A Genetic Algorithm-Based Heuristic Method for Test Set Generation in Reversible Circuits

A. N. Nagamani, *Member, IEEE*, S. N. Anuktha, N. Nanditha,
and Vinod Kumar Agrawal, *Senior Member, IEEE*

***Abstract*—Low power circuit design has been one of the major growing concerns in integrated circuit technology. Reversible circuit (RC) design is a promising future domain in computing which provides the benefit of less computational power. With the increase in the number of gates and input variables, the circuits become complex and the need for fault testing becomes crucial in ensuring high reliability of their operation. Various fault detection methods based on exhaustive test vector search approaches have been proposed in the literature. With increase in circuit complexity, a faster test generation method for providing optimal coverage becomes desirable. In this paper, a genetic algorithm-based heuristic test set generation method for fault detection in RCs is proposed which avoids the need for an exhaustive search. Two approaches, one involving random search and the other, involving directed search have been proposed and validated on benchmark circuits considering missing-gate fault (complete and partial), bridging fault and stuck-at fault with optimum coverage and reduced computational efforts.**

***Index Terms*—ATPG, computation time, fault coverage, genetic algorithm (GA), reversible logic.**

I. Introduction

With the rising demands for quantum computing methods for their promising low power dissipation properties and higher data representation and retention capabilities, the field has witnessed rapid advancements. Landauer's [1] principle states that logical reversibility is associated with physical reversibility which serves the purpose of minimal heat generation per machine cycle. The device is said to be logically irreversible if its output is not uniquely defined by the inputs. A logic circuit is reversible if it computes bijective (one-to-one and onto) logic functions. No information loss occurs during the entire operation, thus lowering power consumption to a large extent. Bennett [2] showed that, if computations are carried out irreversibly, $kT\ln 2$ (where k is the Boltzmann constant—1.3806×10^{-23} $m^2kg^{-2}K^{-1}$ and T is the absolute temperature at which computation takes place) power dissipation would occur for each bit of information erasure or throwaway. Hence, lossless computation must be reversible. Reversible circuits (RCs) used with newer technologies such as quantum computing [3], optical computing [4], quantum cellular automata (QCA) [5], trapped-ion technology [6], and adiabatic CMOS [7] offer reduction in power dissipation. Different types of reversible gates including but not limited to Toffoli [8], Fredkin and Toffoli [9], and Peres [10] have been logically designed and many reversible logic circuits are being built with the same. As the versatility of RCs increase with its capability to handle a large number of gates and its bits storage capacity, the vulnerability of the circuits to potential faults increases and thus the requirement for a testing system to detect faults in such circuits becomes very crucial to ensure the correctness of the implemented logic circuit.

References [11] and [12] have proposed reversible testing algorithms using SAT and authors in [13] propose an exact method of testing. As described in these papers, one common approach is to test the circuit using the entire set or at least a majority of the 2^n test vectors (n being the number of lines in the RC) which makes the algorithm exponentially complex for $n\geq15$. Since the test generation process is an NP-hard problem and exponentially complex, an exhaustive search approach quickly becomes impractical as the search space grows colossal in size. GA thus provides a heuristic approach for arriving at the optimum solution by attempting to find better solutions in each iteration until a threshold criterion is met. GA-based algorithms have been extensively studied and found to be very useful in solving NP-hard problems [14], where the problems cannot be characterized by a polynomial expression. An improvisation over GA by taking into account the quality of individuals (test vectors) selected in each generation—an adaptive algorithm [15] can be developed to obtain an optimum solution in a reasonable amount of time.

In this paper, an alternate solution is proposed to the existing testing methods which is efficient in terms of time, space and effort required to obtain a test set, and which can detect a large number of faults for small ($n<8$), medium ($8<n<30$), and large ($n>30$) circuits, with ranges as found in [16]. Quantum circuits implemented on QCA tend to have defects such as cell displacement fault and cell misalignment faults which can be tested with stuck-at fault (SaF) models [17]. Feinstein *et al.* [18] discussed the applicability of various register transfer level fault models such as single and multiple SaFs that deal with quantum gate interconnection which can be

Manuscript received August 19, 2016; revised November 23, 2016, January 4, 2017, and March 10, 2017; accepted March 29, 2017. Date of publication April 19, 2017; date of current version January 19, 2018. This paper was recommended by Associate Editor L.-C. Wang. *(Corresponding author: A. N. Nagamani.)*

A. N. Nagamani, S. N. Anuktha, and N. Nanditha are with the Department of Electronics and Communication Engineering, PES Institute of Technology, Bengaluru 560098, India (e-mail: nagamani@pes.edu).

V. K. Agrawal is with the Department of Information Science and Engineering, PES Institute of Technology, Bengaluru 560098, India.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2695881

readily adapted to quantum circuits. According to [7] and [19], classical fault models—SaFs, delay faults, missing-gate faults, wrong gate faults, and bridging faults (BFs)—need to be considered for RCs implemented as information RCs and, missing-gate faults and wrong gate faults for RCs implemented as energy RCs.

The proposed work focuses on the set of all missing-gate faults which can occur in both information and energy RCs, SaFs and bridging gate faults which find their relevance in information RCs only. Using GA for fault detection eliminates the requirement of testing with all the 2^n vectors thereby reducing the overall computation time. The proposed GA-based testing with a random search approach and a directed search approach have been validated on small, medium, and large benchmark circuits from the libraries—negative control Toffoli, multiple control Toffoli (MCT), multiple control Fredkin (MCF), and Peres (P) as referred in [20] and [21]. The algorithm has been coded using C language and applied to circuits in .tfc and .real formats on a computer with an Intel i5 processor operating on 4-GB RAM at 1.7-GHz clock speed. The major contributions of this paper are as follows.

1) A heuristic GA-based strategy using random search method for obtaining optimum fault coverage for RCs designed using Toffoli, Fredkin, and Peres gates for faults- single stuck-at, bridging and set of all missing-gate faults [partial missing-gate fault (PMGF) and complete missing-gate fault (CMGF)].
2) Directed search approach, an improvisation over random search approach is proposed which reduces the computation time significantly for k-CNOT circuits.
3) A single test set to detect a combination of faults for the set of all missing-gate faults.

The rest of this paper is organized as follows. Section II briefs about existing RC testing algorithms. In Section III, various fault models and their detection methodologies relevant for RCs are discussed and a brief on GA is given. Section IV presents the proposed algorithm with an illustrative example. Section V discusses the results obtained and compares it with recent work in this domain. Section VI concludes the work and scope for future work follows.

II. Literature Review on Reversible Circuits Testing

Various fault detection methodologies are described in the literature for the testing of RCs. There are two major classifications of testing, namely, online and offline testing. Online testing is a process that can be carried out during normal operation whereas, offline testing is performed on the circuit in the test mode. Various online testing methods applied for RCs using dual-rail coding, NAND blocks, and design for testability methods are discussed in [22]–[26].

An offline testing methodology using integer linear programming is proposed in [27]. Various fault models affecting RCs have been considered and tested for in the literature. A detection methodology for missing control fault model is proposed in [28], which includes several subsets of this model—single missing-gate fault also termed as CMGF, multiple missing-gate fault (MMGF), repeated gate fault (RGF), and PMGF—emphasizing that, modeling of RC faults using only single SaF would not suffice. Lukac *et al.* [19] stated that only missing-gate faults are possible fault models to be considered in quantum RCs. Detection of MMGFs and repeated missing-gate faults have been carried out in [29]–[32], and a test set for combined fault models has been derived. Work in [33] indicates that cross point models are better suited for RCs than SaF models. BFs are more frequent as compared to SaF and missing-gate fault and are complicated to detect. A universal test set for BFs has been proposed in [34]. Further, an exact approach for BF detection with a minimal test set is applied to RCs in [35]. A Boolean satisfiability-based fault detection for missing-gate faults is dealt with in [11]. Work in [36] proposes a method for the fault detection of additional control fault and RGF. Recent work in [37] proposes that symmetric functions with only PMGF can be detected with a maximum of three vectors (for any n).

