Online Fault Detection in Reversible Logic

N. M. Nayeem
Dept. of Math & Computer Science
University of Lethbridge
Lethbridge, Canada
noor.nayeem@uleth.ca

J. E. Rice
Dept. of Math & Computer Science
University of Lethbridge
Lethbridge, Canada
j.rice@uleth.ca

Abstract

A new approach for online fault detection in Boolean reversible circuits is described. Previous work had described this approach for circuits generated by the basic ESOP-based logic synthesis, and in this work we extend the approach for any type of Toffoli networks. An online testable circuit is created by modifying an existing cascade of Toffoli gates in a simple process that involves changing the existing Toffoli gates as well as the addition of one line and $2p$ gates, where p is the number of lines in the original circuit.

Keywords

reversible logic; online testability; Toffoli gates

I. Introduction

Reversible computation has been shown by Bennett [1] and Landauer [2] to be a necessary component in designing lower power circuits. Reversible computing recovers energy by conserving information when performing logic, storage, and communication operations using reversible transformations [3], and industry groups are acknowledging the necessity of moving to such a paradigm if we wish to continue advancements in this field [4]. Research on reversible computing is useful for various technologies such as quantum computing [5], low power CMOS design [6], optical computing [7], nanotechnology [8], and bioinformatics.

A key component in circuit design is that of testing. Testing can be performed offline, with the circuit no longer in operation, or online while normal operations take place. In this work we concentrate on the latter approach. We also focus on a particular type of reversible circuit, one that is built from Toffoli gates. Toffoli networks can be generated by a number of algorithms as described in Section II, following which our simple approach as described in Section IV can be followed to create an online testable reversible circuit. We note that at this point our approach is restricted to cascades of Boolean Toffoli gates; while multiple-valued versions of reversible gates exist, we are not currently considering them in our work.

II. Background

A. Reversible Circuits

In general, a function is reversible if there is a one-to-one and on-to mapping from the inputs to the outputs (and vice versa) of the function. For example, the function $f(X) = x_1x_2$ is not reversible, as there are two inputs and only one output. In any reversible function it should be possible to reconstruct the input values from the given output values. The traditional inverter, or NOT gate, is reversible. In [9], Shende *et al.* provides the following definitions: a gate is reversible if the (Boolean) function it computes is bijective, and a well-formed reversible logic circuit is an acylic combinational logic circuit in which all gates are reversible, and are interconnected without fanout.

Some popular reversible gates are the CNOT (sometimes referred to as Controlled-NOT) gate, which has the behaviour $(x, y) \to (x, x \oplus y)$; the Fredkin gate, which has the behaviour $(x, y, z) \to (x, z, y)$ *iff* $x = 1$; and the Toffoli gate, which has the behaviour $(x, y, z) \to (x, y, xy \oplus z)$. We focus further on Toffoli gates in the next section, as they are integral to this work.

B. Toffoli Gates

An n-bit generalized Toffoli gate is a reversible logic gate that has n inputs and n outputs and is described as $(x_1, x_2, \ldots, x_n) \to (x_1, x_2, \ldots, (x_1x_2 \cdots x_{n-1}) \oplus x_n)$. A NOT gate is a special case of a Toffoli gate where $n = 1$ and the CNOT gate is a special case in which $n = 2$. In general the first $n-1$ bits are known as controls, and the final (n^{th}) bit is the target. The CNOT gate and an n-bit Toffoli gate are shown in Figure 1(a) and Figure 1(b). A negative-control Toffoli gate has one or more negative controls; in that case, the target bit is toggled if all positive controls have the value 1 and the negative controls have the value 0. A 3-bit Toffoli gate with a single negative control in its first input is shown in Figure 1(c).

In this work we also make use of an ETG, or extended Toffoli gate. An ETG is a multi-target Toffoli gate proposed in [10]. For our proposed design, we need an (n+1)-bit ETG with two target outputs (o_n and o_{n+1}) as shown in Figure 1(d). This gate has the input vector [k_1, k_2, ..., k_n, k_{n+1}] and the output vector [o_1, o_2, ..., o_n, o_{n+1}], where o_j = k_j (for j = 1, 2, ..., n-1), o_n = $k_1k_2\cdot\cdot k_{n-1}\oplus k_n$, and o_{n+1} = $k_1k_2\cdot\cdot k_{n-1}\oplus k_{n+1}$. The first $n-1$ bits are controls and the last two bits are targets.

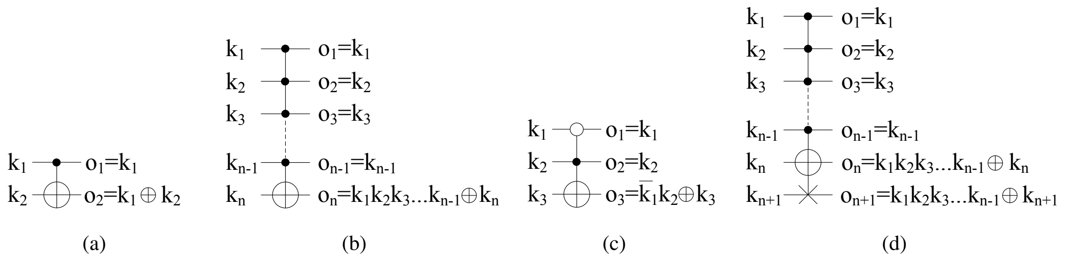

Figure 1. (a) A CNOT gate, (b) an n-bit Toffoli gate, (c) a 3-bit negative-control Toffoli gate, and (d) an (n+1)-bit ETG.

