied with a probability of 35% while mutation is used with a probability of 65%. By this, mutation also can be carried out on newly elements created by recombination.

4.6 Overall Algorithm and Fitness Function

At the beginning of an EA run, an initial population is generated randomly. Each of the individuals corresponds to a valid variable ordering of the ROBDD. In each generation an offspring of the same size of the parent population is created according to the operators described above. With respect to the fitness, the best individuals of both populations are chosen to be in the next generation. If a termination criterion is met, the best individual is returned.

As discussed above, the applied fitness function is thereby different. Instead of minimizing the number of products, further criteria need to be considered. To incorporate this into the EA, the respective cost functions from Table 1 is encoded and integrated in the selection procedure. More precisely, for each individual, the resulting PSDKRO (or ESOP, respectivley) description is traversed and the cost are added according to the quantum cost model or the transistor cost model, respectively. The resulting value is used as fitness for the considered individual.

Example 4. Consider the function *5xp1* from the the LGSynth benchmark library. This PSDKRO description originally has $p_{initial} = 48$ products which is also the optimal result using the original approach from [7]. However, the quantum cost are $qc_{initial} = 1081$. In contrast, if the proposed configuration is applied, a PSDKRO with an increase in the number of products to $p_{qcmin} = 50$ results. But, a circuit with quantum cost of only $qc_{qcmin} = 865$ can be derived. This shows that a decreasing number of products not coincidently means decreasing quantum cost or transistor cost, respectively.

5 Experimental Evaluation

The proposed approach has been implemented in C++ utilizing the EO library [10], the BDD package CUDD [18], and the RevKit toolkit [17]. As benchmarks, we used functions provided in the LGSynth package. All experiments have been carried out on an AMD 64-Bit Opteron 2,8 GHz with 32GB memory running linux.

The obtained results are summarized in Table 2. The first columns give the name of the respective benchmarks as well as the number of their primary inputs (denoted by *PIs*) and primary outputs (denoted by *POs*). Afterwards,

Table 2: Experimental evaluation

Benchmark			Quantum Cost				Transistor Cost			
Name	PIs	POs	Init. Cost	Opt. Cost	Impr %	Time s	Init. Cost	Opt. Cost	Impr %	Time s
5xp1	7	10	1181	865	26.8	177.0	1424	1080	24.2	205.6
rd84	8	4	2072	2062	0.5	133.3	2528	2528	0.0	130.2
sym9	9	1	16535	16487	0.3	42.8	5088	5088	0.0	40.7
sym10	10	1	37057	35227	4.9	45.7	8408	7984	5.0	45.4
add6	12	7	5112	5084	5.5	370.1	5232	5232	0.0	348.6
alu2	10	6	5958	4476	24.9	188.2	4824	3960	17.9	165.0
alu4	14	8	79311	43850	44.7	594.1	56752	36784	35.2	494.6
apex4	9	19	59175	50680	14.4	315.3	54400	48552	10.8	338.3
b9	41	21	4237	3800	10.3	1331.2	4040	3632	10.1	1307.7
b12	15	9	1082	1049	3.0	417.8	1176	1112	5.4	412.1
con1	7	2	188	162	13.8	30.4	264	224	15.2	40.2
clip	9	5	5243	4484	14.5	151.8	4472	3808	14.8	164.4
duke2	22	29	11360	10456	8.0	1849.3	10016	9248	7.7	1846.8
log8mod	8	5	1118	941	15.8	102.3	1312	1160	11.6	101.6
misex1	8	7	475	466	1.9	190.0	608	608	0.0	185.0
misex3	14	14	82914	67206	18.9	940.8	72528	58464	19.4	890.5
misex3c	14	14	100481	85330	15.1	1016.6	88144	74544	19.4	850.0
sao2	10	4	6005	5147	14.3	141.6	3200	2704	15.5	154.4
spla	16	46	50399	49419	1.9	2498.5	42424	41672	1.8	2392.4
sqrt8	8	4	605	461	23.8	108.1	672	512	23.8	98.7
squar5	5	8	292	251	14.0	18.7	488	448	8.2	17.0
t481	16	1	275	237	13.8	56.1	352	320	9.1	55.2
table3	14	14	46727	35807	23.4	825.7	40208	30800	23.4	843.2
table5	17	15	54729	34254	37.4	1253.0	45408	28440	37.4	1147.7
ttt2	24	21	2540	2445	3.7	1216.2	2720	2584	5.0	1181.4
vg2	25	8	22918	18417	19.6	564.2	18280	14432	21.1	566.0

the cost of the circuits generated from the initial PSDKRO representation (denoted by *Init. Cost*) and generated from the optimized PSDKRO representation (denoted by *Opt. Cost*) are provided. Furthermore, the resulting improvement (given in percent and denoted by *Impr.*) as well as the needed run-time (given in CPU seconds and denoted by *Time*) is listed. We distinguish thereby between the optimization with respect to quantum cost and the optimization with respect to transistor cost.