Genetic algorithm (GA) has been explored in literature for reversible and irreversible circuit synthesis and optimization. A design based on GA for space exploration framework for parameterized system-on-chip platforms is proposed in [38]. Fault detection in irreversible circuits using GA has been proposed in [39] for the testing of single event upsets in SRAM-FPGA, in [40] for testing large synchronous circuits and Saab *et al.* [41] employed GA to obtain an automatic test vector cultivation for combinational and sequential very large-scale integration circuits. However, GA for fault detection and test set generation in RCs has not yet been explored. With faults in quantum circuits following a probabilistic distribution, stochastic methods for detection and diagnosis would be very effective [42]. GA being a heuristic approach can therefore be expected to give favorable results for testing RCs which find their applicability in quantum computing.

III. Preliminaries

A. Reversible Logic, Fault Models, and Detection Methodology

RCs, unlike irreversible circuits have the property of input retrieval through output as every input combination has a unique output combination. Standard reversible gates used to build RCs are shown in Fig. 1, with their logical expressions in Table I. Quantum cost (QC), gate count (GC), ancilla inputs, garbage outputs (GOs), and delay ($\triangle$) govern the performance of these circuits. Toffoli and Feynman gates come under the class of k-CNOT gates with k values 3 and 2, respectively. The k-CNOT gate is a k-input, k-output reversible gate which transforms $[V_1, V_2, \ldots, V_{k-1}, V_k]$ to $[V_1, V_2, \ldots, V_{k-1}, (V_1 \cdot V_2 \cdot \ldots \cdot V_{k-1}) \oplus V_k]$. The functionality of a Peres gate is the same as that of a cascade of Toffoli and Feynman gates. Fredkin gate functions as a controlled swap with input A being the control. A class of Fredkin gates with multiple controls are classified under MCF.

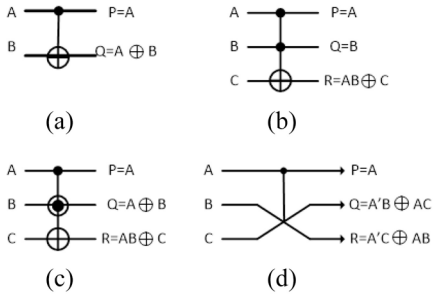

Fig. 1. (a) Feynman (F). (b) Toffoli (T). (c) Peres (P). (d) Fredkin (FR).

TABLE I
REVERSIBLE GATES AND PARAMETERS

Gate	function	QC
Feynman(F)	$F(A,B) = (A, A \oplus B)$	1
Toffoli(T)	$F(A,B,C) = (A, B, AB \oplus C)$	5
Peres (P)	$F(A,B,C) = (A, A \oplus B, AB \oplus C)$	4
Fredkin (FR)	$F(A,B,C) = (A, AB' + AC, AB + A'C)$	5

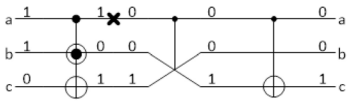

Fig. 2. Illustration of stuck-at 0 at the input of level 1.

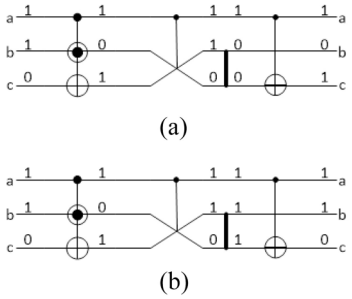

Fig. 3. Illustration of "b" (a) AND bridged with "c" at the input of level 2 and (b) OR bridged with "c" at the input of level 2.

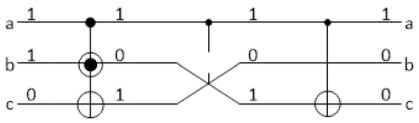

Fig. 4. Illustration of PMGF with a missing control for Fredkin.

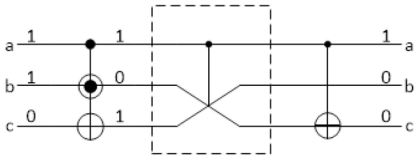

Fig. 5. Illustration of CMGF with a missing Fredkin gate.

Similar to irreversible logic circuits, RCs have a set of fault models described to categorize the various faults that can be observed in them [43]. The ATPG algorithm presented in Section IV is applied on RCs for the detection of faults described here by inducing them at each potential fault location. For every fault induced, the faulty output of the circuit is compared with the fault-free output *R* for a test vector chosen as input. If the two outputs differ, the vector is said to detect the fault and variable *detected* is incremented by 1 from an initial value of 0.

1) Stuck-at Fault, Fig. 2: This fault model describes a line in the circuit at any level, stuck-at a particular signal value (0 or 1), denoted as *x* in Fig. 2, and does not allow any other signal value during circuit operation to change it. The majority of the faults in circuits can be mapped to this type of fault. Such a fault usually occurs during the manufacturing process. Faults of this kind are induced by forcing the input lines to 0 and 1, one at a time for all the levels of the RC with outputs computed in each case which is then compared with *R*, the fault-free output for the same test vector. *detected* is incremented in case of a mismatch. The number of SaFs possible in an *n* input RC with *N* gates are

$$\text{CumulatedFault} = 2.n.(N+1).$$

2) Bridging Fault, Fig. 3: When the signal value on a line is influenced by the signal value on one of the other lines in the circuit, the line under consideration is said to be bridged. Depending on the way in which a line influences the signal value on the bridged line, these faults are classified into two types.

1) *AND Bridging:* If line "a" is AND bridged with line "c," upon induction of the fault, a and c get the logical value "a AND c."
2) *OR Bridging:* If line a is OR bridged with line c, upon induction of the fault, a and c get the logical value "a OR c."

BF is introduced by picking two lines at a time from the *n* lines of the RC. The inputs to these lines are AND*ed* and the common output is forced through the lines. Every possible combination of two lines is made starting from the inputs to the first gate to the output lines after the last gate. In the case of a mismatch, *detected* is incremented. The entire process is repeated by OR*ing* the input lines. Total number of such intralevel faults are

$$\text{CumulatedFault} = \binom{n}{2}.(N+1).2.$$

3) Missing-Gate Fault: Due to manufacturing defects, it might be possible for a gate in the circuit to be skipped or one or many of the control lines to a gate in the circuit go amiss; faults of this kind are labeled missing-gate faults. Two types of missing-gate faults are considered for testing in this paper.

1) *PMGF, Fig. 4:* One or more of the control lines in a gate of an RC are missing giving rise to PMGF. The *CumulatedFault* with one missing control is the sum total of all controls present in the gates of the RC. In order to calculate PMGF, the output *R* is calculated for a test vector. The same test vector is then applied to the RC with controls skipped one at a time. The resulting output is compared with *R* and *detected* is incremented in case of a mismatch.
2) *CMGF, Fig. 5:* An entire level or gate is skipped from the circuit (reducing the GC by 1) constituting a CMGF.

A circuit can have as many CMGF as the total number of gates in it. In this case, *CumulatedFault* is N. *detected* for CMGF is calculated as in PMGF, with a gate skipped one at a time instead of the controls.