C. Reversible Logic Synthesis

There are a number of different techniques for synthesis of reversible logic circuits, including a transformation technique [11], the use of positive polarity Reed-Muller expressions (PPRM) [12], the use of exclusive-or sum-of-products (ESOP) [13], [14], and some decision diagram-based techniques [15]. For this work we require that the reversible circuit be modeled as a cascade of Toffoli gates, so we restrict ourselves in this section to describing two of the most popular techniques which result in this type of circuit.

One approach is to begin with an ESOP representation of the benchmark to be implemented. An ESOP is a sum-of-products format in which the traditional OR operator combining the products is replaced with an EXCLUSIVE-OR (EXOR) operator. Tools such as EXORCISM [16] can be used to generate an ESOP list for most benchmark formats. After a list of ESOP cubes is generated it is a simple process to replace each cube with a Toffoli gate. This approach was introduced in [13] and has been since improved upon in a number of works including [14], [17], [18].

Another approach was introduced in [11] as the transformation-based technique. This technique involves examination of a truth table of a reversible function, and identifying transformations that match the behaviour of reversible gates that can be applied to the truth table to bring the input patterns into agreement with the output patterns. The application of these transformations then can be translated into a cascade of gates; if the transformations are restricted to only those matching Toffoli gates, then the resulting cascade will similarly consist only of Toffoli gates.

D. Measures

Gate count is the simplest way to compare different reversible circuits. According to [19], gate count cannot provide an accurate measure if the gates used in circuits have different functionalities and/or sizes. The quantum cost is a good cost metric in this regard since it refers to the number of elementary (quantum) gates required to implement the circuit. The quantum cost of a reversible circuit can be calculated by adding the quantum costs of its gates. To calculate the quantum cost, we use the costs of gates given in [20]. For more information, please see [21].

In reversible circuits, some unwanted outputs are generated to maintain the reversibility property. These outputs are called garbage outputs. A good design has small number of garbage outputs. Since the previously reported approaches and our proposed approach use different types of gates, we compare the designs in terms of quantum cost and garbage outputs.

III. Related Work

A number of different approaches to generating online testable reversible circuits exist in the literature including designs in [22], [23], [24], [25], [26]. We briefly describe each of these and discuss the limitations.

A design methodology for constructing online testable reversible circuits based on three reversible gates R1, R2 and R was proposed in [22] and [23]. Each gate is a 4×4 gate, with the first gate R1 which is shown in Figure 2(a), used to realize NAND, OR, EXOR, and XNOR operations by setting different values on inputs. R1 has the parity output at q. R2 which is shown in Figure 2(b), passes the inputs through to the outputs, with a parity output again being computed at s, and in order to construct a testable block (TB), the gates R1 and R2 are cascaded by connecting the first three outputs of R1 to the first three inputs of R2 creating a TB with two parity outputs that can be compared to determine whether a fault has occurred or not. Figure 2(c) shows the block diagram of a TB. The TBs are then used to realize the reversible circuit. Since each TB generates two parity outputs, a two-pair two-rail checker circuit is also required to test the parities of two TBs. This checker circuit is built using eight R gates, and a block diagram is shown in Figure 2(d). If a circuit contains more than two TBs, a cascade of checker circuits is required.

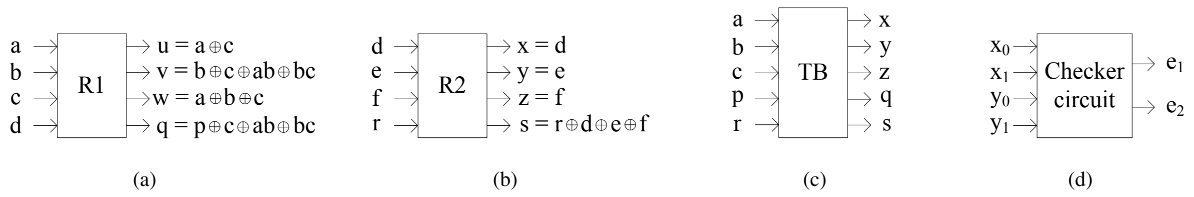

Figure 2. (a) R1 gate, (b) R2 gate, (c) a testable block (TB), and (d) a two-pair two-rail checker circuit.

An online testable circuit for the function $f = ab + \overline{c}$ is implemented in Figure 3. The first TB realizes $\overline{ab}$ which is then fed along with input c to the second TB, producing the output $f = \overline{\overline{ab}c} = ab + \overline{c}$. The two parity outputs of each TB are connected to the checker circuit. By examining the outputs of checker circuit, the circuit detects a fault.

