J Electron Test (2017) 33:591–605
DOI 10.1007/s10836-017-5679-4

Efficient Techniques for Fault Detection and Correction of Reversible Circuits

Hafiz Md. Hasan Babu [1] · Md. Solaiman Mia [1] · Ashis Kumer Biswas [2]

Received: 22 March 2017 / Accepted: 25 July 2017 / Published online: 9 August 2017

Abstract It is very important to detect and correct faults for ensuring the validity and reliability of reversible circuits. Test vectors play an important role to detect as well as correct the faults in the circuits. The optimum number of test vector implies the more capabilities for detecting several types of faults in the circuits. In this paper, we have proposed an algorithm for generating optimum test vectors. We have shown that the proposed algorithm generates optimum test vectors with the least complexity of time as compared to existing methods, i.e., we have proved that the proposed algorithm requires *O(log$_2$N)* time, whereas the best known existing method requires *O(N. log$_2$N)* time, where *N* is the number of inputs. We have also proposed another algorithm for detecting faults using the generated test vectors. This proposed method can detect more faults than existing ones. We have proved that the proposed fault detection algorithm requires least time complexity as compared to the best known existing methods, i.e., the proposed algorithm requires *O(d. 1/N)* time, whereas the best known existing methods require *O(d. N)* time, where *N* is the number of inputs and *d* is the number of gates in a reversible circuit. Finally, we have proposed another algorithm for correcting the detected faults. We have also proved that the proposed methods require the least time complexity as compared to the best known existing methods. In addition, the experimental results using benchmark circuits show the efficiency of the proposed methods.

Keywords Reversible Circuit · Optimum Test Vector · Fault Detection · Fault Correction

1 Introduction

Reversible logic computing has received significant research attention recently [41]. Applications of the reversible circuits can be found in the emerging fields of the quantum computation, low-power computation and digital signal processing. Conventional irreversible logic circuits necessarily dissipate energy due to the erasure of information [6, 20]. But reversible computation can be performed with arbitrarily small energy dissipation. Bennett showed that the power dissipation in reversible logic computing is zero under ideal physical circumstances [6].

Quantum computing is also a rapidly emerging area [28]. Since quantum logic operations are inherently reversible, studying the testing of reversible circuits may assist in the development of physical realizations of quantum gates based on trapped ion technology, photons and non-linear optical media, cavity-quantum electrodynamic devices, spin in semiconductors, etc.

To ensure the validity and reliability of reversible logic circuits, fault detection is necessary. Testing is one of the final phases of the production of a circuit. A test set is a set of test vectors that are applied to the circuit for detecting faults. Generating efficient test vectors is hard for conventional irreversible circuits. On the other hand, for reversible circuits it is significantly simpler. Fault detection probability is the greatest when the test vectors of a reversible circuit are maximized [1].

Responsible Editor: V. D. Agrawal

✉ Hafiz Md. Hasan Babu
hafizbabu@du.ac.bd

[1] Department of Computer Science and Engineering, University of Dhaka, Dhaka, Bangladesh

[2] Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX, USA

An increasingly important problem in digital circuit design is the correction of detected faults. Traditionally, these faults have been a major concern for space applications due to circuit exposure to high radiation levels. With the introduction of smaller device geometries and new process technologies, these faults are now becoming an issue for ground-level applications as well [5]. Several methods have been proposed to protect circuits from the effects of these faults. These range from the use of specific manufacturing techniques to detect and correct the faults in reversible circuits [27].

Testing can be performed either online or offline. Online testing implies that the testing can be performed during the normal mode of operation of the circuit with hardware overhead. But offline testing needs a dedicated test mode without any test hardware overhead. In addition, the offline techniques of testing will not add any overhead in terms of hardware complexity but they will have performance overhead since the circuit needs to be worked in test mode for offline testing [25].

Previous works regarding testing reversible circuits includes ATPG (Automatic Test Pattern Generation) methods [34], fault detection in quantum circuits [7], ILP (Integer Linear Program) method for stuck-at faults [32], DFT (Design for Test) method for missing gate faults [15] and fault modeling for reversible circuits based on quantum technology [2, 36].

Five main contributions are addressed in this paper:

1) A test vector generation algorithm is proposed to generate optimum number of test vectors which will be used to detect the faults in reversible circuits.
2) The proposed test vector generation algorithm requires the minimum complexity of time in the literature till now.
3) A fault detection algorithm is proposed to detect all possible single and multiple bit faults in reversible circuits.
4) The proposed fault detection algorithm requires the minimum complexity of time in the literature till now.
5) A fault correction approach has also been introduced to correct the detected faults.

The remainder of this paper is organized as follows: Section 2 provides some background information about reversible logic gates/circuits, followed by overviews of existing works on TPG (Test Pattern Generation), detection and correction of faults in a reversible circuit in Section 3. Section 4, 5 and 6 describes our proposed optimum test vector generation algorithm, fault detection method and fault correction techniques respectively. Some comparisons between best known existing methods and our proposed method are shown in Section 7. Finally, a brief conclusion along with the goal of offering a new perspective on our proposed optimum test vector generation algorithm, detection and correction techniques of faults in a reversible circuit are given in Section 8.

2 Background Studies

In this Section, we will discuss about some of the basic elements of reversible logic, fault model and reversible circuits.

2.1 Reversible Gate

A logic circuit is reversible if it computes a bijective (one-to-one and onto) logic function, so that the circuit's input can be reconstructed from its output. A reversible circuit usually consists of smaller sub-circuits or gates which are themselves reversible. Reversible gates do not allow any fanout [29]. Commonly studied reversible gates include: NOT, Feynman /CNOT (Controlled-NOT), Toffoli, Fredkin and generalized Toffoli gates [12, 14]. Of these, NOT, CNOT and Toffoli gates are basically n-bit Toffoli gate with $n = 1, 2$ and 3 respectively which is defined as generalized Toffoli gates. The behavior of some reversible gates is defined as follows:

NOT: $i' = 1 \oplus i$
CNOT: $i' = i, j' = i \oplus j$
Toffoli: $i' = i, j' = j, k' = k \oplus ij$
Fredkin: $i' = i, j' = j \oplus ij, k' = k \oplus ij \oplus ik$

Fig. 1 shows the standard graphic symbols for some common reversible gates, where *0*-CNOT, *1*-CNOT and *2*-CNOT correspond to an ordinary NOT gate or inverter, Feynman and Toffoli gates, respectively.

2.2 Garbage Output

Unwanted or unused output of a reversible gate (or circuit) is known as **garbage output** [3], i.e., the output(s) which is (are) needed only to maintain the reversibility is (are) known as garbage output(s). Heavy price is paid off for each garbage output.

Example 1 Feynman Gate (FG) [1] is used to perform Exclusive-OR between two inputs. But in this case, one extra output will be generated as well, which is the garbage output as shown in Fig. 2 with *.

2.3 Constant Input

Constant inputs are the inputs of a reversible gate (or circuit) that are either set to 0 or 1 [1].

Example 2 If the complement of the input A from Fig. 2 is needed, then set B to *1* and get $Q = A'$.

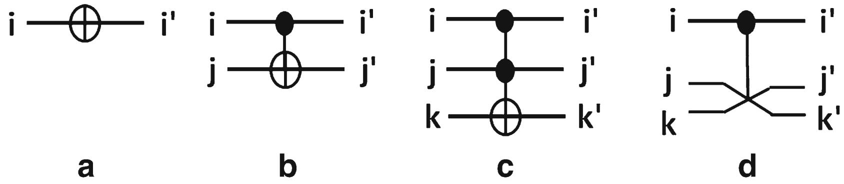

Fig. 1 Reversible Classical Gates: (a) 0-CNOT, (b) 1-CNOT (Feynman), (c) 2-CNOT (Toffoli) and (d) Fredkin

2.4 Reversible Circuits

A reversible circuit can be divided into several *levels* [32]. The dotted lines of Fig. 3 represent the *levels* in the *3*_17tc circuit. Any state of any *level* of a reversible circuit can be generated by appropriate input and any single fault changes an intermediate *level* in the circuit; it also changes the output of the circuit. An example reversible benchmark circuit *3_17tc* in Fig. 3 shows that the inputs are at *level 0* and the outputs are at *1* plus the maximum level of its inputs. The *depth d* is used interchangeably with *level* and is defined as the maximum level [31]. It can be no longer than the number of gates in the circuit. We will often find it convenient to use an *n*-bit vector to refer to the value of the wires at a given *level* in the circuit [32].