The same algorithm can also be extended to RGFs and additional control faults. After obtaining *detected* the fault coverage is calculated as

$$\text{FaultCoverage} = \frac{\text{detected}}{\text{CumulatedFault}} \times 100. \quad (1)$$

B. Genetic Algorithm

GA is a popular evolutionary algorithm developed based on the observation of the sustenance of genes from a population of living organisms. The odds of the survival of the genes depend on their fitness values. GA creates successive generations of individuals by applying simple reproduction operations. Typically, parents are chosen to mate with probability proportional to their fitness called proportional selection. This process of selection and reproduction continues until a maximum fitness is obtained. In circuit testing, the most commonly used operators include bit-string crossover, where, off-springs are formed by swapping a subsequence between the two parent bit strings, and bit-flipping mutation in which a single bit in the string is flipped to form a new offspring string [44]. Mutation helps in faster convergence and its probability of application decreases exponentially with generations [45] and [46].

Fault detection in RCs is easy as they are bijective functions with higher controllability and observability [47]. This property enables detection of maximum faults with very few undetectable faults as most of the faults can be activated through controllable inputs and observed at primary outputs. At the same time, to ensure the property of reversibility, the number of input lines should be made equal to the number of output lines and with an increase in n, the number of potential fault sites increases drastically. An efficient ATPG must converge fast with maximum fault coverage consuming less memory space. ATPG using an exhaustive search as n increases requires more effort in terms of computation time and memory to exercise the complete circuit for minimum test set generation.

GA, unlike other random sampling techniques, steers the search toward more prospective regions due to the combination of reproduction and mutation as directed by the fitness function. While gradient descent techniques can at best arrive at a local optimum, GA can take values from various regions in the spread out curve of the function and hence can arrive at a far superior solution.

In fault detection using GA, test vectors are selected in a way so as to minimize the cost associated with it and previously selected vectors are also taken into consideration to select the filial generation. This ensures that the cost associated with the test set decreases with each generation until no further reduction is possible. This way, an optimum solution can be arrived at without sifting through the entire search space making GA the ideal solution to such problems which cannot be solved using other approaches.

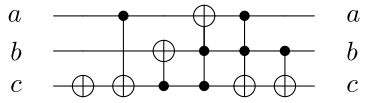

Fig. 6. 3_17*tc* circuit.

IV. Proposed ATPG Algorithms

A. GA Incorporating Random Search

In this section, GA along with its variant where the algorithm is directed by selecting the initial vector population for fault detection is presented. Algorithm 1 describes the basic structure of the GA for fault detection, incorporating a random search approach to produce a test set for a given RC with n input lines and N gates. *Threshold* indicates the desired percentage of fault coverage with the maximum being 100%. The next section expands the steps with the help of an example circuit.

The path taken by the GA is determined by the initial set of vectors chosen. Random selection does not always guarantee maximum coverage for every test run. It might take many iterations to yield optimum test set if the initially selected vectors are weak, which would lead to higher computation time or might even require early termination before optimum coverage has been reached. Thus, by directing the search, faster convergence would be highly likely, especially for MCT-based RCs with a large number of input lines.

B. Illustration of Proposed Algorithm Using Random Search for Complete-Missing-Gate Fault Detection

Consider the benchmark circuit 3_17*tc* as shown in Fig. 6 as the input for the proposed algorithm to find a test set for optimum coverage of CMGF. The Illustration 1 shows the detailed flow of Algorithm 1 for benchmark circuit 3_17*tc*. In this algorithm, gates are also referred to as levels and V_i in a set represents the ith element of the set.

Illustration 1 Illustration for GA Using Random Search for the Circuit 3_17*tc*

INPUT: RC with 'n' lines and 'N' gates; *Threshold*
OUTPUT: The test set and fault coverage%

STAGE I: Input Parameter Extraction

Step 1: Extract the parameters n and N from the RC comprising of n lines $B_0, B_1, ..., B_{n-1}$ and N levels $G_0, G_1, ..., G_{N-1}$. Set fault-type and threshold value where 100 implies maximum fault coverage. Compute *max*—the maximum number of input test vectors for the RC.

$$n = 3 \quad N = 6 \quad max = 2^n = 8 \quad Threshold = 100$$
$$Input\_Fault\_Type = complete\_missing\_gate$$

Step 2: Initialize *generation*, which indicates the current iteration, as 0.

STAGE II: Test Vector Selection

Step 3: Randomly select $n-$which is the upper limit on the number of test vectors for *InitPop*−vectors from 0 to $(max-1)$-(rand(0,*max*)), both inclusive ensuring that none of the selected vectors repeat. *InitPop* is a set containing all the initially selected n vectors. *FinPop* has elements copied from

Algorithm 1 GA for Fault Detection in an RC

```
1: INPUT (RC, Threshold, Input_Fault_Type)
2: OUTPUT (fault coverage%, test set)
3: use (RC) get (n, N)
4: procedure SELECTION(n)                    ▷ To get initial vector set
5:     max ← 2^n
6:     for i ← 1, n do
7:         InitPop(V_i) ← rand(0,max)
8:     end for
9: end procedure
10: procedure FITNESS(InitPop)
11:     for i ← 1, n do
12:         use(Input_Fault_Type, InitPop(V_i))
13:         get(FaultCoverage,DetectedFaults)
14:         FitnessFunction(V_i) ← FaultCoverage
15:         if FitnessFunction(V_i) ≥ Threshold then
16:             Display Output, Exit
17:         end if
18:     end for
19: end procedure
20: procedure MULTIPLY(InitPop,FitnessFunction)
21:     for i ← 1, n do
22:         Parent(V_i1, V_i2) ←Roulette(InitPop,FitnessFunction)
23:         ChildPopulation(V_i) ←CrossOver(Parent(V_i1, V_i2))
24:         ChildPopulation(V_i) ←Mutate(Child(V_i)Population)
25:         use Fitness(ChildPopulation(V_i))
26:         get FitnessFunction(V_i+n), DetectedFaults
27:     end for
28: end procedure
29: FinPop ← InitPop ⋃ ChildPopulation
30: FinFitness ← FitnessFunction
31: ArrangeDescending(FinFitness,FinPop)
32: procedure COMBINE(FinPop,DetectedFaults)
33:     for i ← [n/2], |FinPop| do
34:         high = 0
35:         repeat
36:             Combination_i(.) ←Combinations(i,FinPop)
    ▷ combination taking i vectors
37:             fitness = Fitness(Combination(.))
38:             if fitness ≥ Threshold then
39:                 Display Output, Exit
40:             end if
41:             if high < fitness then
42:                 high = fitness
43:             end if
44:         until high > fitness for |FinPop| iterations
45:     end for
46:     InitPop ← MaxFitness(Combination(.))
47: end procedure
48: Goto line 10
```

InitPop.