We have found that this method cannot detect all single faults. If a fault occurs between two TBs, the circuit cannot detect it. Note that TBs generate parities which are tested by checker circuits; thus it can only detect a fault in TBs. Occurrence of any fault outside the TBs is left undetected. For example, in the circuit given in Figure 3, if a fault occurs at the first input of the second TB, the circuit is unable to detect it. As a result, this approach can detect some single faults, more specifically faults that occur in TBs.

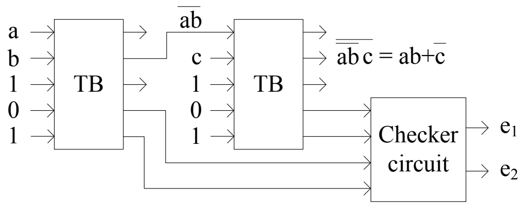

Figure 3. Online testable circuit for $f = ab + \overline{c}$, according to the design in [22] and [23].

An extension of the previously described method was proposed in [24], [25], which constructs an online testable circuit from a given circuit. This technique substitutes each gate in the given circuit by a testable block which generates two parities. To test the parities, an improved checker circuit was also proposed. However, the design flaw that we described in the previous approach also exists in this design. Thus it also fails to detect a fault between the connections of two blocks.

Another approach proposed in [26] uses a set of 4×4 dual rail reversible gates for online fault detection. Each dual rail gate has two pairs of inputs, and two inputs of each pair are given in dual rail form, *i.e.* two inputs that are the complement of each other (either 01 or 10). If the outputs appear in dual rail form, then there is no error. However, a non-dual rail form (either 11 or 00) represents a single fault. These dual rail gates are cascaded to generate the testable circuit. A fault in the circuit propagates to the end of the circuit. Thus the fault is detected by checking the outputs of the circuit. As a result, no checker circuit is required to test the intermediate gates.

IV. Our Approach

Our approach builds on the idea initially presented in [27]. We consider a reversible circuit consisting only of Toffoli gates; in order to make such a circuit online testable, we add some CNOT gates and a parity line L which is initialized with a 0. The procedure is given below.

Assume that the given circuit has p inputs and p outputs; thus the circuit has p lines. The procedure inserts CNOT gates from all lines to L at the beginning of the circuit. It then replaces every n-bit Toffoli gate by an (n+1)-bit ETG. The connections of the first n bits of the ETG are kept the same as that of n-bit Toffoli gate. The last bit of the ETG is connected to L. The NOT gates found on the lines are retained. If the number of NOT gates in the circuit is an odd number, an extra NOT gate is added at the end of line L; otherwise, no extra NOT gate is added. Finally, CNOT gates are added at the end from all lines to L.

This approach requires a total of $2p$ extra CNOT gates and at most one extra NOT gate to construct an online testable circuit. In the testable circuit, if a single fault occurs in any line (including L), the value of L changes from 0 to 1. If no fault occurs, L remains 0. Thus by checking a single bit L at the end of the circuit, the fault is detected. Note that this procedure also works for any circuit consisting of inverted-control Toffoli gates.

The conversion procedure is described by an example. Consider a Toffoli circuit given in Figure 4(a) which has five lines (I_1, I_2, ..., I_5) and four Toffoli gates (t_1, t_2, t_3, and t_4). For constructing an online testable circuit, we add a parity line L as well as replace t_1, t_2, t_3, and t_4 by ETGs e_1, e_2, e_3, and e_4. We also insert five CNOT gates $c_1 - c_5$ at the beginning and five CNOT gates $c_6 - c_{10}$ at the end of circuit. The resultant online testable circuit is shown in Figure 4(b).

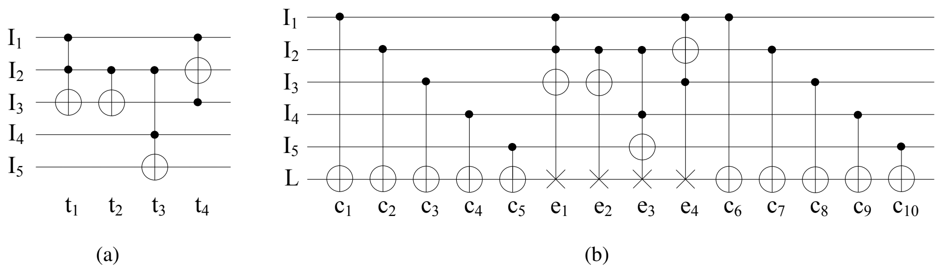

Figure 4. (a) A Toffoli circuit and (b) an online testable circuit.