2.5 Quantum Cost

Reversible circuits can be measured in terms of quantum cost. Every quantum circuit is built using 1 × 1 and 2 × 2 quantum primitives and its cost is calculated as a total sum of 2 × 2 gates used since 1 × 1 gate has no cost, i.e., zero [17]. Basically, the quantum primitives are matrix operation which is applied on qubits state [38]. All the gates of the form 2 × 2 has equal quantum cost and the cost is unity, i.e., one [9]. Since every reversible gate consists of 1 × 1 or 2 × 2 quantum gate, the **quantum cost** of a reversible circuit calculates the total number of 2 × 2 gates used.

Example 3 The quantum costs of Feynman gate [9], Feynman Double gate (F2G) [9], Fredkin gate (FRG) [38], Peres gate [33] and DPG gate (as full adder) [17] are one, two, five, four and six respectively.

2.6 Quantum Fault Model

There are different types of fault models in the literature such as initialization faults, bit-flips and faded control faults [8]. A line (qubit) is incorrectly initialized to some erroneous starting value at initialization faults where in bit-flips a qubit changes its value at an undesired location/time [13]. In faded control faults, the control of a gate does not behave as desired. However, the focus of our paper is on bit-flips faults which can be considered to be at the switch level of abstraction, where the SAF (Stuck-at-Fault), MGF (Missing-Gate-Fault) and other fault models can be considered to be at the structural level.

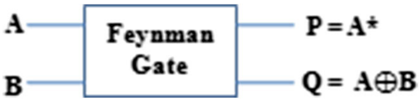

Fig. 2 *Illustrating Garbage Output*

3 Existing Test Pattern Generation, Fault Detection and Correction Approaches

It has been pointed out that finding a test vector for a given fault is a trivial task in the reversible case. A traditional way of generating a complete test for quantum bit faults would be to generate a test vector for each individual fault and then compress the test set. There are two types of testing: offline testing and online testing. In offline testing, a circuit under test is taken out of its normal mode of operation. In contrast, online testing is carried out while the circuit is being used for normal operations. In the case of online testing, additional circuitry is attached to the original circuit to determine whether the system is faulty or fault free. Many researchers focus on online testing and many others also focus on offline testing. There are many current researches ongoing focusing on the offline testing approach [23–25]. In this paper, we use the offline testing approach for fault detection and correction of the faults. In this section, we show some existing methods for finding test vectors in reversible circuits.

When we get some test vectors which are generated for a particular circuit, the fault detection of that circuit is necessary. The faults of a circuit can be detected using the generated test vectors. After detection of faults, it is required to correct that faults in an efficient way. In this section, we also show some existing approaches for detecting and correcting of faults in a circuit.

3.1 Existing Test Pattern Generation Approaches

In this subsection, we discuss some of the existing test pattern generation approaches with their limitations compared to our proposed optimum test vector generation algorithm.

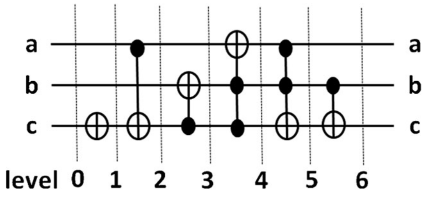

Fig. 3 *Reversible Benchmark Circuit 3_17tc*

3.1.1 Test Generation Algorithm [10]

In [10], a DFT method is proposed to make an arbitrary reversible logic circuit composed of *n*-bit Toffoli gates fully testable for single intra-level bridging faults and single stuck-at faults. This method gives a test generation algorithm to generate the log_2N test vectors for testing bridging faults at any level.

But, [10] considered testing of circuits composed of only *n*-bit Toffoli gates. We know that, adding one control point increases the cost of 3, 4, 5 and 6-bit Toffoli gates by 8, 16, 32 and 64 respectively [30]. As a result, the overhead becomes large for smaller circuits.

3.1.2 Automatic Test Pattern Generation [35]

An overview of some recent developments in the testing and design validation of reversible circuits is provided in [35]. This method gives a classification of ATPG algorithms for reversible circuits which is divided into two classes: exact algorithms and heuristics.

[35] shows that, reversible circuits have very small test sets for most fault models. But in this paper, we describe that optimum test set can detect more faults in a circuit.

3.1.3 Deterministic ATPG Algorithm [24]

The method described in [24] is developed on the basis of set cover approach where the minimal test set is chosen from superset of test vectors. This approach is validated for various benchmark circuits and also few circuits are designed with family of Toffoli, Peres and Fredkin gates to generate complete test set which can detect all the intra level Bridging Faults in a given reversible circuit.

The choice of a value of *N* in [24] does not have much impact on the smaller circuits, but, for large circuits, when the value of *N* increases, the execution time grows exponentially. Also, the approach in [24] only detects intra level Bridging Faults, it does not detect inter level Bridging Faults.

3.2 Existing Fault Detection Approaches

In this subsection, we discuss some of the existing fault detection approaches with some of their limitations compared to our proposed fault detection algorithm.

3.2.1 Detection of Missing Gate Faults [23]

[23] presents the testability issue of reversible circuits under the missing gate fault model. Boolean difference technique for deriving the test set for detecting all faults in a reversible circuit is implemented with k-CNOT gates. Then an optimizing algorithm is used to derive optimal test set to detect all possible partial missing faults in a reversible circuit.

[23] shows that for an $N \times N$ reversible circuit with *d* gates, realized with quantum k-CNOT gates, the time complexity of the generation of Boolean difference is *O(d. N)*, where *N* is the number of inputs and *d* is the number of gates in a reversible circuit. A Boolean difference technique is found to be sufficient for testing all such detectable faults, but, in the proposed method, we have shown that the time complexity of the test vector generation method is $O(log_2N)$ and $O(log_2N)$ time is also sufficient for testing all such detectable faults.

3.2.2 Error Detection using Reversible Gates [18]

Single error correction-double error detection (SEC-DED) is proposed in [18]. The design is done using a new 4 × 4 reversible gate called `HCG' for implementing hamming error coding and detection circuits are shown. Detection of faults generated in a circuit is done by using parity-preserving reversible logic gates.

Different implementations for the reversible (7,4) hamming error coding and detection circuits are presented in [18]. But, in our proposed method, we have shown the self-complemented fully redundant reversible circuit which requires less complexity of running time compared to [18].

3.2.3 Detection of Single and Multiple Missing-gate Faults [11]

Two methods are given to generate the complete test set for reversible circuits to detect missing-gate faults is given in [11]. The methods are dividing subcircuit method and the set covering method. If more than one missing-gate faults appeared in a subcircuit, then [11] adds other vectors to detect them.

But in [11], the dividing subcircuit method does not need any additional optimal software, so it can deal with larger circuits, but the complete test set generated by this is not optimum. On the other hand, the set covering method can generate the minimal test set and it depends on the optimal software, such as CPLEX and LINDO.

3.3 Existing Fault Correction Approaches

In this subsection, we discuss some of the existing fault correction approaches with some of their limitations in the literature till now. It is noted here that, there is no such fault correction approaches for reversible circuits in the literature. So, in this subsection, any fault correction approaches for reversible circuits is not given.

3.3.1 Error Correction using Self-checked Circuits [21]

A group of error detecting/correcting code (EDC-ECC) based self checked/corrected/timed circuits for logic robustness and performance scalability in nanoscale VLSI design is proposed in [21]. The EDC self-checked circuits achieve increased reliability enhancement with comparable hardware overhead, or reduced hardware overhead for the same level of reliability.