$InitPop = \{4, 6, 1\}$

STAGE III: Fitness Function Computation

Step 4: Represent test vector in binary form, as in Fig. 7. Compute *FaultCoverage* for CMGF as explained in Section III-A. Here, test vector 4-[1 0 0], the first element in *InitPop* is selected. The fault-free output R and the faulty outputs after inducing the fault at each level are obtained.

$$R = 100 \quad FaultyOutput = \{010, 010, 100, 100, 100, 100\}$$

Step 5: Obtain *DetectedFault* matrix.

if($FaultyOutput(V_i)$!= R)

set $DetectedFault_{[vector]}(V_i)$ to indicate detection

else

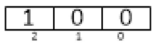

Fig. 7. Test vector as a chromosome.

reset $DetectedFault_{[vector]}(V_i)$

Once *FaultyOutput* has been completely traversed, count the number of 1's in the set $DetectedFault_{[vector]}$ and store it in *detected*. Find *FaultCoverage* using Equation(1).

if($FitnessFunction(V_i) \geq Threshold$)

Goto Step 18.

$$DetectedFault_{[100]} = \{1, 1, 0, 0, 0, 0\}$$
$$CumulatedFault = n, \text{ see Section. III-A}$$
$$FaultCoverage = 33.33\%$$
$$FitnessFunction(V_i) = FaultCoverage = 33.33$$

Step 6: Repeat steps 4 and 5 for all the vectors in *InitPop*. Copy members of *FitnessFunction* into *FinFitness*.

$$DetectedFault_{[100]} = \{1, 1, 0, 0, 0, 0\}$$
$$DetectedFault_{[110]} = \{1, 1, 0, 0, 1, 1\}$$
$$DetectedFault_{[001]} = \{1, 0, 0, 0, 0, 0\}$$
$$FitnessFunction = \{33.33, 66.67, 16.67\}$$
$$FinFitness = \{33.33, 66.67, 16.67\}$$

STAGE IV: Roulette Wheel

Step 7: Select pairs of vectors from *InitPop*. The process here follows the usual flow where vectors after being conformed onto a pie chart are picked two at a time based on their fitness functions. Select n such pairs of parent.

$Parent = \{(100, 110), (110, 001), (110, 100)\}$

STAGE V: Reproduction And Cross-Over

Step 8: A pair of parents is chosen and a random cross-over point is selected in the bit sequence. For 3_17*tc*, [1 0 0] and [1 1 0] are chosen as parents.

$CrossOverPoint = rand(0, n) = 1$

The child is generated taking bits before *CrossOverPoint* from the first parent- '1' from [1 0 0]- and the remaining from the second parent- '1 0' from [1 1 0]- to yield [1 1 0] as the offspring. Continue until n daughters have been generated.

$ChildPopulation = \{110, 111, 110\}$

STAGE VI: Mutation

Step 9: In order to carry out mutation, a number is randomly selected from 0 and $(100^{generation+1} - 1)$, both inclusive. If this number is less than 10, then the test vector undergoes mutation where a random bit is selected and flipped. By doing this, the probability of mutation decreases in exponents of 10 for every *generation*. The genes after mutation are:

$ChildPopulation = \{110, 111, 100\}$

Step 10: Repeated vectors are discarded and fitness function for the non-repeated ones is computed. The new members are added into the set *FinPop* with their fitness values in *FinFitness* along with faults detected by them in *DetectedFault*. The fitness values of the progeny are checked to see if *Threshold* has been met.

$$FinPop = \{100, 110, 001, 111\}$$
$$FinFitness = \{33.33, 66.67, 16.67, 50\}$$
$$DetectedFault_{[111]} = \{1, 1, 1, 0, 0, 0\}$$

STAGE VII: Test Population Generation

Step 11: Vectors of *FinPop* are arranged in descending order based on their fitness values from *FinFitness*. Variable m is defined as shown.

$$FinPop = \{110, 111, 100, 001\}$$
$$FinFitness = \{66.67, 50, 33.33, 16.67\}\ m = |FinPop| = 4$$

STAGE VIII: Minimal Test Set Generation From Test Population

Step 12: if($\frac{n}{2} < 2$), $L = 2$, else, $L = [\frac{n}{2}]$, rounded to the closest integral value. Combinations of L vectors from *FinPop* are made.

$$Combination_{21} = \{110, 111\}$$
$$Combination_{22} = \{110, 100\}$$
$$Combination_{23} = \{110, 001\}$$
$$Combination_{24} = \{111, 100\}$$
$$Combination_{25} = \{111, 001\}$$
$$Combination_{26} = \{100, 001\}$$

Step 13: Set *upperBound* as m. Reset *bestCoverage* which is the best coverage that the algorithm can yield for the current test run.

Step 14: Initialize x and i as 0. Reset *high* which gives the best fitness value for the current combination. Update *DetectedFault* and *detected* for each of the $Combination_{Li}$ and compute *Fitness*.

if($Fitness \geq Threshold$)

$FinalTestSet = Combination_{Li}$, goto Step 18.

if($Fitness > high$)

high=Fitness

else, Increment x

if($bestCoverage < high$)

bestCoverage = *high*, *FinalTestSet*=$Combination_{Li}$

if($x = upperBound$)

Goto Step 15

Increment i and repeat Step 14 until all combinations of vectors are made or if x hits *upperBound*. For 3_17*tc* with combinations of 2 vectors,

$Fitness = \{83.33, 66.67, 66.67, 50\}$

$high = 83.33 \qquad x = 4$

Step 15: if($L = n$), Goto Step 16, else, Increment L and repeat Step 14.

$$Combination_{31} = \{110, 111, 100\}$$
$$Combination_{32} = \{110, 111, 001\}$$
$$Combination_{33} = \{110, 100, 001\}$$
$$Combination_{34} = \{111, 100, 001\}$$
$$Fitness = \{83.3, 83.33, 66.67, 50\}$$
$$high = 83.33\% \qquad bestCoverage = 83.33\%$$

STAGE IX: Initial Population for filial generation

Step 16: if($|FinalTestSet| \leq n - 1$)

Copy all vectors from *FinalTestSet* into *InitPop* and fill up the remaining slots upto n using random selection method described in Step 3.

else, feed $(n - 1)$ vectors from *FinalTestSet* into *InitPop* and select 1 vector randomly using random selection method described in Step 3 and add it to *InitPop*.

$InitPop = \{110, 111, 101\}$

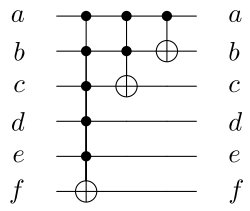

Fig. 8. Three level RC with multiple control k-CNOT gates.

$FinPop = \{110, 111, 101\}$

Vector [1 0 1] is randomly generated.

generation is incremented.

Step 17: Goto Step 4 and repeat the process for the next generation. The values obtained are:

$ChildPopulation = \{101, 101, 100\}$

$FinPop = \{101, 110, 111, 100\}$

$FinFitness = \{83.33, 66.67, 50, 33.33\}$

For the combination, $Combination_{21} = \{101, 110\}$ the fitness is 100. Since *Threshold* is met, goto Step 18.

$DetectedFault_{[110]||[101]} = \{1, 1, 1, 1, 1, 1\}$

STAGE X: Optimum Test Set

Step 18: Display the test set and fault coverage.

$FinalTestSet = \{101, 110\}$, $bestCoverage = 100$, Exit.

Since this approach is heuristic, maximum coverage for every test run cannot be expected. In case the algorithm runs indefinitely with no change in *bestCoverage*, the process is exited prematurely and restarted.

C. GA Incorporating Directed Search

In GA with directed search, the initial population is selected based on the configuration of the first gate in the RC. Values for *InitPop* are selected such that the output of the target is 1, which is ensured by forcing opposite values to the AND of control inputs and the target. Directed search makes use of the fact that by assigning opposite values to the control and target of the first gate, maximum faults can be detected. In case of Toffoli, for instance, whose function is given as (A, B, C) = (A, B, A·B⊕C), line C value toggles if A·B ≠ C. When A·B = 1, and C is 0, faults: Sa0-A, Sa0-B, Sa1-C, CMGF, PMGF-A, PMGF-B, BF-A·C, BF-B·C, BF-A+C and BF-B+C, a total of ten faults can be detected. Whereas, with test vector [0 1 0], seven faults and with [1 1 1], six faults can be detected. By selecting a test vector which can detect a majority of faults, probability of fault propagation increases, and with it, faster convergence of the algorithm is possible. Test set generation for GA incorporating directed search, is explained in Algorithm 2 for k-CNOT circuits. The example circuit as shown in Fig. 8 is used to explain the method. It consists of three gates with the first gate comprising of five controls and a target f.