V. Analysis

A fault in the target of an ETG certainly affects the gate as it changes the output of the target. However, a fault in control affects an ETG, only if it causes the target to have the faulty value. In other words, faults in control bits affect the gate if all fault-free control bits have the value 1, and all faulty control bits are either 1 or 0. A single fault in the control of an ETG can propagate to target lines if the fault affects the gate. Thus a fault can cause multiple faults. Note that a single fault on the parity line L cannot propagate to other lines since controls of the CNOTs and ETGs are not connected to L in the proposed design. Our proposed approach can detect any single-bit fault even though it causes several faults. The following example shows how a single fault on a line can propagate to several lines.

Consider a circuit consisting of two ETGs as shown in Figure 5. The lines I_1, I_2, I_3, and L are initialized by 0, 0, 1, and 0, respectively. Assume a fault occurs on I_1 just before the first ETG; thus the value of I_1 changes to 1. For the faulty lines, values are given in the form [fault-free value/ faulty value]. The fault on I_1 affects the first ETG and propagates to I_2 and L by the targets of this gate. As a result, two extra lines I_2 and L become faulty. The second ETG causes I_3 to have the faulty value and fixes the value of L. Therefore, after the second ETG, lines I_1, I_2 and I_3 are faulty.

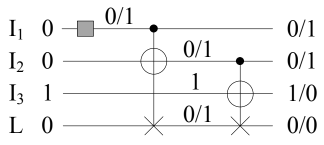

Figure 5. Fault propagation in multiple lines.

Lemma 1 proves that both targets of an ETG calculate the same function regardless of the occurrence of the fault. Lemma 2 proves that the proposed technique detects any single fault even if it propagates to other lines, causing multiple faults.

Lemma 1: Consider an (n+1)-bit ETG which maps the input vector [k_1, k_2, ..., k_{n-1}, k_n, k_{n+1}] to the output vector [o_1, o_2, ..., o_{n-1}, o_n, o_{n+1}], where $o_j = k_j$ (for j =1, 2, ..., n-1), $o_n = f \oplus k_n$, $o_{n+1} = f \oplus k_{n+1}$, and $f = k_1k_2 \cdots k_{n-1}$.

a) If faults occur in controls of the ETG and affect the gate, then both o_n and o_{n+1} compute $\overline{f}$; otherwise both outputs compute f.
b) If faults occur in targets of the ETG, then both o_n and o_{n+1} compute f.
c) If faults occur in targets and controls which affect the ETG, then both o_n and o_{n+1} compute $\overline{f}$; otherwise both outputs compute f.

◁

Proof:

a) Consider faults in k_i, k_j, ..., k_l such that $\{i, j, \ldots, l\} \subseteq \{1, 2, \ldots, n\text{-}1\}$. These faults affect the calculation of function f if the following two conditions hold.
   i) k_m = 1, $\forall$ $m \in \{1, 2, \ldots, n\text{-}1\}$ - $\{i, j, \ldots, l\}$, and
   ii) $k_i = k_j = \ldots = k_l$ = 0 or $k_i = k_j = \ldots = k_l$ = 1.

   If both conditions are true, then the fault has impact on o_n and o_{n+1} since both outputs compute f. Thus $o_n = \overline{f} \oplus k_n$ and $o_{n+1} = \overline{f} \oplus k_{n+1}$. If at least one of the conditions is false, then the fault does not affect the calculation of f. Thus according to the definition, $o_n = f \oplus k_n$ and $o_{n+1} = f \oplus k_{n+1}$.
   Therefore, if the faults affect, then o_n and o_{n+1} compute $\overline{f}$; otherwise these outputs compute f.

b) We consider three cases, depending on whether one of the targets is faulty or both targets are faulty.
   Case 1. Assume a fault occurs in k_n. This fault does not propagate to o_{n+1} since o_{n+1} is independent of k_n; thus o_{n+1} = $f \oplus k_{n+1}$. However, due to the fault, $o_n = f \oplus \overline{k}_n$.
   Case 2. Assume a fault occurs in k_{n+1}. The proof is similar to Case 1. We get $o_n = f \oplus k_n$ and $o_{n+1} = f \oplus \overline{k}_{n+1}$.
   Case 3. Consider that faults occur in both k_n and k_{n+1}. Due to these faults, $o_n = f \oplus \overline{k}_n$ and $o_{n+1} = f \oplus \overline{k}_{n+1}$.
   Hence, for any of the cases, both o_n and o_{n+1} compute f.

c) We consider two cases, depending on whether faults in controls affect the gate or not.
   Case 1. First consider that faults in controls affect the gate. From (a), we can write $o_n = \overline{f} \oplus k_n$ and $o_{n+1} = \overline{f} \oplus k_{n+1}$. Assume that faults also occur in both targets. According to (b), we can rewrite the outputs as follows. $o_n = \overline{f} \oplus \overline{k}_n$ and $o_{n+1} = \overline{f} \oplus \overline{k}_{n+1}$ Thus both outputs compute $\overline{f}$. Similarly, we can reach the same conclusion if a fault occurs in any of the targets.
   Case 2. Now consider that faults in controls have no effect on the gate. This proof is similar to (b). ◀

Lemma 2: If any single fault occurs on any line, the circuit is able to detect it. ◁