But, a self-checked circuit slows down at the occurrence of a soft error or a timing violation for logic correctness, similar to a Razor logic circuit. A self-timed circuit has the same performance scalability as an equivalent handshaking protocol-based asynchronous circuit in the presence of parametric variations.

3.3.2 Error Correcting Code in SRAMs [26]

In [26], a double error correcting error-correcting codes (DEC-ECC) implementation technique suitable for SRAM applications is presented. For binary vectors, an erroneous bit is corrected merely by complementing an error location decoder. The error corrector circuit shown in [27] is simply a stack of XOR gates.

[26] shows that, the ECC encoders and decoders can be implemented without a significant area impact. But it is not notified in [26] that, the extra redundancy required within the memory array for a particular code is a function of the fault detection and correction capability of that code and block size.

3.3.3 Error Correction of Digital Circuits [16]

An efficient symbolic method for automatic fault correction of both combinational and synchronous sequential circuits is presented in [16]. Several optimizations are also introduced in [16]. All optimizations are safe, meaning that they neither affect the number of computed solutions nor do they decrease the quality of results.

The methods shown in [16] are efficient for digital circuits. But, when the mechanisms are imposed to a reversible circuit, rectification is increased due to a combinational problem. Then the strategies shown in [16] become unoptimized.

4 Proposed Optimum Test Vector Generation Algorithm

A fault is a logical model of a physical disturbance, which changes the Boolean function of the circuit. Faults are detected by applying a test vector at input of the circuit and observe whether the output is same as the input. If it does, then we proceed to the next test vector. If it does not, then some wire(s) of the circuit must be faulty.

It is proved that optimum test vectors can detect more faults in a circuit [1]. Our proposed algorithm, unlike other greedy algorithms, does not search exhaustively to find a minimum test set. Yet it aims to generate test vectors comparably optimum in size by the method of heuristics.

While generating test vectors, it is much time consuming to generate all possible combinations, since some of the test vectors have logically no operation for testing a circuit. For this reason, *000* and *001* patterns are discarded in the proposed Algorithm 1 (Step 3.a), because from the beginning of testing of a circuit, one or more control bit is logic *0*. Algorithm 1 also discards the pattern consisting of all *1*'s, because when all inputs of a circuit become logic *1*, every gate of the circuit remains simultaneously active. Testing a circuit with all active control bits require more times and more calculation complexity. As a result, the pattern *111* (Step 3.b) is discarded.

Algorithm 1: Optimum Test Vector Generation Algorithm

```
1.  Read in the specification of a reversible circuit and compute the number of
    Constant Inputs and Garbage Outputs.
2.  If the circuit has $i$ number of Constant Inputs, then generate test vectors $V$ for
    $N = N - i$ inputs.
3.  Produce test patterns for all possible combinations while
        a. Discard test vectors consisting of first $[N/2]$ $0$'s. [These test vectors
           have logically no operation for testing a circuit, since from the
           beginning of testing of a circuit, one or more control bit is logic $0$]
        b. Discard test vectors consisting of all $1$'s. [Testing a circuit with all
           active control bits require more times and more calculation
           complexity]
4.  Get the test vectors $V$ for testing a reversible circuit to detect faults.
```

In this paper, we have proposed such a test vector generation algorithm which generates optimum test vectors for detecting faults in a circuit compared to existing researches. Our algorithm also requires less complexity of execution time compared to others [10, 24, 35]. Total number of inputs in a reversible circuit is expressed with *N* in our algorithm. The method is stated in Algorithm 1.

The proposed algorithm works with four steps. The working principle of the algorithm is described below:

Step 1: The algorithm checks whether there is any constant inputs (input is set 0 or 1 [30]) and garbage outputs (unused output [37]) in the circuit. Consider the example circuit *3_17tc* in Fig. 3. The circuit has no constant input and no garbage output while Fig. 4 illustrates *rd32* circuit with one constant input and two garbage outputs.

Step 2: Since the example circuit *3_17tc* in Fig. 3 has no constant input and garbage output, our proposed algorithm generates test vectors for three inputs and three outputs.

Step 3: Now generate all possible combinations of inputs and discard the patterns which are not needed.

Step 3.a: Test patterns 000 and 001 are discarded.

Step 3.b: Test pattern 111 is discarded.

Step 4: Get the test vectors V = {010, 011, 100, 101 and 110}.

Finally, we can conclude from our proposed algorithm, the generated test vectors for *3_17tc* are 010, 011, 100, 101 and 110. These test vectors have to pass through a circuit for detecting the faults in that circuit.

The comparison table in Section 7 shows that our proposed algorithm generates five test vectors for *3_17tc*, where other methods generate a less number of test vectors.

Theorem 1 The proposed Optimum Test Vector Generation Algorithm requires $O(log_2N)$ complexity of time, where N is the number of inputs.

Proof Assume that N is the number of inputs and we represent the algorithm as the following expression, where a constant term log_24 is used to indicate the discarded test vectors. For example, the *4*-bit with maximal gate count circuit has different $log_2N = log_24 = 2$ types of control inputs, i.e., 0 and 1. Therefore, the expression is as follows:

$$\mathrm{T(N) = 2.T(\sqrt{N}) + log_2 4}$$

Then with a change of variables, we can simplify the recurrences. For the recurrence, let $m = log_24$. The value of m yields $T(2^m) = 2.\ T(2^{m/2}) + m$.

We can now rename the recurrence with $S(m) = T(2^m)$ to produce a new recurrence:

$$\mathrm{S(m) = 2.S(m/2) + m}$$

Indeed, the new recurrence has the same solution:

$$\mathrm{S(m) = O(m.}\mathit{logm}\mathrm{)}$$

So, $T(N) = T(2^m) = S(m) = O(m.logm)$.

$= O(log_24.loglog_24)$.

$= O(2.log2)$.

Now, if we think, $N = 100$, then we replace $N = 100$ by $N = log^{-1}2$, and we get, $2 = logN$ as $log100 = 2$.

So, the resulting recurrence is: $T(N) = O(2.log2)$.

$= O(logN.log2)$.

$= O(log_2N)$ [since $(log_kN).(log_k2) = (log_2N)$].

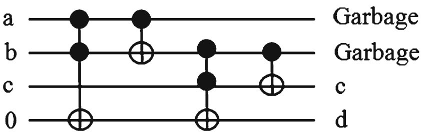

Fig. 4 *rd32 circuit with Constant input and Garbage outputs*

Finally, the complexity of Algorithm 1 is gained i.e., $O(log_2N)$.

Example 4 For computing the running time of the proposed optimum test vector generation algorithm, we see that, for 2, 4 and 8 bits, it requires 1, 2 and 3 ms of running time respectively. So, all the above calculations prove that the time complexity of the proposed algorithm is $O(log_2N)$.

Lemma 1 Let N be the number of inputs, then the 4-bit with maximal gate count circuit requires at least $2^N – (N + 1)$ test vectors to detect the faults.

Proof Assume that an N input reversible circuit consisting of m number of $k – CNOT$ gates, where $m > = 0$ is required for generating optimum test vectors. From the specification of the *4*-bit with maximal gate count circuit, we see that the circuit has no constant inputs and no garbage outputs [22]. So, if we have $0 – CNOT$ and/or $1 – CNOT$ gates in the circuit, the previous value is passed through the circuit without changing its value. Thus, without losing any generality, we assume that we have only $k – CNOT$ gates.

Let any test vector V be $V_1V_2...V_n$ where V_i is i^{th} bit and g_i be the set of positions, i.e., wire indices, where gate g_i has control inputs $1 < = i < = m$, then $g_i = \{g_{i1}, g_{i2}, ..., g_{ip}\}$ $(1 < = p < = N – 1)$ means gate g_i has control inputs on wires having indices $g_i = g_{i1}, g_{i2}, ..., g_{ip}$.

