Designing Parity Preserving Reversible Circuits

Goutam Paul[1(✉)], Anupam Chattopadhyay[2], and Chander Chandak[3]

[1] Cryptology and Security Research Unit (CSRU),
R.C. Bose Centre for Cryptology and Security,
Indian Statistical Institute, Kolkata 700 108, India
goutam.paul@isical.ac.in
[2] School of Computer Engineering,
Nanyang Technological University (NTU), Singapore, Singapore
anupam@ntu.edu.sg
[3] Liv Artificial Intelligence Pvt. Ltd., Bengaluru, India
chandar.chandak@gmail.com

Abstract. With the emergence of reversible circuits as an energy-efficient alternative of classical circuits, ensuring fault tolerance in such circuits becomes a very important problem. Parity-preserving reversible logic design is one viable approach towards fault detection. Interestingly, most of the existing designs are ad hoc, based on some pre-defined parity preserving reversible gates as building blocks. In the current work, we propose a systematic approach towards parity preserving reversible circuit design. We prove a few theoretical results and present two algorithms, one from reversible specification to parity preserving reversible specification and another from irreversible specification to parity preserving reversible specification. We derive an upper-bound for the number of garbage bits for our algorithm and perform its complexity analysis. We also evaluate the effectiveness of our approach by extensive experimental results and compare with the state-of-the-art practices. To our knowledge, this is the first work towards systematic design of parity preserving reversible circuit and more research is needed in this area to make this approach more scalable.

Keywords: Fault tolerance · Parity · Quantum computing · Reversible circuits

1 Introduction and Motivation

It is known that erasure of a single bit of information dissipates heat equivalent to $K_B T \ln 2$ [3,12], where $K_B = 1.38 \times 10^{-23}$ J/K is Boltzmann constant and T is the room temperature in Kelvin. This heat dissipation is in conformity with the laws of thermodynamics applied to any irreversible process. Using reversible logic implementation of Boolean functions, it is theoretically possible to make heat dissipation and hence power loss negligible. Though classical logic is not reversible, it is possible to represent classical Boolean functions using reversible logic [2]. On the other hand, any quantum computation is based on unitary

The original version of this chapter was revised: Table 2 was corrected. An erratum to this chapter can be found at 10.1007/978-3-319-59936-6_20

I. Phillips and H. Rahaman (Eds.): RC 2017, LNCS 10301, pp. 77–89, 2017.
DOI: 10.1007/978-3-319-59936-6_6

evolution of quantum mechanical systems and is inherently reversible. However, with increasing demand on low power design, reversible logic finds application not only in quantum circuits, but also in designing conventional circuits for encoding/decoding etc [34].

Any physical device performing classical or quantum computation is subject to error due to noise in the environment or imperfections in the device. *Fault tolerant computing* can mitigate this. There are two broad approaches towards fault tolerance - one focuses on fault prevention and the other focuses on first fault detection and then fault correction. For fault detection, usage of redundant parity bits is one of the most popular approaches. For classical circuits, *bit flip* is the most common type of error. For quantum circuits, in addition to bit flip, there might be *phase flip* as well. In this short technical note, we focus on bit flip errors.

Most common method for detecting bit-flip errors in storage or transmission is by means of parity checking. Classically, most arithmetic and other processing functions do not preserve the parity. One has to use redundant circuitry to compute and check the parity. In general, making a reversible circuit fault-tolerant is much more difficult than classical circuit, since reversible logic allows no feedback or fan-out. The notion of parity-preserving arithmetic circuits goes back to [19]. Later, in [20], the concept of *parity preserving reversible circuits* was introduced. The idea is to design the reversible circuit in such a way that the parity between the input and the output bits are automatically conserved in absence of any error.

After [20], there has been a series of sporadic works in this area, such as designing adders [11], divider [4], multiplier [23], multiplexer [25], ALU [26] etc. The work [36] discusses the various steps required in the logic design of quantum circuits.