Proof: Consider an online testable circuit generated by the proposed technique which has N ETGs, p lines (I_1, I_2, ..., I_p), and a parity line L. Let $G = \{g_1, g_2, ..., g_N\}$ be the set of ETGs used in the circuit. Let the initial values of lines I_1, I_2, ..., I_p be i_1, i_2, ..., i_p. The line L is initialized with a 0. Given an (n+1)-bit ETG $g \in G$ and the input vector [k_1, k_2, k_3, ..., k_{n-1}, k_n, k_{n+1}], the output vector is [k_1, k_2, k_3, ..., k_{n-1}, $fg \oplus k_n$, $fg \oplus k_{n+1}$], where $fg = k_1 k_2 \cdots k_{n-1}$, and n can be at most p. In order to prove this lemma, consider the following two cases.

Case 1. Assume that a single fault occurs on a line I_d (for d = 1, 2, ..., p) and propagates to multiple lines.
Let $W = \{w_1, w_2, ..., w_q\} \subseteq$ G be the set of gates that are not affected by the faults.
Let $X = \{x_1, x_2, ..., x_r\} \subseteq$ G be the set of gates such that faults occur only on controls and affect the gates.
Let $Y = \{y_1, y_2, ..., y_s\} \subseteq$ G be the set of gates with faults only on targets.
Let $Z = \{z_1, z_2, ..., z_t\} \subseteq$ G be the set of gates such that faults occur on targets and controls which affect the gates.
We have $G = W \cup X \cup Y \cup Z$, and W, X, Y or Z can be empty. According to the definition of the ETG, two targets of a gate $g \in G$ compute the same function fg. Similarly, a gate $w_j \in W$ computes fw_j (for j = 1, 2, ..., q). A gate $x_l \in X$ computes $\overline{fx_l}$ (for l = 1, 2, ..., r) according to Lemma 1(a). A gate $y_u \in Y$ computes fy_u (for u = 1, 2, ..., s) according to Lemma 1(b). A gate $z_v \in Z$ computes $\overline{fz_v}$ (for v = 1, 2, ..., t) according to Lemma 1(c).
At the beginning of the circuit, all I_m (for m = 1, 2, ..., p) lines are EXORed to parity line L. Thus the value of L, just before the first gate in G, is $i_1 \oplus i_2 \oplus \ldots \oplus i_d \oplus \ldots \oplus i_p$. After the last gate in G, the value of L becomes $i_1 \oplus i_2 \oplus \ldots \oplus i_d \oplus \ldots \oplus i_p \oplus fw_1 \oplus fw_2 \oplus \ldots \oplus fw_q \oplus \overline{fx_1} \oplus \overline{fx_2} \oplus \ldots \oplus \overline{fx_r} \oplus fy_1 \oplus fy_2 \oplus \ldots \oplus fy_s \oplus \overline{fz_1} \oplus \overline{fz_2} \oplus \ldots \oplus \overline{fz_t}$.
At the end, when all I_m (for m = 1, 2, ..., p) lines are EXORed to L again, the faulty value of I_d (which is $\overline{i}_d$) propagates to L and hence L becomes $i_1 \oplus i_2 \oplus \ldots \oplus i_d \oplus \ldots \oplus i_p \oplus fw_1 \oplus fw_2 \oplus \ldots \oplus fw_q \oplus \overline{fx_1} \oplus \overline{fx_2} \oplus \ldots \oplus \overline{fx_r} \oplus fy_1 \oplus fy_2 \oplus \ldots \oplus fy_s \oplus \overline{fz_1} \oplus \overline{fz_2} \oplus \ldots \oplus \overline{fz_t} \oplus i_1 \oplus i_2 \oplus \ldots \oplus \overline{i}_d \oplus \ldots \oplus i_p \oplus fw_1 \oplus fw_2 \oplus \ldots \oplus fw_q \oplus \overline{fx_1} \oplus \overline{fx_2} \oplus \ldots \oplus \overline{fx_r} \oplus fy_1 \oplus fy_2 \oplus \ldots \oplus fy_s \oplus \overline{fz_1} \oplus \overline{fz_2} \oplus \ldots \oplus \overline{fz_t}$
$= i_d \oplus \overline{i}_d$ = 1.
Since at the end, the line L contains 1, the fault is detected.

Case 2. A fault can also occur on L. This causes L to have the value 1 since it was initialized with a 0. Hence, the circuit detects the fault. Note that a fault on I_d can cause other lines faulty (Case 1). However, a fault on L does not propagate to any I_d since controls of ETGs and CNOTs are not connected to L. ◀

Consider an online testable circuit given in Figure 6. In the circuit, ETGs are labeled as e_1, e_2, e_3, and e_4. The initial values of lines I_1, I_2, I_3, I_4, and I_5 are i_1, i_2, i_3, i_4, and i_5. The parity line L is initialized with a 0. Assume that ETGs e_1, e_2, e_3, and e_4 compute f_1, f_2, f_3, and f_4 when the circuit is fault free. Before the first ETG e_1, the value of L is $i_1 \oplus i_2 \oplus i_3 \oplus i_4 \oplus i_5$. Let, $l = i_1 \oplus i_2 \oplus i_3 \oplus i_4 \oplus i_5$.