As can be seen, exploiting evolutionary algorithms significantly helps to reduce the cost of reversible circuits. For the majority of the benchmarks, double-digit improvement rates are achieved in less than an hour – in many cases just a couple of minutes is needed. If transistor cost is considered, the reductions are somewhat smaller. This was expected as this cost model is linear in comparison to the exponential quantum cost model (see Table 1). In the best case, quantum cost (transistor cost) can be reduced by 44.7% (37.4%) in less than 10 minutes (20 minutes).

6 Conclusions

In this paper, an evolutionary algorithm is applied in order to improve ESOP-based synthesis of reversible circuits. By this, PSDKROs are considered which are a subclass of ESOPs. Reversible circuits received significant attention in the past – not least because of the promising applications in the domain of low-power design or quantum computation. ESOP-based synthesis is an efficient method for synthesis of this kind of circuits. Applying the proposed approach, the results obtained by this method can be improved significantly – in most of the cases by double-digit percentage points.

Acknowledgment

This work was supported by the German Research Foundation (DFG) (DR 287/20-1).

References

1. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev 17(6), 525–532 (1973)
2. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Comp. 35(8), 677–691 (1986)
3. Cuykendall, R., Andersen, D.R.: Reversible optical computing circuits. Optics Letters 12(7), 542–544 (1987)
4. Davio, M., Deschamps, J., Thayse, A.: Discrete and Switching Functions. McGraw-Hill (1978)
5. Desoete, B., Vos, A.D.: A reversible carry-look-ahead adder using control gates. INTEGRATION, the VLSI Jour. 33(1-2), 89–104 (2002)
6. Fazel, K., Thornton, M.A., Rice, J.E.: ESOP-based Toffoli gate cascade generation. In: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing. pp. 206–209 (2007)
7. Finder, A., Drechsler, R.: An evolutionary algorithm for optimization of pseudo kronecker expressions. In: Int'l Symp. on Multi-Valued Logic. pp. 150–155 (2010)
8. Goldberg, D., Lingle, R.: Alleles, loci, and the traveling salesman problem. In: Int'l Conference on Genetic Algorithms. pp. 154–159 (1985)
9. Gupta, P., Agrawal, A., Jha, N.K.: An algorithm for synthesis of reversible logic circuits. IEEE Trans. on CAD 25(11), 2317–2330 (2006)
10. Keijzer, M., Merelo, J.J., Romero, G., Schoenauer, M.: Evolving objects: a general purpose evolutionary computation library. In: Int'l Conference in Evolutionary Algorithms. pp. 231–244 (2001), the EO library is available at eodev.sourceforge.net
11. Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. 5, 183 (1961)
12. Merkle, R.C.: Reversible electronic logic using switches. Nanotechnology 4, 21–40 (1993)
13. Miller, D.M., Maslov, D., Dueck, G.W.: A transformation based algorithm for reversible logic synthesis. In: Design Automation Conf. pp. 318–323 (2003)
14. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge Univ. Press (2000)
15. Reed, I.: A class of multiple-error-correcting codes and their decoding scheme. IRE Trans. on Inf. Theory 3, 6–12 (1954)
16. Shende, V.V., Prasad, A.K., Markov, I.L., Hayes, J.P.: Synthesis of reversible logic circuits. IEEE Trans. on CAD 22(6), 710–722 (2003)
17. Soeken, M., Frehse, S., Wille, R., Drechsler, R.: RevKit: a toolkit for reversible circuit design. In: Workshop on Reversible Computation (2010), RevKit is available at www.revkit.org
18. Somenzi, F.: CUDD: CU Decision Diagram Package Release 2.3.1. University of Colorado at Boulder (2001), CUDD is available at vlsi.colorado.edu/~fabio/CUDD/
19. Song, N., Perkowski, M.: Minimization of exclusive sum of products expressions for multi-output multiple-valued input, incompletely specified functions. IEEE Trans. on CAD 15(4), 385–395 (1996)
20. Thomson, M.K., Glück, R.: Optimized reversible binary-coded decimal adders. J. of Systems Architecture 54, 697–706 (2008)
21. Toffoli, T.: Reversible computing. In: de Bakker, W., van Leeuwen, J. (eds.) Automata, Languages and Programming, p. 632. Springer (1980), technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.
22. Whitley, D., Starkweather, T., Fuquay, D.: Scheduling problems and traveling salesman: The genetic edge recombination operator. In: Int'l Conference on Genetic Algorithms. pp. 133–140 (1989)
23. Wille, R., Drechsler, R.: BDD-based synthesis of reversible logic for large functions. In: Design Automation Conf. pp. 270–275 (2009)
24. Zhirnov, V.V., Cavin, R.K., Hutchby, J.A., Bourianoff, G.I.: Limits to binary logic switch scaling – a gedanken model. Proc. of the IEEE 91(11), 1934–1939 (2003)