However, all of these designs are ad hoc, based on some pre-defined parity preserving reversible gates as building blocks. To the best of our knowledge, in this article, we for the first time propose a novel and systematic approach towards parity preserving reversible circuits design. We provide some related theoretical results and give two algorithms. The first algorithm converts a reversible specification to parity preserving reversible specification and the second one converts an irreversible specification directly to parity preserving reversible specification.

There are other approaches than parity preservation, for achieving fault-tolerance in reversible circuits, as described in [17] and in [18]. The advantage of parity-preserving circuit is that one need not do any extra operations in order to detect errors or faults; the fault detection becomes a by-product of the usual computation in the circuit. With this motivation, we focus on designing parity-preserving reversible circuits in this paper.

2 Reversible Logic Synthesis

An n-variable Boolean function is *reversible* if all its output patterns map uniquely to an input pattern and vice-versa. It can be expressed as an n-input, n-output bijection or alternatively, as a permutation over the truth value

set $\{0, 1, \ldots 2^{n-1}\}$. The problem of reversible logic synthesis is to map such a reversible Boolean function on a reversible logic gate library.

The gates are characterized by their implementation cost in quantum technologies, which is dubbed as Quantum Cost (QC) [14,16]. Reversible logic gates can also be represented as an unitary transformation, therefore serving as building blocks for quantum computers. Few prominent classical reversible logic gates are presented below.

- NOT gate: On input A, it produces $\overline{A}$ as output.
- CNOT gate: On input (A, B), it produces $(A, A \oplus B)$ as output.
- CCNOT gate: Also known as Toffoli gate. On input (A, B, C), it produces $(A, B, AB \oplus C)$ as output. This gate can be generalized with Tof_n gate, where first $n-1$ variables are used as control lines. NOT and CNOT gates are denoted as Tof_1 and Tof_2 respectively.
- Peres gate: A sequence of $Tof_3(a, b, c)$, $Tof_2(a, b)$ or its inverse is known as Peres gate.
- Controlled Swap gate, also known as Fredkin gate. On input (A, B, C), it produces $(A, \overline{A}.B + A.C, \overline{A}.C + A.B)$ as output. This gate can be generalized with $Fred_n$ gate $(n > 1)$, where first $n-2$ variables are used as control lines.

Multiple sets of reversible gates form an universal gate library for realizing classical Boolean functions such as, (i) NCT: NOT, CNOT, Toffoli. (ii) NCTSF: NOT, CNOT, Toffoli, SWAP, Fredkin. (iii) GT: Tof_n. (iv) GTGF: Tof_n and $Fred_n$. Of late, Clifford+T gate library is preferred for Quantum circuit construction due to the known constructions of Clifford group of operators and T gate for most promising error correcting codes, including surface code. In this work, we focus on the logical fault tolerance issue and focus on the classical reversible logic gates. Efficient Clifford+T realization of classical reversible logic gates form an important research problem.

Reversible logic synthesis begins from a given n-variable Boolean function, which can be irreversible. The first step is to convert it to a reversible Boolean function by adding distinguishing output bits, known as *garbage outputs*. When additional input Boolean variables are needed for constructing the output function, those are referred as *ancilla*. In this work, we focus on minimizing the number of garbage outputs. However, for a full generalized analysis, one should consider joint minimization of both the numbers of garbage outputs and the ancilla inputs.

Reversible logic synthesis methods can be broadly classified in four categories as following. A different and more detailed classification is presented in a recent survey of reversible logic synthesis methods [24].

- **Exact and Optimal methods:** These methods consider step-by-step exhaustive enumeration or formulating the logic synthesis as a SAT problem [7] or reachability problem [10]. Optimal implementations for all 4-variable Boolean functions [6] and for selected benchmarks up to 6-variable Boolean functions are known [9].

- **Transformation-based method** [13,35]**:** These methods use a weighted graph representation for performing the transformations, while [13] proceed row-wise in the Boolean truth-table.
- **Methods based on decision diagrams** [29,32]**:** In this approach, each node of the decision diagram is converted to an equivalent reversible circuit structure. These methods reported excellent scaling for large Boolean functions, low QC at the cost of high number of garbage bits.
- **ESOP-based methods:** For classical logic synthesis, the exclusive sum of products (ESOP) formulation is studied well for specific target technologies [15]. For reversible logic synthesis, the ESOP formulation [8] maps directly to the basic reversible logic gates and has led to significant research interest.