The ETG e_1 computes $f_1 = i_1 i_2$. Thus after this gate, the values of I_3 and L are $i_3 \oplus f_1$ and $l \oplus f_1$. Assume a fault occurs on I_2 between e_1 and e_2, which changes the value of I_2 from i_2 to $\overline{i}_2$. This fault affects e_2; thus e_2 computes $\overline{f}_2$ =

$\bar{i}_2$. Note that the fault-free values of I_3 and L, after the gate e_2, are $i_3 \oplus f_1 \oplus f_2$ and $l \oplus f_1 \oplus f_2$, respectively. However, due to the fault, the values are $i_3 \oplus f_1 \oplus \overline{f}_2$ and $l \oplus f_1 \oplus \overline{f}_2$. Thus two targets of e_2 cause the lines I_3 and L to have the faulty values.

Now consider the ETG e_3. One of the controls connected to I_2 and target connected to L are faulty. Let, i_4 be 0. Thus the fault on I_2 does not affect e_3 since other fault-free control bit, connected to I_4, has the value 0. The fault on L does not have any impact on calculating the function $f_3 = \bar{i}_2 i_4 = \bar{i}_2.0 = 0$. After this gate, the value of I_5 is $i_5 \oplus f_3$, and the value of L is $l \oplus f_1 \oplus \overline{f}_2 \oplus f_3$.

The ETG e_4 has the control on I_3 and targets on I_2 and L, which are faulty. Assume that other fault-free control bit, connected to I_1, has the value 1. Thus the fault on I_3 affects e_4 in calculating f_4. As a result, e_4 computes $\overline{f}_4 = i_1(i_3 \oplus f_1 \oplus \overline{f}_2)$. After the operation of this gate, the value on I_2 is $\bar{i}_2 \oplus \overline{f}_4$, and the value on L is $l \oplus f_1 \oplus \overline{f}_2 \oplus f_3 \oplus \overline{f}_4$.

At the end, when I_1, I_2, I_3, I_4, and I_5 are EXORed to L again, the value of L becomes
$l \oplus f_1 \oplus \overline{f}_2 \oplus f_3 \oplus \overline{f}_4 \oplus i_1 \oplus \bar{i}_2 \oplus \overline{f}_4 \oplus i_3 \oplus f_1 \oplus \overline{f}_2 \oplus i_4 \oplus i_5 \oplus f_3 = l \oplus i_1 \oplus \bar{i}_2 \oplus i_3 \oplus i_4 \oplus i_5$
$= i_1 \oplus i_2 \oplus i_3 \oplus i_4 \oplus i_5 \oplus i_1 \oplus \bar{i}_2 \oplus i_3 \oplus i_4 \oplus i_5 = 1.$
Since L is 1, the circuit detects the fault.

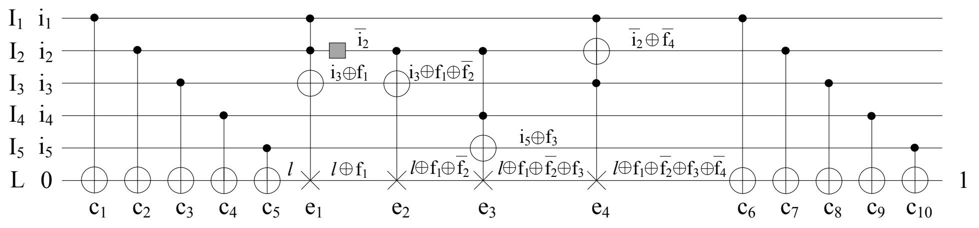

Figure 6. Fault detection in testable circuit.

VI. Experimental Results

We conducted our experiments for 20 benchmark circuits collected from [28]. Toffoli networks were generated using improved ESOP-based synthesis described in [29]. We then applied our approach to make the circuits online testable. In Table I , we compare the testable circuits generated by our approach to that of previously reported approaches [22], [23], [26], [24], [25] in terms of quantum cost and garbage outputs. Note that QC and GO columns in the table represent the quantum cost and the number of garbage outputs, respectively. Our approach shows significant results in reducing the quantum cost and garbage outputs for all circuits.

Table II summarizes the improvements achieved by our approach. Quantum costs are decreased by 74.72%, 78.68%, and 26.30% in average, compared to ([22], [23]), [26], and ([24], [25]), respectively. Reductions of garbage outputs range from 96.89% to 99.84% in average, compared to the previous approaches.

The percentages of overheads in quantum cost and garbage outputs for integrating the online testability feature are given in Table III. The quantum cost overhead for our approach is only 4.21% in average, compared to 312.28% for the approach in [22], [23], 388.67% for the approach in [26], and 41.40% for the approach in [24], [25]. It is interesting to note that our approach has absolutely no overhead in terms of garbage outputs since the testability feature does not produce any extra garbage. However, existing approaches produce extremely large number of garbage outputs; thus the average overheads are 63409.48%, 5163.03%, and 3113.74% for the approaches in ([22], [23]), [26], and ([24], [25]), respectively.