If $m = 0$, then 2^N test vectors are generated and no speciality of the circuit belongs to the V, but, in the *4*-bit with maximal gate count circuit has m number of k - $CNOT$ gates, where $m > =7$.

As a result, by using Algorithm 1, the first $N/2\ 0^k$ and 1^k (where 0^k and 1^k means k consecutive zeros and ones) is discarded for the *4*-bit with maximal gate count circuit. The test vectors of the first $N/2\ 0^k$ is the same as the number of N for g_i of the circuit.

So, resulting number of V be at least 2^N - N - 1 which is $2^N –$ $(N + 1)$ thus completes the proof.

Example 5 In *rd32* circuit shown in Fig. 4, we can see that it has four inputs and four outputs, where one input is control input and two outputs are garbage outputs. If we apply the proposed Algorithm 1 into the *rd32* circuit, then we get the test vectors are $V = \{010, 011, 100, 101$ and $110\}$.

5 Proposed Fault Detection Algorithm

A problem that may arise during fault detection in a circuit is that the given set of test patterns is not sufficient in order to distinguish all faults. Then, further test patterns have to be generated. Therefore, Algorithm 1 is applied to reversible circuits which determine not only an arbitrary test pattern, but also a pattern which distinguishes a fault from another. In addition, sometimes not all faults are distinguishable from each other [39].

Therefore, Algorithm 2 is introduced here for three reasons:

1) Use the optimum test vectors generated from Algorithm 1, test a reversible circuit whether there is any fault or not.
2) Discard the patterns if there is some garbage outputs.
3) Provide a list of fault detection patterns. This enables the efficient use of special programs (Algorithm 3) to correct the faults.

Definition of the Complemented Reversible Circuits Reversible circuits can be measured in terms of quantum realization of that circuit. Quantum realization of a reversible circuit consists of dividing the circuit into several *levels*. Fig. 3 shows the quantum realization of reversible benchmark *3_17tc* circuit. It is possible to reorganize the circuit from higher number of levels to the lower number of levels. When such kind of reorganizes occurs in a reversible circuit, the resulting circuit is called the complemented reversible circuits. Fig. 6 represents the complemented reversible benchmark *3_17tc* circuit.

There is a technique for complementing a reversible circuit of its own [4]. The method in [4] uses redundancy to make a reversible circuit to self-complemented reversible circuit. The proposed fault detection algorithm is based on redundancy for the purpose of fault detection in self-complemented reversible circuits. A general schema of this kind of redundancy is shown in Fig. 5. Most of the occurred faults in a reversible circuit can be detected using the method of duplication. In order to duplicate, we use self-complement feature of some reversible benchmark circuits. Accordingly, we complement the duplicated circuit. In view of the fact, if all the gates of a circuit are self-complemented, the whole circuit is self-complemented. Afterwards, we connect the output of the original circuit to the complemented one.

Algorithm 2: Algorithm for Detecting Faults in a Reversible Circuit (RC)

```
INPUT     : Test Vector $V$ from Algorithm 1
OUTPUT    : Detection of faults in a Reversible Circuit

begin
    $V[]$ = Test Vector Array
    $PI$ = Primary Input
    $PO$ = Primary Output (Input to the Complemented RC)
    $FO$ = Final Output (Output from the Complemented RC)
    $F[]$ = Fault List Array
    $d$ = level of the RC
    for $i \leftarrow 0$ to $n - 1$ do
        $V[] \rightarrow$ Test Vectors from Algorithm 1
    end for
    for $i \leftarrow 0$ to $n - 1$ do
        $F[i] \rightarrow V[i] = PI$
    end for
    for $i \leftarrow 0$ to $d - 1$ do
        Pass $V[i]$ to the RC
        Get $F[i] \rightarrow PO$
    end for
    for $i \leftarrow 0$ to $d - 1$ do
        Pass $F[i]$ to the Complemented RC
        Get $F[i] \rightarrow FO$
    end for
    for $i \leftarrow 0$ to $n - 1$ do
        if $(P[i] = FO)$ then
            No Fault Detected
        end if
        else
            Fault Detected
        end if
    end for
    return $Boolean$ (Detection of Faults)
end
```

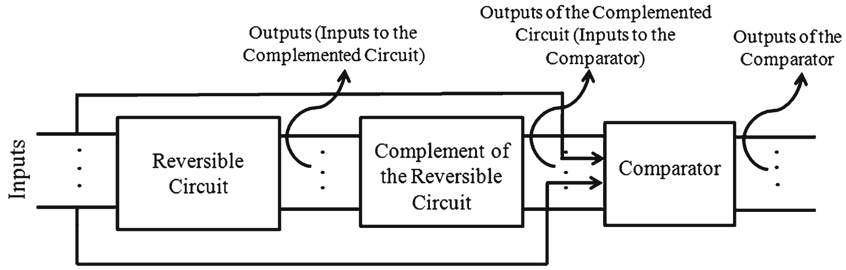
Outputs (Inputs to the Complemented Circuit)
Outputs of the Complemented Circuit (Inputs to the Comparator)
Outputs of the Comparator
Inputs
Reversible Circuit
Complement of the Reversible Circuit
Comparator

Fig. 5 Self-Complemented Fully Redundant Reversible Circuit

The circuit shown in Fig. 5 is fully redundant with two types of outputs: 1) Main outputs of the reversible circuits and 2) Outputs of the complemented reversible circuits. Main outputs of the reversible circuits are the outputs of the first part and outputs of the complemented reversible circuits are the outputs of the redundant part. As two parts of the circuit are complement, it is clear that the values of inputs and outputs of the complemented reversible circuits are the same.

5.1 Contributions of the Proposed Fault Detection Algorithm

In this paper, we consider the detection of single and multiple bit faults. The contributions of the proposed fault detection algorithm are:

- All possible single and multiple bit faults of the faulty circuits can be detected.
- To detect the faults by the minimum time in the literature till now.
- The proposed algorithm can detect bit faults for any reversible circuit.

The proposed fault detection algorithm works with five main steps. In the first step, the generated test vectors *V* using Algorithm 1 are stored as strings. In the second step, a fault list *F* is created which acts as primary input (*PI*). The proposed fault detection algorithm passes *F* to a circuit and checks whether there is any fault or not. The value of *V* is initially assigned to *F*. In the third step, pass the test vectors to the reversible circuit and get primary outputs (*PO*). The primary outputs are then updated in the fault list array (*F[]*) . In the fourth step, pass the updated fault list array to the complemented reversible circuit and get final outputs (*FO*). The fault list array is then updated with the final outputs. In the last step, check the value of *FO* with *PI*. To do this, we use a comparator to check whether the value of *FO* is equal to the value of *PI* or not. The `0′ result of the comparator indicates that there is no change between *FO* and *PI*, as a result no faults are detected. But a nonzero result of the comparator indicates that there is a change between *FO* and *PI* as well as the circuit is faulty. If faults are detected in the circuits, we should make a way to correct the detected faults which we will show in Section 6.

The proposed fault detection method considers the following issues which have been discussed in Example 6:

i. Use the optimum test vectors generated from Algorithm 1 and test a reversible circuit whether there is any fault or not.
ii. Make a fully redundant circuit with the help of main and complemented reversible circuits.
iii. Provide a list of fault detection patterns. This enables the efficient use to correct the faults.

Example 6 A simple reversible circuit *3_17tc* is shown in Fig. 3, in which Algorithm 2 is used to detect the faults. The circuit has three inputs, three outputs, six gates and six levels. Considering Fig. 3 as the main circuit, the circuit shown in Fig. 6 is the complemented *3_17tc* circuit. After connecting these two parts, the whole circuit will be a fully redundant circuit which is shown in Fig. 7. The process of detecting faults is accomplished using a comparator to compare primary inputs and outputs of the complemented reversible circuits. A `0′ in the output of the comparator indicates that there is no fault in the circuit and a nonzero output indicates that the circuit is faulty.