Among the above methods, methods based on Decision Diagrams and ESOP-based methods can synthesize an irreversible Boolean specification to reversible circuit by adding extra garbage lines. However, these methods do not guarantee the minimum garbage count. On the other hand, determination of minimum garbage count and their assignment is non-trivial, particularly for Boolean functions with large number of variables [33]. To the best of our knowledge, no automatic reversible logic synthesis tool supports automatic derivation of parity-preserving Boolean specification from an irreversible/reversible Boolean specification. Our flow proposed in the paper can be complemented with any reversible logic synthesis flows, which work on reversible Boolean specifications.

3 Theoretical Results

First we discuss how to convert a reversible Boolean specification (that does not necessarily consider parity preservation) into parity-preserving reversible specification. Before proceeding, we count the number of n-variable parity preserving reversible Boolean functions in Theorem 1.

Theorem 1. *Total number of n-variable parity preserving reversible Boolean functions is* $\left(2^{n-1}!\right)^2$.

Proof. In the truth table of an n-variable reversible Boolean function, there are 2^n input and output rows. Half of the 2^n input (or output) rows, i.e., total 2^{n-1} rows would have odd parity and the other half would have even parity. For the function to be parity-preserving, the odd-parity input rows must map to the odd-parity output rows. There are $2^{n-1}!$ such mappings. Corresponding to each of these, the even-parity input rows must map to the even-parity output rows and there are again $2^{n-1}!$ such mappings. Hence the result follows.

The method of constructing a parity-preserving reversible specification from any reversible specification is described in the proof of Theorem 2.

Theorem 2. *Given any n-variable reversible Boolean specification, it can be converted to a parity-preserving reversible Boolean specification with the introduction of at most one extra variable.*

Proof. If the function is already parity-preserving, we need not do anything. If not, then in the output column of the truth table, we can just put a 0 in the parity-matching rows and a 1 in the parity-mismatching rows. On the input side, the extra variable can be set to the constant 0. Hence the result follows.

3.1 Direct Method of Converting Irreversible Specification to Parity-Preserving Reversible Specification

Next, we discuss the case when we are given an irreversible Boolean specification. One simple approach can be a two-phase procedure: first, to use some standard approaches [33] for converting the irreversible specification to a reversible specification, and next, use the result of Theorem 2. However, the first phase in this approach may incur unnecessary extra garbage bits. To avoid this problem, we provide a direct method of converting a given irreversible specification to a parity-preserving reversible specification with theoretically bounded number of extra bits. The method is as follows.

Since the specification is irreversible, the output rows must contain duplicate bit-strings. Suppose there are n input variables and hence 2^n rows in the truth table. Suppose there are $k < 2^n$ distinct output bit-strings, with the counts $n_1, \ldots, n_k$, such that $\sum_{i=1}^{k} n_i = 2^n$. For each $i = 1, \ldots, k$, out of n_i rows with the same output bit-string, let $n_{i,p}$ be the number of rows where the input and the output parity is matching and so $n_i - n_{i,p}$ is the number of rows where the parity is not matching. To differentiate the matching rows we need at least $\lceil \log_2 n_{i,p} \rceil$ extra bits. Similarly, to differentiate the mismatching rows, we need at least $\lceil \log_2 (n - n_{i,p}) \rceil$ extra bits. Hence, for the rows corresponding to the bit-string category i, the number of extra bits needed is at most one more than the maximum of these two numbers. The one additional bit may be required to match the parity, in case the specification with the garbage bits is not already parity-preserving. Thus, the total number of extra bits needed is given by the maximum of the above quantity over all i's. Hence, with the above formulation, we have the following result.