We have also noticed an average improvement of 48.80% in quantum cost results reported from the initial version [27] of this approach. It is worth noting that our initial work was restricted only to a particular type of ESOP-based approach; thus we generally cannot expect better results than the results that we obtained from our initial work. For our proposed approach, although we chose an improved ESOP-based synthesis [29] for experiments, it is possible to use any type of synthesis as long as the resulting reversible circuit contains only Toffoli gates including NOT and negative-control Toffoli gates. As a result, this approach is more flexible, and the results that we have shown in the "Our approach" column of Table I could possibly be even further optimized by choosing a more efficient reversible logic synthesis approach.

VII. Conclusion

In this paper we first discuss the limitations of previous approaches reported in [22], [23] and [24], [25]. We then present a technique that adds online testability to any reversible circuit built as cascade of Toffoli gates. There is no restriction on how the initial cascade is generated, just that the circuit be designed only from these gates. As indicated in Section II, there

Table I
COMPARISON OF DIFFERENT ONLINE TESTABLE TECHNIQUES

Circuit	Approach in [22], [23]		Approach in [26]		Approach in [24], [25]		Our approach	
	QC	GO	QC	GO	QC	GO	QC	GO
9symml	25598	5952	32220	532	12262	139	11066	9
apex5	195061	45362	213843	3518	47847	1620	35354	117
apla	17815	4142	20574	334	2934	190	1859	10
bw	19019	4422	25140	386	4464	723	1249	5
cm82a	2163	502	2823	50	385	48	173	5
con1	1647	382	1929	38	307	30	184	7
cu	5560	1292	6234	110	1288	84	876	14
dk17	10376	2412	11478	186	1649	95	1113	10
ex2	1733	402	2241	42	267	23	171	5
ex5p	84852	19732	98898	1540	12484	1623	4918	8
f51m	128927	29982	164889	2670	37323	832	29072	14
frg2	218109	50722	247332	4048	150509	3616	115211	143
max46	15321	3562	19314	326	5585	114	4627	9
misex3	141354	32872	174732	2816	69153	2149	50826	14
rd73	16482	3832	20583	342	1617	115	959	7
sqn	11236	2612	14256	240	2053	100	1437	7
sqrt8	5689	1322	7086	122	835	63	530	8
sym9	24652	5732	31890	528	12262	139	11066	9
table3	223054	51872	265566	4294	31011	1767	20050	14
z4ml	3883	902	5070	88	1048	92	575	7
Average	57626.55	13400.5	68304.9	1110.5	19764.15	678.1	14565.8	21.1

Table II
IMPROVEMENTS ACHIEVED BY OUR APPROACH

	Parameter	Existing approaches		
		Approach in [22], [23]	Approach in [26]	Approach in [24], [25]
Achieved improvements	QC	74.72%	78.68%	26.30%
	GO	99.84%	98.10%	96.89%

are a number of approaches that can be used to generate such a circuit. Our technique consists of making small modifications to the existing gates, and then simply adding a number of gates and a parity line to the circuit. Thus the overhead in terms of quantum cost is very small, and in terms of garbage our overhead is 0. This is a startling result, as previous approaches in this area require a significant increase in both of these measures. Table III shows this very clearly. Previous work in [27] introduced an initial version of this approach which was limited to only reversible circuits generated by a particular synthesis approach. Not only have we extended the technique to any type of Toffoli gate cascade, but we have also found a significant improvement in the overhead. Future work will include investigations as to how this technique can be extended to multiple-valued reversible circuits and to reversible circuits that incorporate other types of gates.

ACKNOWLEDGMENT

This research was funded by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] C. H. Bennett, "Logical reversibility of computation," *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973.

[2] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, pp. 183–191, 1961.

Table III
OVERHEAD CALCULATION OF THE TESTABLE DESIGN OVER THE NON-TESTABLE DESIGN

Method	Overhead for testability feature	
	QC	GO
Approach in [22], [23]	312.28%	63409.48%
Approach in [26]	388.67%	5163.03%
Approach in [24], [25]	41.40%	3113.74%
Our approach	4.21%	0%

[3] M. P. Frank, “Introduction to reversible computing: motivation, progress, and challenges,” in *Proceedings of the 2nd Conference on Computing Frontiers*, Ischia, Italy, 4-6 May 2005, pp. 385–390.

[4] “International Technology Roadmap for Semiconductors (ITRS),” http://public.itrs.net, 2009 executive summary.

[5] A. N. Al-Rabadi, *Reversible Logic Synthesis: From Fundamentals to Quantum Computing*. Springer-Verlag, 2004.

[6] W. C. Athas and L. J. Svensson, “Reversible logic issues in adiabatic CMOS,” in *Proceedings of the Workshop on Physics and Computation (PhysComp)*, Dallas, TX, 1994, pp. 111–118.