Theorem 2 The proposed fault detection algorithm requires *O(d. 1/N)* complexity of time, where *d* is the number of gates and *N* is the number of inputs of the circuit.

Proof Suppose, the number of detected faults using Algorithm 2 is *n* and the number of gates of the circuit is *d* for any circuit, s.t., $\forall n$ is positive and $n \in N$. Then a recurrence *T(N)* defined by Algorithm 2 is as follows:

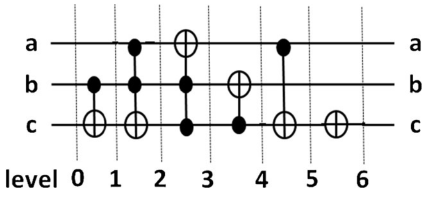

Fig. 6 *Complemented Circuit of 3_17tc*

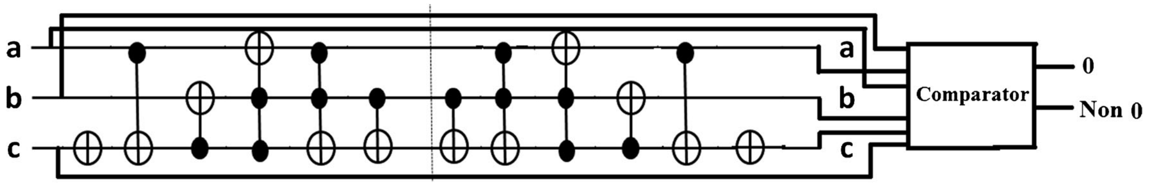

Fig. 7 *Fully Redundant Circuit of 3_17tc*

Table 1 Optimum Test Vector Generation and Fault Detection in the Reversible Benchmark Circuits [22]

Benchmark Circuit	Test Vector	Detected Faults
3_17tc	010	0
	011	1
	100	0
	101	0
	110	0
rd32	010	0
	011	1
	100	0
	101	0
	110	1
4b15g_1	0100	0
	0101	0
	0110	0
	0111	1
	1000	0
	1001	0
	1010	1
	1011	0
	1100	1
	1101	1
	1110	0
hwb4tc	0100	1
	0101	0
	0110	0
	0111	1
	1000	1
	1001	0
	1010	1
	1011	0
	1100	0
	1101	1
	1110	1
rd53rcmg	11,001	1
	11,010	0
	11,011	0
	11,100	1
	11,101	0
	11,110	0

$$T(N) = n.T(N/2) \tag{1}$$

A function is required to compute the complexity of time. Let *f(N)* be a function, where *a, b* ∈ *N* and ∀*(a, b)* > =*1*. Then *f(N)* is calculated as follows:

$$f(N) = O(d^{\cdot}\ log_b a/N){-}\in$$

By considering *T(N)* = *a. T(N/b)* + *f(N)*, we get *a* = *N* and *b* = *N* from Eq. (1), where *f(N)* = *1*.

Since, *f(N)* = *1* = *O(d. log*$_N$*N / N)* - ∈, case 1 of Master Theorem [40] proves that *d. log*$_N$*N / N* = *d. 1/N*, which is *O(d. 1/N)*; where ∈ > 0.

Now, we use the lemma of Master Theorem [40] to get the following:

$$\begin{aligned} \mathrm{T(N)} &= \mathrm{n.T(N/2)+1} \\ &= O(d.log_N N/N) \\ &= O(d.1/N) \end{aligned}$$

So, the claim *T(N)* = *O(d. 1/N)* is correctly proved.

Table 1 summarizes our proposed Algorithms 1 and 2. We show the generated optimum test vectors and detection of faults using generated test vectors for some of the reversible benchmark circuits described in [22]. A 1 in the Detected Faults column of Table 1 indicates that there exists a fault for corresponding test pattern and 0 indicates there is no fault.

Example 7 For computing the running time of the proposed fault detection algorithm, we see that, for 2, 4 and 8 bits, it requires 1, 2 and 3 ms of running time respectively. So, all the above calculations prove that the time complexity of the proposed algorithm is *O(log*$_2$*N)*.

Example 8 For computing the running time of the proposed fault detection algorithm, we see that, for 2, 4 and 8 bits, it requires 1, 2 and 3 ms of running time respectively. So, all the above calculations prove that the time complexity of the proposed algorithm is *O(log*$_2$*N)*.

Lemma 2 Let *n* be the number of inputs, then minimum number of detected faults in the 4-bit with maximal gate count circuit is *log*$_2$*N* + *1.*

Proof According to the specification of Lemma 1, the 4-bit with maximal gate count circuit is tested for *V* patterns, since the circuit has no garbage outputs. Actually, *DF*(Detected Faults) consists of the outcome from *V* of a circuit using Algorithm 2.

A circuit is tested with the help of $g_i = \{g_{i1}, g_{i2}, ..., g_{ip}\}$ and some *V* can detect faults and some can not. This means i^{th} wire is tested for detecting the faults. We must make sure that every gate fixes some *V* i.e., each gate contains at least one control input which is 0 i.e., there should be no gate such that g_i □ *DF*. So, at least one control input of g_i must have 0 value.

When we get the control point is 0, pass the *V* to the circuit without changing its value. The 4-bit with maximal gate count circuit has different $log_2N = log_24 = 2$ types of control inputs i.e., 0 and 1.

So, the resulting number of *DF* be the minimum number of $log_2N + 1$, thus completes the proof.

6 Proposed Fault Correction Algorithm

There are many ways to correct the detected faults in a reversible circuit. The proposed method is based on the existence of implications, i.e., fully invariant relations between pairs of wires in a circuit. An implication from an input pattern to an output pattern indicates that a value assignment at the input wire forces a consistent value assignment at the output wire. It is noted here that consistent value assignment means that for any pattern, there is a corresponding output pattern. The output patterns are also known as the fault list array *F[]* mentioned in the proposed Algorithm 2. Such forced relations provide a source of invariant that can be used to correct the detected faults. Suppose that a fault distorts the output pattern by changing the value of a bit on the implication path between the input pattern and the output pattern. Then, if the input pattern is added as an input to the gates of a reversible circuit driven by the faulty output pattern, the effect of the distorted value can be corrected and the output of the circuit will still be correct. Addition of such patterns, however, is possible only if the function realized is preserved, i.e., if the added pattern is functionally redundant.

6.1 Working Principle of the Proposed Approach

Here we describe the idea of utilizing logic implications to add redundant patterns to the circuit. Let $(I_p,u) \rightarrow (O_p,v)$ denote an implication, i.e., the fact that a value of *u* on the input pattern I_p will cause a value of *v* on the output pattern O_p, where $u,v \in \{0,1\}$. If a detected fault flips the value of any bit on the implication path, then the output will obtain a faulty value. However, if we add the functionally redundant pattern, then the detected fault will be corrected at the implication path before reaching the final output. The addition of the redundant pattern introduces a new source where bit faults may appear, and propagate to the output. Essentially, a detected fault that distorts the value of a bit on an implication path can be corrected by adding a pattern from the input of the implication to the gates driven by the output pattern. When such a pattern is added to a reversible circuit, the operation of the circuit in the presence of a fault is more reliable.

6.2 Identification of Redundant Patterns

Now we will discuss how to find candidate redundant patterns that can be added to a reversible circuit to correct a particular detected bit fault. Functionally redundant patterns are identified when there exists an implication $(I_p,u) \rightarrow (O_p,v)$. Given a specific output pattern and value pair (O_p,v), several methods can be used to find implicating input patterns and values (I_p,u). The method of indirect implication is discussed in the following.