Theorem 3. *The number of extra bits needed by the proposed algorithm to convert an irreversible specification to parity-preserving reversible specification is at most*

$$\max_{i=1}^{k} \{\max\{\lceil \log_2 n_{i,p} \rceil, \lceil \log_2 (n - n_{i,p}) \rceil\}\} + 1.$$

Note that the expression before 1 is the number of garbage lines needed to convert the irreversible specification to reversible specification which has been explained in the following subsection.

3.2 Algorithm and Its Complexity Analysis

We present the algorithm for converting an irreversible specification to parity-preserving reversible specification in Algorithm 1. Suppose $x_1, \ldots, x_k$ are k integers $\in \{0, \ldots, 2^n - 1\}$ corresponding to the distinct output bit-strings. Note that according to our notation, x_i appears n_i times. We will keep two arrays *match* and *mismatch* as follows. In the algorithm, $match[x_i]$ will contain $n_{i,m}$ and $mismatch[x_i]$ will contain $n - n_{i,m}$. The array $count[i]$, for $0, \ldots, 2^n - 1$, is filled from top to bottom order, corresponding to each output row as follows: $count[i]$ contains how many times the i-th output row has appeared so far starting from the top row in both the cases when the parity is preserved and when it is not preserved.

ALGORITHM 1. Irreversible to Parity Preserving Reversible Specification

Input: n, An integer array $out[0 \ldots 2^n - 1]$, containing the decimal equivalent of the output rows of an n-variable Boolean function.
Output: Parity preserving reversible specification.

```
1  max = 0;
2  for i = 0 to 2^n − 1 do
3  |  match[i] = 0, mismatch[i] = 0, count[i] = 0;
   end
4  for row ← 0 to 2^n − 1 do
5  |  if parity matches then
6  |  |  match[out[row]]++;
7  |  |  count[row] = match[out[row]];
8  |  |  if max < match[out[row]] then
9  |  |  |  max = match[out[row]];
   |  |  end
   |  end
10 |  else
11 |  |  mismatch[out[row]]++;
12 |  |  count[row] = mismatch[out[row]];
13 |  |  if max < mismatch[out[row]] then
14 |  |  |  max = mismatch[out[row]];
   |  |  end
   |  end
   end
15 g = log2 max + 1;
16 Add g columns to the Boolean output specification;
17 for row ← 0 to 2^n − 1 do
18 |  k = count[row];
19 |  Append binary value of k in the g − 1 bits;
20 |  Use the last bit, if necessary, to match parity;
   end
```

Now we present the complexity of our algorithm in Theorem 4.

Theorem 4. *For an n-input m-output Boolean specification, the running time of Algorithm 1 is* $O((n+m)2^n)$.

Proof. The maximum number of input or output rows in the Boolean specification is 2^n. Let there be $k < 2^n$ distinct output bit-strings with the counts $n_1, \ldots, n_k$, such that $\sum_{i=1}^{k} n_i = 2^n$. For each row we have to compute the number of 1's in the input and output bit-strings for computing the parity. The algorithmic complexity for this traversal is $O((n+m)2^n)$, which accounts for Steps 2 to 14. After this computation, we have one more iteration over the output rows through Step 17 to 20, the running time of which is dominated by $O((n+m)2^n)$. Hence the result follows.

4 Experimental Results

The proposed algorithm has been implemented and tested on several benchmark circuits, using C++ on an Intel(R) Core(TM) i5-3570 CPU (Quad-core) with 3.40 GHz clock and 6 MB cache, having Linux version 2.6.32-358.6.2.el6.x86_64 as the OS, and gcc version 4.4.7 as the compiler. First, we compared our automatically generated parity-preserving reversible circuits with manually created parity-preserving reversible circuits reported by others. Our comparison metric is the number of additional garbage lines required for preserving parity. Quantum cost and Gate Count for different specifications can vary considerably. In the paper we have given an example of rd53 circuit. For this circuit we can have a total of $(10! \times 10! \times 5! \times 5! = 1.8962193\text{e}+17)$ different possible parity preserved specifications. Even Table 2 in the paper with Full Adder Boolean specification has $(3! \times 3! = 36)$ different possible reversible specifications.