The values for the *InitPop* are selected from $\{00000, 00001, 00010, \ldots, 11110\}$. The target is always opposite in value to the AND*ed* output of all the controls thereby, ensuring target output as 1 [32]. A vector with all bits set as 1 or 0 detect a large number of faults

Algorithm 2 Directed Search for k-CNOT Circuits

1: **INPUT** (RC)
2: **OUTPUT** (*InitPop*)
3: **use**(RC)**get**(*controls*-$\{B_{c0}, B_{c1}...B_{ck}\}$,*target* $-\{B_t\}$ of first gate with $B_0...B_{n-1}$ as input lines;n)
4: **procedure** SELECTION($n, B_t, B_{c0}, B_{c1}, ..., B_{ck}$)
5: m$\leftarrow$1
6: **if** $|controls| = \emptyset$ **then**
7: Goto line 30
8: **end if**
9: $v \leftarrow |target| + |controls|$
10: $B_t \leftarrow 0$
11: $B_{c0}, B_{c1}, ..., B_{ck} \leftarrow$11...1 ▷ set all control bits
12: **for** $i \leftarrow 0, n-1$ **do**
13: **if** $B_i \notin target$ **and** $B_i \notin controls$ **then**
14: $B_i \leftarrow$ **rand**(0,2) ▷ random allocation of 0/1 for remaining input lines
15: **end if**
16: **end for**
17: $InitPop(V_m) \leftarrow$**Decimal**($B_{n-1}B_{n-2}...B_0$)
18: increment m
19: $B_t \leftarrow 1$
20: $var \leftarrow$00...0 ▷ *var*, a (k+1) bit binary variable is reset
21: **for** $j \leftarrow 1, 2^{v-1} - 1$ **do**
22: **if** $m \leq n-2$ **then**
23: $B_{c0}B_{c1}...B_{ck} \leftarrow var$
24: increment *var*
25: repeat lines 12$-$18
26: **else**
27: Exit **for**
28: **end if**
29: **end for**
30: **for** $m \leftarrow m, n-2$ **do**
31: $InitPop(V_m) \leftarrow$ **rand**(0,max)
32: **end for**
33: $InitPop(V_m) \leftarrow$**Decimal**(111...11)
34: increment m
35: $InitPop(V_m) \leftarrow$**Decimal**(000...00)
36: **end procedure**

especially SaF in most circuits [48]. Hence, $InitPop = \{111110, 000001, 000011, 000101, 000000, 111111\}$.

D. Illustration of Proposed Algorithm Using GA With Directed Search for Complete Missing-Gate Fault Detection

The fault coverage and the test set for the example circuit in Fig. 6 is obtained by substituting directed search for random search in Illustration 2.

Illustration 2 Illustration for GA Using Directed Search for the RC 3_17*tc*

INPUT: RC with 'n' lines and 'N' gates; *Threshold*
OUTPUT: The test set and fault coverage

STAGE I: Input Parameter Extraction

Extracting and initializing the appropriate values.

$n = 3 \quad N = 6 \quad Threshold = 100$
$Input\_Fault\_Type = complete\_missing\_gate$

STAGE II: Test Vector Selection

Using the method explained in Algorithm 2, the initial population is selected.

$$InitPop = \{5, 7, 0\} \quad FinPop = \{5, 7, 0\}$$

STAGE III: Fitness Function Computation

Fitness function for the selected vectors and the *DetectedFaults* are determined.

$$DetectedFault_{[101]} = \{1, 1, 1, 1, 0, 1\}$$
$$DetectedFault_{[111]} = \{1, 1, 1, 0, 0, 0\}$$
$$DetectedFault_{[000]} = \{1, 0, 1, 1, 1, 1\}$$
$$FitnessFunction = \{83.33, 50, 83.33\}$$

STAGE IV: Roulette Wheel

Parent population is arrived at using roulette wheel.

$$Parent = \{(000, 111), (000, 101), (101, 111)\}$$

STAGE V: Reproduction And Cross-Over

The parent chromosomes undergo reproduction to produce offsprings.

$$ChildPopulation = \{000, 000, 101\}$$

STAGE VI: Mutation

Mutation is carried out and *FinPop* is arrived at. For 3_17*tc*, mutation did not occur for any of the vectors.

$$FinPop = \{101, 111, 000\} \; FinFitness = \{83.33, 50, 83.33\}$$

Since fitness value did not meet *Threshold*, move to the next stage.

STAGE VII: Test Population Generation

Vectors of *FinPop* are arranged in descending order based on their corresponding fitness values in *FinFitness*.

$$FinPop = \{101, 000, 111\} \; FinFitness = \{83.33, 83.33, 50\}$$
$$m = |FinPop| = 3$$

STAGE VIII: Minimal Test Set Production From Test Population

Combinations of vectors are made and checked for fitness function.
$Combination_{21} = \{101, 000\}, \quad Fitness = \{100\}$
Threshold has been met. Goto Stage X.

STAGE X: Output

$FinalTestSet = \{101, 000\}, \quad bestCoverage = 100$
Exit.

A comparison of the results obtained using GA with directed search and random search for the circuit in Fig. 6 is presented in Table II for CMGF.

Directed search approach poses a problem for circuits with constant inputs. In such cases, only certain vectors would be permissible which are selected and added into *InitPop*. In case the upper limit has not been met, vectors are randomly selected and added to the set as in the case of GA with random search. The algorithm described here is extended to

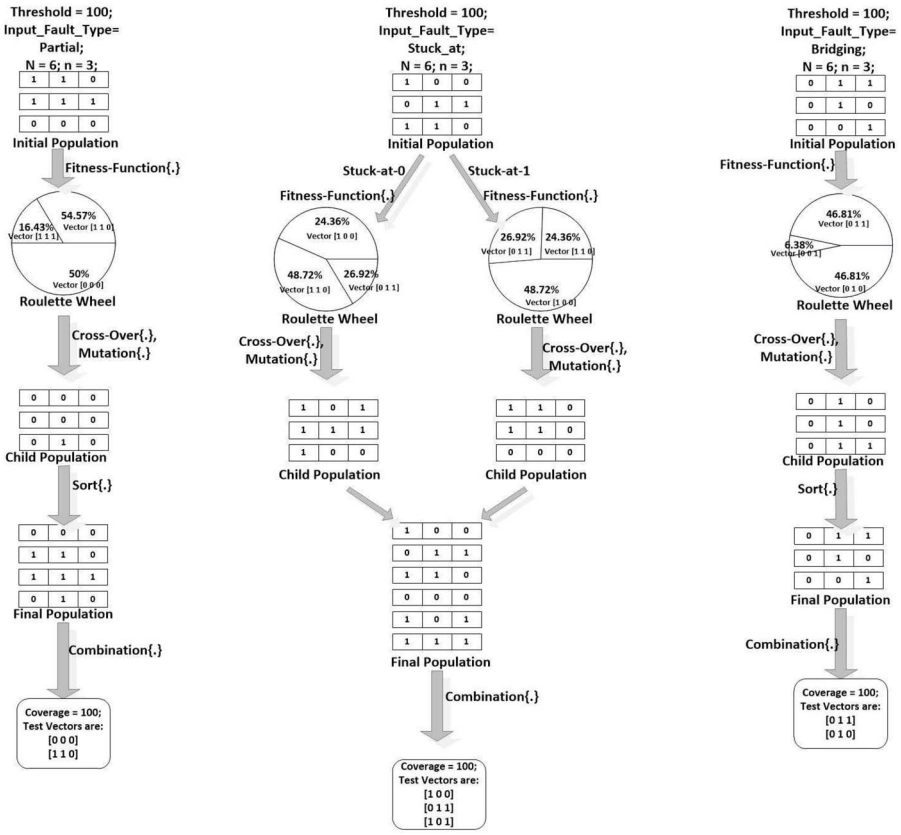

Fig. 9. GA applied to detect (a) PMGF, (b) SaF, and (c) BF in 3_17tc.