[7] P. Picton, “Optoelectronic, multivalued, conservative logic,” *International Journal of Optical Computing*, vol. 2, pp. 19–29, 1991.

[8] R. C. Merkle, “Reversible electronic logic using switches,” *Nanotechnology*, vol. 4, no. 1, pp. 21–40, 1993.

[9] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, “Synthesis of reversible logic circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 710–722, 2003.

[10] J. Chen, X. Zhang, L. Wang, X. Wei, and W. Zhao, “Extended Toffoli gate implementation with photons,” in *Proceedings of the 9th International Conference on Solid-State and Integrated-Circuit Technology (ICSICT)*, China, 20-23 Oct 2008, pp. 575–578.

[11] D. M. Miller, D. Maslov, and G. W. Dueck, “A transformation based algorithm for reversible logic synthesis,” in *Proceedings of the 40th Annual Design Automation Conference (DAC)*, 2003, pp. 318–323.

[12] P. Gupta, A. Agrawal, and N. Jha, “An algorithm for synthesis of reversible logic circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2317–2330, 2006.

[13] K. Fazel, M. Thornton, and J. E. Rice, “ESOP-based Toffoli gate cascade generation,” in *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, Victoria, BC, Canada, 22-24 Aug. 2007, pp. 206–209.

[14] J. E. Rice and V. Suen, “Using autocorrelation coefficient-based cost functions in ESOP-based Toffoli gate cascade generation,” in *Proceedings of the 23rd Canadian Conference on Electrical and Computer Engineering (CCECE)*, Calgary, Canada, May 2010, downloaded Jun. 2010 from `http://www.cs.uleth.ca/~rice/publications/CCECE2010.pdf`.

[15] A. N. Al-Rabadi, “New classes of Kronecker-based reversible decision trees and their group-theoretic representations,” in *Proceedings of the International Workshop on Spectral Methods and Multirate Signal Processing (SMMSP)*, 2004, pp. 233–243, Vienna, Austria, September 11-12.

[16] A. Mishchenko and M. Perkowski, “Fast heuristic minimization of exclusive sum-of-products,” in *Proceedings of the 5th International Reed-Muller Workshop*, Starkville, Mississippi, August 2001, pp. 242–250.

[17] Z. Hamza and G. W. Dueck, “Near-optimal ordering of ESOP cubes for Toffoli networks,” in *Proceedings of the 2nd Annual Workshop on Reversible Computation (RC)*, 2010, July 2–3 Bremen, Germany.

[18] Y. Sanaee, “Generating Toffoli networks from ESOP expressions,” Master’s thesis, University of New Brunswick, 2010.

[19] M. Mohammadi and M. Eshghi, “On figures of merit in reversible and quantum logic designs,” *Quantum Information Processing*, vol. 8, pp. 297–318, August 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1555567.1555601

[20] D. Maslov, “Reversible logic synthesis benchmarks page,” `http://www.cs.uvic.ca/~dmaslov/`.

[21] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, “Elementary gates for quantum computation,” *Phys. Rev. A*, vol. 52, no. 5, pp. 3457–3467, Nov 1995.

[22] D. P. Vasudevan, P. K. Lala, and J. P. Parkerson, “Online testable reversible logic circuit design using NAND blocks,” in *Proceedings of the IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, Los Alamitos, CA, USA, 10-13 October 2004, pp. 324–331.

[23] D. P. Vasudevan, P. K. Lala, D. Jia, and J. P. Parkerson, “Reversible logic design with online testability,” *IEEE Transactions on Instrumentation and Measurement*, vol. 55, no. 2, pp. 406–414, 2006.

[24] S. N. Mahammad, S. Hari, S. Shroff, and V. Kamakoti, “Constructing online testable circuits using reversible logic,” in *Proceedings of the 10th IEEE International VLSI Design and Test Symposium (VDAT)*, Goa, India, August 2006, pp. 373–383.

[25] S. N. Mahammad and K. Veezhinathan, “Constructing online testable circuits using reversible logic,” *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 1, pp. 101–109, 2010.

[26] N. Farazmand, M. Zamani, and M. B. Tahoori, “Online fault testing of reversible logic using dual rail coding,” in *Proceedings of the 16th IEEE International On-Line Testing Symposium (IOLTS)*, 5-7 July 2010, pp. 204–205.

[27] N. M. Nayeem and J. E. Rice, “A simple approach for designing online testable reversible circuits,” in *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, Victoria, B.C., 23-26 August 2011.

[28] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, “RevLib: An online resource for reversible functions and reversible circuits,” in *Proceedings of the 38th International Symposium on Multiple Valued Logic*, 2008, pp. 220–225, RevLib is available at `http://www.revlib.org`.

[29] N. M. Nayeem and J. E. Rice, “Improved ESOP-based synthesis of reversible logic,” in *Proceedings of the Reed Muller Workshop*, Tuusula, Finland, 25-26 May 2011.