4.1 Comparison with State-of-the-Art

We apply the proposed algorithm on Half Adder and Full Adder as two test cases. The transformation of irreversible Boolean specification to a reversible one is depicted in Tables 1 and 2 respectively, with the required number of constant input and garbage lines. The ancilla inputs and garbage outputs are referred as A_i and G_i respectively. The reversible specification thus obtained can be used to implement the reversible circuit using the well-known reversible logic synthesis methods for garbage-free synthesis [13].

We do not compare the gate count and quantum cost incurred in realizing the circuit as the proposed algorithm does not aim to optimize those parameters. Our aim was to minimize the number of garbage lines. For the parity preserved half adder circuit obtained from the proposed algorithm, the gate count and quantum cost required for the realization of the circuit are 8 and 28 respectively. The approach followed for the construction of the circuit is similar to the transformation based synthesis as proposed in [13]. The circuit is shown in Fig. 1.

Table 1. Half Adder Boolean specification

Irreversible specification		Reversible specification					
Input	Output	Input	A_1	A_2	Output	G_1	G_2
00	00	00	0	0	00	0	0
01	10	01	0	0	10	0	0
10	10	10	0	0	10	1	1
11	01	11	0	0	01	0	1

Table 2. Full Adder Boolean specification

Irreversible specification		Reversible specification						
Input	Output	Input	A_1	A_2	Output	G_1	G_2	G_3
000	00	000	0	0	00	0	0	0
001	10	001	0	0	10	0	0	0
010	10	010	0	0	10	0	1	1
011	01	011	0	0	01	0	0	1
100	10	100	0	0	10	1	0	1
101	01	101	0	0	01	0	1	0
110	01	110	0	0	01	1	0	0
111	11	111	0	0	11	0	0	1

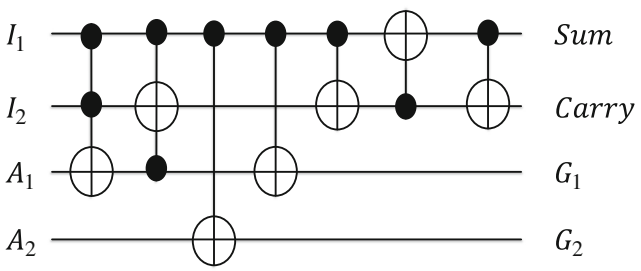

Fig. 1. Realization for the parity preserved half adder circuit as per Table 1

In terms of the ancilla and garbage count, we obtain exactly the same number for both the Half Adder and Full Adder circuits as obtained manually in [1,27].

It is also worthwhile to compare with the online testability approaches proposed in [17,18]. There, an additional parity line and modulo-redundancy is added corresponding to every reversible gate after the circuit is synthesized. Naturally, this leads to a significant design overhead, which can be up to 300% in terms of gate count [18]. Even with such an overhead, there are fault scenarios that cannot be covered. In contrast, our proposition only requires $2N$ additional CNOT gates, where N is the number of inputs in the parity-preserved reversible circuit. The CNOT gates are targeted towards one additional parity line, similar to the *Preamble* and *Postamble* blocks suggested in [18].

A limitation of our approach is that it assumes a rather simplistic bit-flip model arising from classical reversible logic circuits. In the context of, say, Quantum technologies, the fault models are different [22] and requires a deeper analysis. For example, it is indeed possible to interpret a Single Missing Gate Fault (SMGF) or Single Missing Control Fault (SMCF) as a bit-flip, though, it is not guaranteed that a parity-preserving reversible circuit can lead to a 100% detection of all possible missing faults. For that, the propagation of an individual bit-flip and the masking effects of the subsequent gates need to be taken into account. Moreover, the correlation between parity violation and the two kinds of missing faults is circuit specific. Clearly, it is an interesting open problem to identify the minimum performance overhead to guarantee complete fault coverage with a solution lying between gate-wise redundancy advocated earlier [18] and circuit-level parity-preservation proposed here.

4.2 Tests for Boolean Functions with Large Variable Count

We also tried the algorithm for several Boolean functions with large number of variables, for which obtaining a parity-preserving Boolean specification manually would be hard.