TABLE II
COMPARISON OF GA WITH DIRECTED AND RANDOM SEARCH METHODS FOR DETECTING CMGF IN THE CIRCUIT 3_17$tc^{\#}$

Parameter	Directed	Random	†Directed
Time(avg)⋆	<0.01	<0.01	<0.01
V(avg)⋆	2	2	2
$generation$(avg)⋆	1	1.467	1
P(100% Fault Coverage)	1	0.8	1

#Values are for 30 test runs ⋆Values are for 100% fault coverage †For Mirrored RC

the other fault models as well. The only difference arises in the computation of fault coverage% which is carried out as described in Section III-A.

In the case of SaF, through analysis, it was found that fault coverage was always 50% when both Sa0 and Sa1 faults are considered. Since fitness function remains constant, the selection of members in *ParentPop* cannot be directly adopted. This problem is addressed by considering the fault coverage for Sa0 and Sa1 separately. Stages III–VI of the algorithm is performed separately for Sa0 and Sa1. *CumulatedFault* would be half the original value and the faults introduced would be either a 0 or a 1. The *ChildPopulation* obtained at the end of the two processes are combined and added into *FinPop*. Because of the branching, |*FinPop*| generally doubles. *DetectedFault* is found anew by introducing both Sa0 and Sa1 type of faults for all the members of the set *FinPop*. The rest of the process follows as in PMGF and CMGF. The process for PMGF, SaF, and BF is depicted in Fig. 9.

The methodology developed in this section can be applied to the mirrored RC with data traversing from the last gate to the first gate. While doing so, the GOs must take on appropriate constant values to preserve the reversible properties of the former. In Table II, results of mirrored 3_17*tc* have been added to illustrate the applicability of the algorithm when an input is fed to the last gate. Directed search is performed by feeding opposite values to the inputs *b* and *c* for the inverted circuit. One of the possible sets of *InitPop* would be {000, 111, 010}.

Directed search has been devised to cater to the requirements of *k*-CNOT circuits, as, most benchmark circuits commence with a *k*-CNOT gate. The functionality of Fredkin gate requires its control to be activated for swapping contents of signal values on its target lines. In order to obtain maximum coverage, controls must be activated and targets must take on opposite values. RCs with Toffoli and Fredkin with Toffoli as the first gate can be directed using the same approach. It can be observed from Section V that, there has been a significant improvement for *k*-CNOT and Peres-based circuits using directed approach and a slight improvement for Fredkin-based circuits.

E. Test Set Generation for the Set of All Missing-Gate Faults

The algorithm developed here is extended to obtain a single test set for the set of all missing-gate faults. Both CMGF and PMGF are induced individually on the RC as discussed in Section III-A in Stage III of the Algorithm 1 by following the remaining steps without any change. In this case, the

TABLE III
COMPARISON OF CMGF DETECTION PARAMETERS

Ref	Benchmark Circuit	N	n	Ref. Paper		Directed			Random		
				#V	time(s)	#FC %	#V	time(s)	#FC %	#V	time(s)
[13]	$ham7tc^{T}$	23	7	4	6.93	100	4	0.046	100	5	0.375
	$mod5adders^{T}$	21	6	3	1.056	100	3	0.015	100	3	0.265
	$mod5d1^{T}$	8	5	1	0.0479	100	1	<0.01	100	1	<0.01
	$hwb4tc^{T}$	17	4	2	0.058	100	2	<0.01	100	3	0.109
	$rd32^{T}$	4	4	2	0.0202	100	2	<0.01	100	3	<0.01
	3_17tc^{T}	6	3	2	0.017	100	2	<0.01	100	2	0.015
[11]	$hwb5\_13^{T}$	88	28	8	0.05	100	5	0.772	100	6	0.797
	$rd84\_142^{T}$	28	15	9	0.06	100	6	0.046	100	4	0.359
	$sym9\_148^{T}$	210	10	32	0.59	100	1	0.29	100	1	0.297
	$4gt12{-}v0\_86^{T}$	14	5	3	<0.01	100	3	0.082	100	2	<0.01
	$one{-}two{-}three{-}v0\_97^{T}$	11	5	3	<0.01	100	3	0.015	100	3	<0.01
[12]	$hwb5\_13$	88	28	7	0.24	100	5	0.578	100	5	0.563
	$rd84\_142^{T}$	28	15	8	0.14	100	5	0.046	100	9	0.125
	$sym9\_148^{T}$	210	10	14	1.79	100	1	0.525	100	5	24.456
	$4gt12{-}v0\_86^{T}$	14	5	5	0.02	100	2	<0.01	100	2	0.015
	$one{-}two{-}three{-}v0\_97^{T}$	11	5	3	0.02	100	3	¡0.01	100	3	0.015

#V = No. Of Test Vectors for detection #FC % = Fault Coverage Percentage x^{T} = Toffoli based circuit

TABLE IV
COMPARISON OF PMGF DETECTION PARAMETERS

Ref	Benchmark Circuit	N	n	Ref. Paper		Directed			Random		
				#V	time(s)	#FC %	#V	time(s)	#FC %	#V	time(s)
[13]	$ham7tc^{T}$	23	7	4	6.83	100	4	0.062	100	4	0.328
	$mod5adders^{T}$	21	6	3	1.076	100	3	0.015	100	3	0.078
	$mod5d1^{T}$	8	5	1	0.0486	100	1	< 0.01	100	2	0.062
	$hwb4tc^{T}$	17	4	2	0.054	100	2	0.015	100	2	0.062
	$rd32^{T}$	4	4	2	0.0196	100	2	0.015	100	2	0.062
	3_17tc^{T}	6	3	2	0.016	100	2	<0.01	100	2	0.015
[11]	$hwb5\_13^{T}$	88	28	9	0.07	100	5	0.578	100	5	0.563
	$rd84\_142^{T}$	28	15	120	0.03	100	5	0.046	100	9	0.125
	$sym9\_148^{T}$	210	10	120	3.71	100	1	0.525	100	5	24.456
	$4gt12{-}v0\_86^{T}$	14	5	10	<0.01	100	2	<0.01	100	2	0.015
	$one{-}two{-}three{-}v0\_97^{T}$	11	5	3	0.02	100	3	¡0.01	100	3	0.015

#V = No. Of Test Vectors for detection #FC % = Fault Coverage Percentage x^{T} = Toffoli based circuit

DetectedFault matrix takes on a larger dimension with the combined faults and results in a single test set for missing-gate faults.