The direct implication identification procedure covers the straightforward case where there is only a single possible value that justifies an unjustified gate. A gate is considered unjustified if the current assignments of its inputs do not justify the output value of the gate. For example, an AND gate that has an output value of 0 with none of its inputs assigned a value of 0 is an unjustified gate. In this case, there exist multiple possible input value combinations for justifying the gate. As a result, the direct implication identification procedure will stop. However, if all justifications of the unjustified gate yield a common implication, then this implication holds true regardless of the actual justification. This type of implication is called indirect implication.

The main idea of the proposed fault correction algorithm is to identify the implications through *Learning*. The implications obtained through *Extended Backward Learning* are a strict subset of the implications obtained using *Recursive Learning*. In general, Recursive Learning [19] with unlimited recursion depth is the most powerful method, because it is able to identify all direct and indirect implications for a given output pattern and value assignment (O_p,v). When some detected faults are corrected using Algorithm 3, the process *Learning* starts and the rest of the faults are automatically corrected.

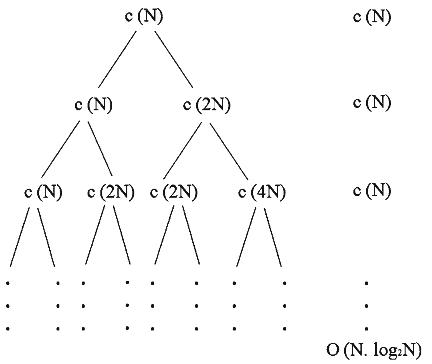

Fig. 8 *A Recursion Tree for Eq. (2)*

Example 9 The circuit *3_17tc* is shown in Fig. 3, in which Algorithm 3 is used to correct the detected faults. The detected faults are imported using Algorithm 2 and number of expected corrected faults is saved to *EMF*. Suppose that the input pattern is 101 and corresponding faulty output pattern is 100. So, by using Algorithm 3 the value of the third bit is flipped on the implication path and the fault is corrected. It is noted that when a large number of corrections is needed, the process *Learning* starts working. When the number of corrected faults is closed to *EMF*, extended backward learning is applied and the process of recursive learning starts and the rest of the faults are corrected automatically.

Logic implications constitute a source of fully invariant that can be exploited to correct the detected bit faults. More specifically, the addition of functionally redundant patterns that realize these logic implications reduces the susceptibility of reversible circuits.

Algorithm 3: Algorithm for the Correction of the Detected Faults in a Reversible Circuit (RC)

```
INPUT     : Detected Faults from Algorithm 2
OUTPUT    : Correction of faults in a Reversible Circuit

begin
    $I_p$ = Input Pattern
    $O_p$ = Output Pattern
    $u$ = A value of the Input Pattern
    $v$ = A value of the Output Pattern
    DF[] = Detected Faults Array
    EMF = Expected Number of Corrected Faults
    for $i \leftarrow 0$ to $n - 1$ do
        DF[] → Detected Faults from Algorithm 2
    end for
    for $i \leftarrow 0$ to $n - 1$ do
        $(I_p, u) => (O_p, v)$
        if (Redundant Pattern is added) then
            Faults will be Corrected
        end if
        if (No. of Corrected Faults is close to EMF) then
            Apply Extended Backward Learning
            Apply Recursive Learning [To correct the faults automatically]
        end if
    end for
end
```

Theorem 3 The proposed fault correction algorithm requires *O(N. log$_2$N)* time, where *N* is the number of inputs.

Proof Suppose that our proposed fault correction algorithm works for all *N*, s.t., ∀*N* ∈ number of corrected faults from any circuit. If we apply our fault correction algorithm ∃*c*, where *c* is a constant, then we can represent the algorithm by the recurrence as follows:

$$T(N) = T(N/3) + T(2N/3) + c.N \tag{2}$$

A recursion tree for Eq. (2) is given in Fig. 8.

Assume that T(N) = O(N. log$_2$N).

For the completion of the proof, we have to show that T(N) < = d. N. log$_2$N, s.t., d ∈ N.

From Eq. (2), we can write

$$\begin{gathered}
T(N) <= T(N/3) + T(2N/3) + c. \\
<= d(N/3).log_2(N/3) + d(2N/3).log_2(2N/3) + c.N \\
= d(N/3).log_2(N) – d(N/3).log_2(3) + d(2N/3).log_2(2N) – d(2N/3).log_2 3 + c.N \\
= d(N/3).log_2(N) – d(N/3).log_2(3) + d(2N/3).log_2(N) – d(2N/3).log_2 3/2 + c.N \\
= d.N.log_2 N – d\{(N/3).log_2 3 + (2N/3).log_2(3/2)\} + c.N \\
= d.N.log_2 N – d\{N.log_2 3 – (2N/3).log_2 2\} + c.N \\
<= d.N.log_2 N, \text{iff} d >= c/(log_2 3) – (2/3)
\end{gathered}$$

This proves that *T(N) < = d. N. log$_2$N.*

Therefore, *T(N) = O(N. log2N)* which completes the proof.

7 Experimental Results

In this Section, we show some comparisons between some of the existing approaches in the literature and our proposed methods described earlier in this paper.

Table 2 Test Vector Generation in the Reversible Circuits

Benchmark Circuit	No. of Inputs, N	No. of gates, *d*	Test Vector ([10])	Test Vector ([24])	Test Vector (Proposed)
3_17tc	3	6	4	2	5
2_4decd1	3	3	4	3	5
ham3tc	3	5	4	2	5
2of5d1s	5	15	5	4	3
rd32	3	4	5	3	5
xor5d1	5	4	5	3	13
4_49tc1	4	16	5	3	11
4b15g_1	4	15	5	3	11
4b15g_2	4	15	5	3	11
5mod5tc	5	17	5	3	3
4b15g_3	4	15	5	4	11
4b15g_4	4	15	5	4	11
rd53rcmg	5	30	5	2	6
4b15g_5	4	15	5	4	11
hwb4tc	4	17	5	3	11

7.1 Comparison among Existing and Proposed Methods of Test Vector Generation

Using our proposed optimum test vector generation algorithm, we have constructed test vectors for detecting faults for some of the benchmark circuits given in [22]. Table 2 summarizes the results of the proposed algorithm for the benchmark circuits. The columns of the table give the circuit's name, number of inputs *N*, number of gates *d* and the size of the test vector generated by the existing and proposed methods.

In Table 2, we have shown that, for most of the benchmark circuits, our proposed method generates optimum test vectors than the existing ones, e.g., for *xor5d1* circuit, our proposed method generates 13 test vectors, whereas the existing methods [10] and [24] generate test vectors 5 and 3 respectively. A comparison of generated test vectors for some benchmark circuits using the existing and our proposed methods is illustrated in Fig. 9.

From Table 3, we can find that, the proposed algorithm is very efficient since the best existing method [10] requires $O(N.\ log_2N)$ execution time, whereas our proposed algorithm requires only $O(log_2N)$, where *N* is the number of inputs. For the larger circuits, the execution time is also very small, e.g., for a 8-input benchmark circuit, the proposed method requires 3 ms, whereas the best known existing method [10] requires 24 ms. So, our proposed algorithm can easily be adapted for much bigger circuits.

7.2 Comparison among Existing and Proposed Methods of Fault Detection

After generating optimum test vectors with lower times, now we have to detect the faults in a circuit. For detecting faults in a circuit, we use our proposed fault detection algorithm. Using our proposed fault detection algorithm, we have detected faults for some of the benchmark circuits given in [22]. Table 4 shows the number of detected faults for the benchmark circuits. The columns of the table give the circuits name, Number of detected faults using existing [11, 23] and proposed methods.