As an example, our algorithm converts the irreversible specification *rd53* [13] into reversible one as enlisted in Table 3. A summary of all the functions we tried is presented in Table 4. In this table, the *tar* functions are from Tarannikov's paper [31]. From [31, Eq. 2], we use the parameter c as 001 to construct an 8-variable, 2-resilient function then we get tar82_2_001.pla. Similarly tar93_110.pla and tar93_101.pla are 9 variable 3-resilient functions with the c vector as 110 and 101 respectively. The functions like $rdNK$ is presented in several benchmarks on reversible logic synthesis [16]. The input weight function $rdNK$ has N inputs and $K = \lfloor logN \rfloor + 1$ outputs. Its output is the binary encoding of the number of ones in its input. The other functions are obtained from RevKit benchmark [28].

Table 3. Reversible Boolean specification for rd53 function

Input	A_1	A_2	A_3	Output	G_1	G_2	G_3	G_4	G_5
00000	0	0	0	000	0	0	0	0	0
00001	0	0	0	001	0	0	0	0	0
00010	0	0	0	001	0	0	0	1	1
00011	0	0	0	010	0	0	0	0	1
00100	0	0	0	001	0	0	1	0	1
00101	0	0	0	010	0	0	0	1	0
00110	0	0	0	010	0	0	1	0	0
00111	0	0	0	011	0	0	0	0	1
01000	0	0	0	001	0	0	1	1	0
01001	0	0	0	010	0	0	1	1	1
01010	0	0	0	010	0	1	0	0	0
01011	0	0	0	011	0	0	0	1	0
01100	0	0	0	010	0	1	0	1	1
01101	0	0	0	011	0	0	1	0	0
01110	0	0	0	011	0	0	1	1	1
01111	0	0	0	100	0	0	0	0	1
10000	0	0	0	001	0	1	0	0	1
10001	0	0	0	010	0	1	1	0	1
10010	0	0	0	010	0	1	1	1	0
10011	0	0	0	011	0	1	0	0	0
10100	0	0	0	010	1	0	0	0	0
10101	0	0	0	011	0	1	0	1	1
10110	0	0	0	011	0	1	1	0	1
10111	0	0	0	100	0	0	0	1	0
11000	0	0	0	010	1	0	0	1	1
11001	0	0	0	011	0	1	1	1	0
11010	0	0	0	011	1	0	0	0	0
11011	0	0	0	100	0	0	1	0	0
11100	0	0	0	011	1	0	0	1	1
11101	0	0	0	100	0	0	1	1	1
11110	0	0	0	100	0	1	0	0	0
11111	0	0	0	101	0	0	0	0	1

Table 4. Summary of results for exemplary Boolean functions with large no. of variables

Function	Input count	Output count	Garbage count	Ancilla count	Runtime (ms)
tar82_2_001.pla	8	1	8	1	0.66
tar93_110.pla	9	1	8	0	1.89
tar93_101.pla	9	1	8	0	1.63
rd53	5	3	5	3	0.18
rd73	7	3	7	3	0.35
rd84	8	4	8	4	0.64
rd20_5	20	5	19	4	34.70
rd10_4	10	4	9	3	23.17
0410184_85.pla	14	14	1	1	14.17
cycle10_2_61.pla	12	12	1	1	3.39
ham15_30.pla	15	15	1	1	30.15
ham7_29.pla	7	7	1	1	0.20
ham8_64.pla	8	8	1	1	0.31
life_175.pla	9	1	9	1	0.45
squar5.pla	5	8	1	4	6.77
urf4_89.pla	11	11	1	1	1.76
urf6.pla	15	15	1	1	29.21
plus63mod8192.pla	13	13	1	1	6.76

5 Conclusion and Future Work

We propose the first systematic algorithm to convert any irreversible specification into a parity-preserving reversible specification. In existing works such as in [11,27], a new specific gate is introduced to realize one particular parity-preserving circuit. However, these gates may not be useful to realize other circuits. Our method is fully automated and general and can work on any given circuit. The relevant code for the Algorithm 1 has been shared at [5].