V. RESULTS AND DISCUSSION

In this section, results obtained using the two approaches of GA are presented along with comparisons with methods proposed in the recent literature. Circuit information for every benchmark circuit is provided with *n* indicating the number of lines, *N* the number of gates and superscripts on every benchmark name to indicate the type of gate library, that the circuit has been implemented from. The circuits are sorted based on *n*. Computation time stated are processor specific. Tables V and VI carry out a comparison of the values obtained using GA with other proposed methods. All the existing works compared here consider 100% fault coverage. GA being a heuristic approach, cannot guarantee global minima in all cases like exact method. As is to be expected, the test set cardinality using GA has increased in a few cases over exact which can be observed in Tables V and VI. GA still remains a viable option as exact cannot be used without expending a great deal of effort for medium and large circuits. Table VIII gives the values for GA with directed and random search approach for each of the faults as discussed in Section IV. It can be observed

TABLE V
COMPARISON OF BF DETECTION PARAMETERS

Ref	Benchmark Circuit	N	n	Ref. Paper		Directed			Random		
				#V	time(s)	#FC %	#V	time(s)	#FC %	#V	time(s)
[49]	$rd32^{T}$	28	15	6	⋆	100	4	<0.01	100	3	0.015
	$ham7tc^{T}$	23	7	4	⋆	100	6	0.343	100	6	0.187
	$graycode6^{T}$	5	6	9	⋆	100	3	0.031	100	5	0.093
	$xor5d1^{T}$	4	5	8	⋆	100	3	0.015	100	4	0.062
	$4\_49{-}12{-}32^{T}$	12	4	5	⋆	100	4	0.015	100	4	<0.01
	$ham3tc^{T}$	5	3	3	⋆	100	3	<0.01	100	3	0.015
[35]	$gf2\hat{}4mult\_19\_83^{T}$	19	12	4	76.3734	98.56	8	1.093	98.64	6	0.656
	$nth\_prime6\_inc\_55\_667^{T}$	55	6	4	116.3606	100	6	3.219	100	6	1.673
	$2of5d1^{T}$	18	6	4	25.4031	100	5	0.343	100	5	0.031
	$5mod5\_fc^{T}$	10	6	3	4.768	100	4	0.109	100	5	0.015

#V = No. Of Test Vectors for detection #FC % = Fault Coverage Percentage x^{T} = Toffoli based circuit ⋆time not mentioned in reference [49]

TABLE VI
COMPARISON OF SAF DETECTION PARAMETERS WITH [13]

Benchmark Circuit	N	n	Ref. Paper		Directed			Random		
			#V	time(s)	#FC %	#V	time(s)	#FC %	#V	time(s)
$ham7tc^{T}$	23	7	3	7.126	100	5	0.078	100	5	0.187
$mod5adders^{T}$	21	6	3	1.12	100	3	0.046	100	4	0.265
$mod5d1^{T}$	8	5	3	0.1866	98.89	4	0.031	98.89	4	0.171
$hwb4tc^{T}$	17	4	3	0.0839	100	3	<0.01	100	3	0.046
$rd32^{T}$	4	4	3	0.0408	97.5	3	<0.01	97.5	4	0.125
3_17tc^{T}	6	3	3	0.018	100	3	<0.01	100	3	0.031

#V = No. Of Test Vectors for detection #FC % = Fault Coverage Percentage x^{T} = Toffoli based circuit

TABLE VII
TEST SET AND COVERAGE FOR COMBINATION OF MISSING-GATE FAULTS—CMGF AND PMGF

Benchmark Circuit	N	n	Faults	Directed			Random		
				#V	TIME(s)	#FC %	#V	time(s)	#FC %
$gf2\hat{}10mult\_109\_509^{T}$	109	30	318	2	4.142	100	20	276.76	71.70
$cycle17\_3^{T}$	48	20	457	3	2.531	100	10	32.111	1.31
$cnt3{-}5\_180^{T}$	20	16	60	1	0.657	100	2	0.719	100
0410184_169^{T}	46	14	95	4	0.109	100	9	0.218	100
$sym6\_316^{T}$	29	14	72	11	0.062	100	10	0.078	100
$nth\_prime7\_inc\_1427\_3172^{T}$	1427	7	3197	7	706.275	98	7	735.567	96.90

#V = No. Of Test Vectors for detection #FC % = Fault Coverage Percentage x^{T} = Toffoli based circuit

that the values obtained using directed approach have surpassed values obtained using the random approach in terms of time taken, coverage and number of test vectors, in most cases. As the complexity of the circuit decreases, the values obtained using the two methods nearly coincide which can be attributed to the smaller |*InitPop*|. In the case of symmetric and cyclic circuits, as bit width increases, the disparity in the results obtained using the two methods is very large with directed search being more favorable. This can be observed in $cycle17_3$ circuit for PMGF. These results lead to the conclusion that GA being a powerful tool can be amplified in terms of effectiveness by directing the search in the initial stage. Table VII shows the values obtained when the algorithm is extended to produce a single test set for missing-gate fault detection.

A. Comparison of Results for BF

Table V compares results obtained for BF with those proposed in [35] and [49]. There is an average reduction of 37.14% test vectors with GA compared to [49]. For the same test set, time taken by GA is less than in exact approach although the test set cardinality has increased by an average of 27%.

TABLE VIII
TEST SET AND COVERAGE FOR PMGF, CMGF, SAF, AND BF USING DIRECTED AND RANDOM SEARCH APPROACH FOR BENCHMARK CIRCUITS