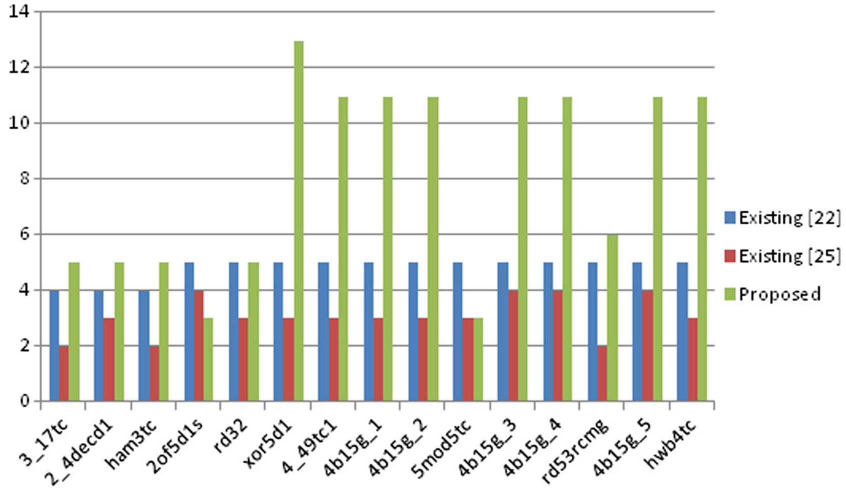

Fig. 9 *Generated Test Vectors for the Benchmark Circuits using Existing and Proposed Methods*

Table 3 Comparison among Existing and Proposed Methods of Test Pattern Generation w.r.t. Time

Method	Required Time
Existing [10]	*$O(N.log_2N)$*
Existing [24]	*$O(2^N)$*
Proposed	*$O(log_2N)$*

N ≡ Number of Inputs

Table 4 Number of Detected Faults in the Reversible Circuits

Benchmark Circuit	No. of Faults [23]	No. of Faults [11]	No. of Faults (Proposed)
3_17tc	1	0	1
2_4decd1	2	0	2
ham3tc	1	1	2
2of5d1s	0	0	0
rd32	0	0	2
xor5d1	4	3	9
4_49tc1	2	1	3
4b15g_1	4	0	4
4b15g_2	2	1	3
5mod5tc	1	0	1
4b15g_3	4	1	4
4b15g_4	3	2	4
rd53rcmg	2	2	2
4b15g_5	2	1	3
hwb4tc	4	2	6

In Table 4, we have shown that, for most of the benchmark circuits, our proposed method detects more faults than the existing ones, e.g., for *xor5d1* circuit, our proposed method detects 9 faults, whereas the existing methods [23] and [11] detect 4 and 3 faults respectively. A comparison of number of detected faults for some benchmark circuits using the existing and our proposed methods is illustrated in Fig. 10.

From Table 5, we can find that, the proposed algorithm is very efficient since the best existing method [23] requires *$O(d.log_2N)$* execution time where d is the number of gates in the circuit, whereas our proposed algorithm requires only *(log_2N)*, where N is the number of inputs. For the larger circuits, the execution time is also very small, e.g., for a 8-input benchmark circuit, the proposed method requires 3 ms, whereas the best known existing method [23] requires *$(3 \times d)$* ms. So, our proposed algorithm can easily be adapted for much bigger circuits.

7.3 Comparison among Existing and Proposed Methods of Fault Correction

After detecting more faults with lower times, now we have to correct the faults of a reversible circuit. For correcting faults in a circuit, we use our proposed fault correction algorithm. Using our proposed fault correction algorithm, we have corrected faults for some of the benchmark circuits given in [22].

From Table 6, we can find that, the proposed algorithm is very efficient since the existing methods [16, 27] requires *$O(N^2)$* execution time, whereas our proposed algorithm requires only *$O(N. log_2N)$*, where N is the number of inputs. For the larger circuits, the execution time is also very small, e.g., for a 8-input benchmark circuit, the proposed method requires 3 ms, whereas the existing methods [16, 27] requires 64 ms. So, our proposed algorithm can easily be adapted for much bigger circuits.

8 Conclusion

In this paper, we briefly explain the difference between offline and online testing. In offline testing, a test vector consisting of inputs identified to be useful in detecting errors is applied to the circuit. This requires that the circuit be taken out of oper-

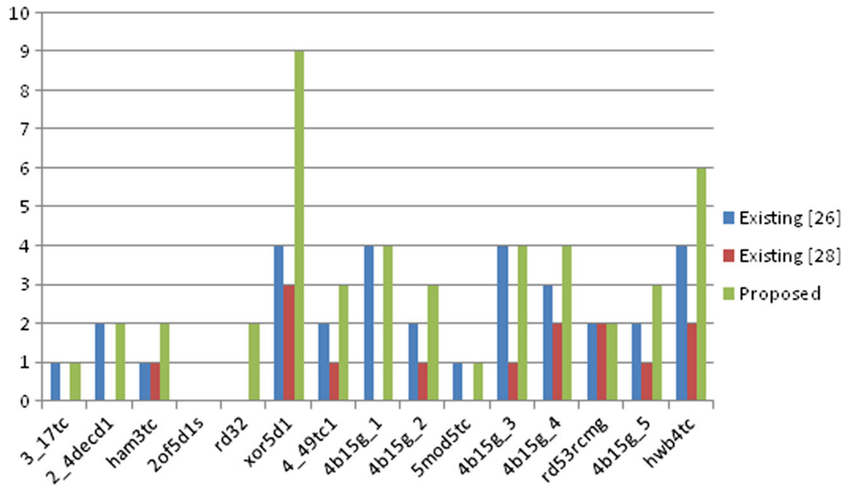

Fig. 10 *Number of Detected Faults for Benchmark Circuits using Existing and Proposed Methods*

Table 5 Existing and Proposed Methods of Fault Detection w.r.t. Time

Method	Required Time
Existing [23]	$O(d. N)$
Existing [11]	$O(N^2)$
Proposed	$O(d. 1/N)$

d ≡ Number of gates; NN ≡ Number of Inputs

Table 6 Existing and Proposed Methods of Fault Correction w.r.t. Time

Method	Required Time
Existing [32]	$O(N^2)$
Existing [15]	$O(N^2)$
Proposed	$O(N. log_2N)$

N ≡ Number of Inputs

ation for some time, and that the outputs resulting from the tests be compared with a set of known correct outputs. In contrast, online testing is carried out while the circuit is being used for normal operations, and additional circuitry is used to identify if a fault is occurred. In this paper, we proposed a method for generating optimum test vectors to detect quantum bit faults in reversible circuits. In our experiments, it is shown that optimum test vectors can detect more faults in a circuit. Simulation results using benchmark circuits show that the proposed algorithm generates optimum test vectors than a traditional ATPG algorithm [2]. It yields very good performance at running time also. The proposed method may assist in the development of physical realizations of quantum gates based on trapped ion technology, spin in semiconductors, etc. [10]. As the generation of optimum test vectors in reversible logic circuits is very important [1], the proposed algorithm is useful for detecting faults in the circuits [15, 26, 32]. An efficient algorithm was proposed to detect the faults in a reversible circuit. For detecting the faults, we used the test vectors generated from the proposed optimum test vector generation algorithm. The proposed fault detection algorithm outperforms the existing ones in terms of time complexity. The proposed algorithm is useful for correcting faults in the circuits [16, 27]. Another efficient algorithm was proposed to correct the detected faults. It is shown in the comparison tables that, the proposed fault correction algorithm corrects more faults than existing ones with lower time complexity.