In the current work, we have focused on bit-flip error only. However, the fault coverage for different logical fault models [22] arising in the context of Quantum circuit implementation requires further work, which we plan to undertake. Another interesting future work could be to tackle the complexity of the input representation.

One limitation of our work is that it uses truth-table specification and hence is not scalable for large variables. An interesting future work could be exploring the possibility of direct synthesis of parity-preserving circuits based on more compact representations, such as BDDs or other hierarchical reversible logic synthesis [30]. Such an approach may be more efficient for functions of larger number of variables and hence more scalable. Moreover, as pointed out in a recent work [21], even if a reversible circuit is parity preserving, it has to be checked against a particular fault model. As part of our future work, we also

plan to inject faults at different gates and estimate the fault coverage of our circuits against different fault models.

References

1. Azad Khan, M.H.: Design of full-adder with reversible gates. In: International Conference on Computer and Information Technology, pp. 515–519 (2002)
2. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**, 525–532 (1973)
3. Bérut, A., Arakelyan, A., Petrosyan, A., Ciliberto, S., Dillenschneider, R., Lutz, E.: Experimental verification of Landauer's principle linking information and thermodynamics. Nature **483**, 187–189 (2012)
4. Dastan, F., Haghparast, M.: A novel nanometric fault tolerant reversible divider. Int. J. Phys. Sci. **6**(24), 5671–5681 (2011)
5. https://github.com/cchandak/parity_preserving_rev_ckt
6. Golubitsky, O., Falconer, S.M., Maslov, D.: Synthesis of the optimal 4-bit reversible circuits. In: Proceedings of DAC, pp. 653–656 (2010)
7. Grosse, D., Wille, R., Dueck, G.W., Drechsler, R.: Exact multiple-control toffoli network synthesis with SAT techniques. IEEE TCAD **28**(5), 703–715 (2009)
8. Gupta, P., Agrawal, A., Jha, N.K.: An algorithm for synthesis of reversible logic circuits. IEEE TCAD **25**(11), 2317–2330 (2006)
9. Grosse, D., Wille, R., Dueck, G.W., Drechsler, R.: Exact multiple-control tooli network synthesis With SAT techniques. IEEE TCAD **28**(5), 703–715 (2009). doi:10.1109/TCAD.2009.2017215
10. Hung, W.N.N., Xiaoyu, S., Guowu, Y., Jin, Y., Perkowski, M.: Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis. IEEE TCAD **25**(9), 1652–1663 (2006)
11. Islam, M.S., Rahman, M.M., Begum, Z., Hafiz, A., Al Mahmud, A.: Synthesis of fault tolerant reversible logic circuits. In: Proceedings of IEEE Circuits and Systems International Conference on Testing and Diagnosis, pp. 1–4 (2009)
12. Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. **5**, 183–191 (1961)
13. Miller, D.M., Maslov, D., Dueck, G.W.: A transformation based algorithm for reversible logic synthesis. In: Proceedings of DAC, pp. 318–323 (2003)
14. Miller, D.M., Wille, R., Sasanian, Z.: Elementary quantum gate realizations for multiple-control toffolli gates. In: Proceedings of International Symposium on Multiple-Valued Logic, pp. 288–293 (2011)
15. Mishchenko, A., Perkowski, M., Fast heuristic minimization of exclusive-sums-of-products. In: Proceedings of the Reed-Muller Workshop, pp. 242–250 (2001)
16. Maslov, D.: Reversible Benchmarks. http://webhome.cs.uvic.ca/~dmaslov, Accessed Jun 2013
17. Nayeem, N.M., Rice, J.E.: Online testable approaches in reversible logic. J. Electron. Test. **29**(6), 763–778 (2013)
18. Nashiry, M.A., Bhaskar, G.G., Rice, J.E.: Online testing for three fault models in reversible circuits. In: Proceedings of ISMVL, pp. 8–13 (2011). doi:10.1109/ISMVL.2015.36
19. Parhami, B.: Parity-preserving transformations in computer arithmetic. In: Proceeding of SPIE, vol. 4791, pp. 403–411 (2002)