References

1. Agarwal VD (1981) An information theoretic approach to digital fault testing. IEEE Trans Con Comp, 582–587
2. Allen JS, Biamonte JD, Perkowsky MA (2005) ATPG for Reversible Circuits using Technology Related Fault Models. Proc 7th Int Symp on Representations and Methodology of Future Computing Technologies, RM2005, Tokyo, Japan 2005, 1-8
3. Babu HMH, Islam MR, Chowdhury SMA, Chowdhury AR (2004) Synthesis of Full-Adder Circuit Using Reversible Logic. Proc 17th Int Conf VLSI Des, 757–760
4. Barshan M, Bahramnejad S, Kalantary Z (2011) A Fault Detection Method for Reversible Circuits. J Adv Math Comput Methods 1(1): 15–20
5. Baumann R (2005) Soft Errors in Advanced Computer Systems. IEEE Des and Test of Computers 22(3):258–266
6. Bennett CH (1973) Logical reversibility of computation. IBM J Res Dev 17:525–532
7. Biamonte JD, Allen JS, Lukac M, Perkowsky M (2004) Principles of Quantum Fault Detection. McNair Res Journal, USA, 1–12
8. Biamonte JD, Allen JS, Perkowsky MA (2010) Fault models for quantum mechanical switching networks. J Electron Test Theory Appl 26(5):499–511
9. Biswas AK, Hasan MM, Chowdhury AR, Babu HMH (2008) Efficient Approaches for designing reversible binary coded decimal adders. Microelectron J 39(12):1693–1703
10. Bubna M, Goyal N, Sengupta I (2007) A DFT Methodology for Detecting Bridging Faults in Reversible Logic Circuits. IEEE Region 10 Conf, 1–4
11. Fang-Ying X, Han-Yu C, Wen-jie L, Zhi-qiang L (2008) Fault Detection for Single and Multiple Missing-gate Faults in Reversible Circuits. IEEE World Cong Computational Intelligence, 131–135
12. Feynman R (1985) Quantum Mechnical Computers. Optics News 11:11–20
13. Feynman R (1986) Quantum Mechanical Computers. Found Phys 16(6):507–531
14. Fredkin E, Toffoli T (1982) Conservative Logic. Int J Theor Phys 21:219–253
15. Hayes JP, Polian I, Becker B (2004) Testing for Missing Gate Faults in Reversible Circuits. Proc. 13th Asian Test Symp., Taiwan, 1-6
16. Hoffmann DW, Kropf T (2000) Efficient Design Error Correction of Digital Circuits. Proc Int Conf Computer Des, 465–472
17. Hung W, Song X, Yang G, Yang J, Perkowski M (2006) Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis. IEEE Trans Comput Aided Des Integr Circuits Syst 25(9):1652–1663
18. James RK, Shahana TK, Jacob JP, Sasi S (2007) Fault Tolerant Error Coding and Detection using Reversible Gates. IEEE Region 10 Conf TENCON, 1–4
19. Kunz W, Pradhan DK (1994) Recursive learning: A new implication technique for efficient solutions to CAD-problems: Test, verification and optimization. IEEE Trans Comput Aided Des Integr Circuits Syst 13(9):1149–1158
20. Landauer R (1961) Irreversibility and heat generation in the computing process. IBM J Res Dev 3:183–191
21. Liu B (2010) Error - Detecting/Correcting - Code - Based Self - Checked/Corrected/Timed Circuits. NASA/ESA Conf Adaptive Hardware and Sys, 66–72
22. Maslov D (2017) Reversible Benchmark Circuits. http://www.cs.uvic.ca/dmaslov, Department of Computer Science, University of Victoria
23. Mondal J, Mondal B, Kole DK, Rahaman H, Das DK (2015) Boolean Difference Technique for Detecting All Missing Gate Faults in Reversible Circuits. IEEE 18th Int Symp Des and Diagnostics of Electronic Circuits and Sys, 95–98
24. Nagamani AN, Abhishek B, Agrawal VK (2015) Deterministic approach for Bridging fault detection in Peres-Fredkin and Toffoli based Reversible circuits. IEEE Int Conf Computational Intelligence and Computing Res, 87–92
25. Nagamani AN, Ashwin S, Abhishek B, Agrawal VK (2016) An Exact approach for Complete Test Set Generation of Toffoli-

Fredkin-Peres based Reversible Circuits. J Electron Test Theory Appl 32(2):175–196

26. Naseer R, Draper J (2008) Parallel Double Error Correcting Code Design to Mitigate Multi-Bit Upsets in SRAMs. Solid State Circuits Conf, 222–225
27. Nicolaidis M (2005) Design for Soft Error Mitigation. IEEE Trans Device Mater Reliab 5(3):405–418
28. Nielsen MA, Chuang IL (2000) Quantum Computation and Quantum Information. Cambridge Univ Press, Cambridge
29. Pan WD, Nalasani M (2005) Reversible Logic. IEEE Potentials 24(1):38–41
31. Parhami B (2006) Fault tolerant reversible circuits. Proc 40th Asimolar Conf Sig Sys Comp Pacific Grove, 1726–1729
32. Patel KN, Hayes JP, Markov IL (2003) Fault Testing for Reversible Logic Circuits. Proc. 21st IEEE VLSI Test Symp., USA pp. 410–416
32. Patel KN, Hayes JP, Markov IL (2004) Fault Testing for Reversible Circuits. IEEE Trans Computer Aid Des 23(8):1220–1230
33. Peres A (1985) Reversible logic and quantum computers. Phys Rev A 32:3266–3276
34. Perkowski M, Biamonte J, Lukac M (2005) Test Generation and Fault Localization for Quantum Circuits. Proc 35th ISMVL, Canada, 1-7
35. Polian I, Hayes JP (2010) Advanced Modeling of Faults in Reversible Circuits. Des and Test Symp, St. Petersburg, Russia, 376–381
36. Polian I, Hayes JP, Fiehn T, Becker B (2005) A Family of Logical Fault Models for Reversible Circuits. IEEE Proc 14th Asian Test Symp, 422–427
37. Saligram R (2013) Design and Implementation of Logical Cost Efficient Nanometric Fault Tolerant Reversible BCD Adder. IEEE Annual India Conf 1–6
38. Smoline J, DiVincenzo DP (1996) Five two-qubit gates are sufficient to implement the quantum Fredkin Gate. Phys Rev A 53(4): 2855–2856
39. Veneris A, Chang R, Abadir MS, Amiri M (2004) Fault equivalence and diagnostic test generation using ATPG. IEEE Int Symp Circuits and Systems, Canada, 1-4
40. Verma RM (1994) A General Method and a Master Theorem for Divide-and-Conquer Recurrences with Applications. J Algo 16(1): 67–79
41. Wille R, Drechsler R (2009) BDD-based synthesis of Reversible Logic for large functions. Proc Des Autom Conf., USA, 270–275

Hafiz Md. Hasan Babu was born in Bangladesh. He received the M.Sc. degree in Computer Science and Engineering from the Brno University of Technology, Czech Republic, in 1992 under the Czech Government Scholarship. He was with the Department of Computer Science and Engineering, Khulna University, Bangladesh from 1992 to 2000. In 1995, he was at the Asian Institute of Technology (AIT), Thailand under the DAAD Fellowship from the Federal Republic of Germany. He completed his Ph.D. Degree in Computer Science and Electronics in 2000 with the Japanese Government Scholarship from the Kyushu Institute of Technology, Iizuka, Japan. He was the Founder Chairman and Professor of the Department of Robotics and Mechatronics Engineering, University of Dhaka, Bangladesh. At present, Prof. Dr. Hasan Babu is working as Professor in the Department of Computer Science and Engineering of the University of Dhaka, Dhaka 1000, Bangladesh, where he also served as the Chairman of the same department from 2003 to 2006. Currently, he has been serving as the Pro-Vice Chancellor of the National University of Bangladesh on deputation from the Department of Computer Science and Engineering of the University of Dhaka, Bangladesh. His research interests include logic synthesis, representations of logic functions, reversible computing, DNA computing, quantum computing, etc.

Md. Solaiman Mia has completed his B.Sc and M.Sc from the Department of Computer Science and Engineering, University of Dhaka, Bangladesh. Currently, he is working as a lecturer in the Department of Computer Science and Engineering, Hamdard University Bangladesh. He is interested mostly in digital logic synthesis and design, reversible logic circuit design and quantum computations.

Asish Kumer Biswas has completed his B.Sc and M.Sc from the Department of Computer Science and Engineering of University of Dhaka. Currently, he is pursuing Ph.D. with the Department of Computer Science and Engineering of University of Texas at Arlington, USA. He is interested mostly in digital logic synthesis and design, reversible logic circuit design and quantum computations.