20. Parhami, B.: Fault-tolerant reversible circuits. In: Proceeding of 40th Asilomar Conference Signals, Systems, and Computers, Pacific Grove, CA, pp. 1726–1729, October 2006
21. Przigoda, N., Dueck, G.W., Wille, R., Drechsler, R.: Fault detection in parity preserving reversible circuits. In: Proceeding of IEEE 46th International Symposium on Multiple-Valued Logic (ISMVL), Sapporo, Japan, pp. 44–49, 18–20 May 2016
22. Polian, I., Fiehn, T., Becker, B., Hayes, J.P.: A family of logical fault models for reversible circuits. In: Proceedings of Asian Test Symposium, pp. 422–427 (2011)
23. Qi, X., Chen, F., Zuo, K., Guo, L., Luo, Y., Hu, M.: Design of fast fault tolerant reversible signed multiplier. Int. J. Phys. Sci. **7**(17), 2506–2514 (2012)
24. Saeedi, M., Markov, I.L.: Synthesis and optimization of reversible circuits - a survey. In: CoRR abs/1110.2574, http://arxiv.org/abs/1110.2574 (2011)
25. Saligram, R., Hegde, S.S., Kulkarni, S.A., Bhagyalakshmi, H.R., Venkatesha, M.K.: Design of fault tolerant reversible multiplexer based multi-boolean function generator using parity preserving gates. Int. J. Comput. Appl. **66**(19), 20–24 (2013)
26. Saligram, R., Hegde, S.S., Kulkarni, S.A., Bhagyalakshmi, H.R., Venkatesha, M.K.: Design of parity preserving logic based fault tolerant reversible arithmetic logic unit. In: CoRR abs/1307.3690, http://arxiv.org/abs/1307.3690 (2013)
27. Syal, N., Sinha, H.P., Sheenu: Comparison of different type parity preserving reversible gates and simple reversible gates. In: International Journal of Research and Innovation in Computer Engineering, vol. 1, issue 1 (2011)
28. Soeken, M., Frehse, S., Wille, R., Drechsler, R.: RevKit: a toolkit for reversible circuit design. In: Proceedings of Workshop on Reversible Computation, pp. 64–76 (2011)
29. Soeken, M., Wille, R., Hilken, C., Przigoda, N., Drechsler, R.: Synthesis of reversible circuits with minimal lines for large functions. In: Proceedings of ASP-DAC, pp. 85–92 (2012). doi:10.1109/ASPDAC.2012.6165069
30. Soeken, M., Chattopadhyay, A.: Unlocking efficiency and scalability of reversible logic synthesis using conventional logic synthesis. In: Proceedings of the 53rd Annual Design Automation Conference (DAC), Article no. 149, Austin, Texas, 05–09 June 2016
31. Tarannikov, Y.: New constructions of resilient boolean functions with maximal nonlinearity. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 66–77. Springer, Heidelberg (2002). doi:10.1007/3-540-45473-X_6
32. Wille, R., Drechsler, R.: BDD-based synthesis of reversible logic for large functions. In: Proceedings of DAC, pp. 270–275 (2009)
33. Wille, R., Keszöcze, O., Drechsler, R.: Determining the minimal number of lines for large reversible circuits. In: Proceedings of DATE, pp. 1–4 (2011)
34. Wille, R., Drechsler, R., Osewold, C., Garcia-Ortiz, A.: Automatic design of low-power encoders using reversible circuit synthesis. In: Proceedings of DATE, pp. 1036–1041 (2012). doi:10.1109/DATE.2012.6176648
35. Zheng, Y., Huang, C.: A novel toffoli network synthesis algorithm for reversible logic. In: Proceedings of ASP-DAC, pp. 739–744 (2009)
36. Wille, R., Chattopadhyay, A., Drechsler, R.: From reversible logic to quantum circuits: logic design for an emerging technology. In: Proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), pp. 268–274 (2016)