			PMGF							CMGF						
				Directed			Random				Directed			Random		
BenchMark Circuit	N	n	Total Faults	#FC %	#V	time(s)	#FC %	#V	time(s)	Total Faults	#FC %	#V	time(s)	#FC %	#V	time(s)
bw_291^{T}	307	87	307	100	4	9.593	100	5	16.406	432	100	4	3.5	100	5	6.421
$hwb7\_302^{F}$	281	73	281	100	26	39.85	100	29	39.96	426	100	17	31.09	100	18	33.67
$hwb6\_14^{F}$	159	46	159	100	9	5.76	100	9	6.64	241	100	11	4.328	100	11	4.69
$ham15\_298^{T}$	153	45	153	100	1	0.109	100	2	4.48	157	100	1	0.09	100	2	4.142
$cycle10\_293^{T}$	78	39	78	100	2	0.828	100	3	1.203	98	100	1	0.09	100	2	6.143
$rd84\_313^{T}$	104	34	104	100	16	1.31	100	21	2.734	143	100	19	1.33	100	23	2.69
$gf2\hat{}10mult\_109\_509^{T}$	109	30	209	100	2	2.546	61.72	4	74.039	109	100	2	1.828	63.3	15	24.845
$t-add-8^{T}$	122	24	160	100	14	1.339	88.13	22	6.437	122	100	14	1.132	90.16	17	4.296
mux_246^{T}	35	22	131	25.95	2	0.156	4.58	2	5.54	35	97.14	22	12.156	34.29	17	2.063
$cycle17\_3^{T}$	48	20	409	100	3	2.071	0.73	2	39.799	48	100	3	0.908	6.25	2	9.015
$ryy6\_256^{T}$	44	17	328	37.8	8	10.859	37.8	8	15.562	44	100	2	1.671	54.55	2	6.656
$cnt3-5\_180^{T}$	20	16	40	100	2	0.078	100	2	0.453	20	100	2	0.594	100	2	0.328
$ham15tc1^{T}$	132	15	352	95.74	15	12.178	93.18	8	45.348	132	96.97	14	4.851	98.48	9	24.455
0410184_169^{T}	46	14	49	100	4	0.078	100	3	0.062	46	100	5	0.062	100	4	0.063
$sym6\_316^{T}$	29	14	43	100	11	0.062	100	10	0.031	29	100	10	0.062	100	11	0.031
$squar5\_261^{T}$	43	13	95	100	7	0.265	100	6	0.219	43	100	8	0.281	100	9	0.203
$cycle10\_2^{T}$	19	12	100	100	2	0.062	100	2	0.07	19	100	2	0.062	100	2	0.201
$sym9\_148^{T}$	210	10	756	100	1	0.525	100	1	17.273	210	100	1	0.29	100	1	9.592
$sys6-v0\_144^{T+P}$	15	10	23	100	4	0.015	100	4	0.015	15	100	3	0.015	100	3	0.078
$urf\_2\_277^{T}$	3144	8	3777	94.07	8	74.455	94.07	8	74.455	44	100	2	1.671	54.55	2	6.656
$nth\_prime7\_inc\_1427\_3172^{T}$	1427	7	1770	95.58	7	16.957	95.99	7	16.304	1427	97.83	7	14.909	96.71	7	43.472
$hwb7\_60^{F}$	166	7	396	100	7	9.375	100	7	8.937	30	100	7	13.329	98.80	7	49.23
$rd53\_131^{T}$	28	7	24	100	5	0.015	100	6	0.046	28	100	7	0.062	100	6	<0.01
$hwb6\_57^{F}$	65	6	122	100	5	0.438	100	5	1.515	65	100	6	2.249	100	6	1.218
$hwb5\_54^{F}$	24	5	30	100	4	0.015	100	4	0.125	24	100	4	0.0310	100	5	0.421
$4gt12-v0\_86^{T}$	14	5	20	100	3	0.015	100	3	0.015	14	100	3	0.11	100	2	<0.01
$one-two-three-v0\_97^{T}$	11	5	23	100	3	<0.01	100	3	<0.01	11	100	3	0.015	100	3	0.015
$alu-v0\_26^{T}$	6	5	8	100	2	0.015	100	2	<0.01	6	100	2	0.015	100	2	0.015
$hwb4\_51^{F}$	11	4	10	100	2	0.015	100	2	0.015	11	100	2	0.015	100	2	0.015
			SaF							BF						
				Directed			Random				Directed			Random		
BenchMark Circuit	N	n	Total Faults	#FC %	#V	time(s)	#FC %	#V	time(s)	Total Faults	#FC %	#V	time(s)	#FC %	#V	time(s)
bw_291^{T}	307	87	53592	82.28	5	663.81	-	-	†TO	2304456	-	-	†TO	-	-	†TO
$hwb7\_302^{F}$	281	73	41172	99.9	7	1971.547	-	-	†TO	1482192	-	-	†TO	-	-	†TO
$hwb6\_14^{F}$	159	46	14720	82.29	6	79.296	82.29	6	84.95	331200	87.13	6	681.655	86.54	14	2075.418
$ham15\_298^{T}$	153	45	13860	80.84	5	527.958	80.84	6	975.797	304920	91.26	9	5057.34	87	4	4649.65
$cycle10\_293^{T}$	78	39	6162	83.14	19	109.056	82.88	19	140.925	117078	92.12	11	680.173	91.84	4	791.142
$rd84\_313^{T}$	104	34	7140	79.43	8	252.99	79.3	8	235.418	117810	87.55	22	677.573	87.43	22	627.14
$gf2\hat{}10mult\_109\_509^{T}$	109	30	6600	95.5	7	121.809	88.83	15	102.88	95700	97.16	15	577.157	92.68	15	718.27
$t-add-8^{T}$	122	24	5904	91.82	12	46.877	89.33	12	46.877	67896	97.46	14	259.306	95.81	13	263.918
mux_246^{T}	35	22	1584	99.94	11	3.687	86.36	11	3.437	16632	100	14	90.024	93.94	11	14.534
$cycle17\_3^{T}$	48	20	1960	100	7	34.769	90.2	10	16.174	18620	99.33	13	81.332	97.79	10	143.758
$ryy6\_256^{T}$	44	17	1530	99.87	12	73.207	96.01	18	10.515	12240	99.98	9	31.316	99.77	8	61.864
$cnt3-5\_180^{T}$	20	16	672	92.56	8	9.438	92.56	8	18.297	5040	97.34	15	3.093	97.62	9	31.051
$ham15tc1^{T}$	132	15	3990	100	10	63.631	100	7	30.126	27930	100	12	151.116	100	9	96.02
0410184_169^{T}	46	14	1316	100	6	1.687	100	6	1.578	8554	100	11	4.484	100	8	4.172
$sym6\_316^{T}$	29	14	840	91.31	7	0.891 29	91.31	7	0.922	5460	96.15	12	1.687	96.08	13	13.609
$squar5\_261^{T}$	43	13	1144	94.67	6	2.82	94.67	9	2.658	6864	95.92	8	8.628	95.92	7	4.047
$cycle10\_2^{T}$	19	12	480	100	3	0.875	100	4	0.546	2640	100	7	3.046	100	7	2.391
$sys6-v0\_144^{T+P}$	15	10	320	95.63	5	0.141	95.63	7	0.515	1350	98.47	7	0.25	98.47	8	0.829
$sym9\_148^{T}$	210	10	4220	99.98	10	101.898	99.27	10	93.485	18990	100	9	43.89	100	9	48.831
$urf\_2\_277^{T}$	3144	8	50320	99.9	8	184.11	99.86	8	723.68	176064	99.73	8	2590.541	99.73	8	2597.485
$nth\_prime7\_inc\_1427\_3172^{T}$	1427	7	19992	99.78	7	391.9	99.86	7	229.698	59976	99.7	7	693.734	99.7	7	701.58
$hwb7\_60^{F}$	166	7	2338	100	2	7.5	100	5	6.875	7014	100	7	520.108	99.86	7	7.344
$rd53\_131^{T}$	28	7	406	98.28	3	0.062 28	98.28	4	0.078	1218	99.84	5	0.124	99.84	5	0.234
$hwb6\_57^{F}$	65	6	792	100	2	0.641	100	5	0.593	1980	100	5	1.344	100	5	1.032
$hwb5\_54^{F}$	24	5	250	100	2	0.046	100	5	1.375	500	100	4	0.266	100	4	0.016
$4gt12-v0\_86^{T}$	14	5	150	95.33	4	0.062	95.33	4	0.015	300	100	4	0.015	100	4	0.031
$one-two-three-v0\_97^{T}$	11	5	120	95.83	4	1.171	95.83	3	<0.01	240	86.67	4	0.015	86.67	3	0.01
$alu-v0\_26^{T}$	6	5	70	100	3	<0.01	100	5	0.016	140	100	4	<0.01	100	4	<0.01
$hwb4\_51^{F}$	11	4	96	100	2	0.015	100	4	<0.01	144	100	2	0.078	100	2	<0.01

#V = No. Of Test Vectors for detection #FC % = Fault Coverage Percentage x^{T} = Toffoli based circuit x^{F} = Fredkin based circuit x^{P} = Peres based circit †TO - time out

B. Comparison of Results for Missing-Gate Faults

Tables III and IV compare the values obtained for missing-gate type faults with exact method as described in [13], SAT method in [11], and PBO in [12]. The number of test vectors required for complete fault coverage in the case of GA is found to be the same in most of the cases as that using the

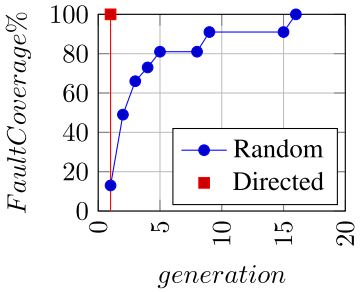

Fig. 10. Variation of fault coverage with *generation* for the circuit *cycle*10_2 using GA with random and directed approaches.

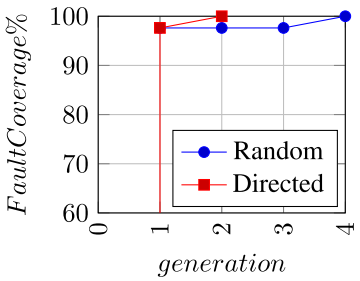

Fig. 11. Variation of fault coverage with *generation* for the circuit *ham*15−109−214 using GA with random and directed approaches.

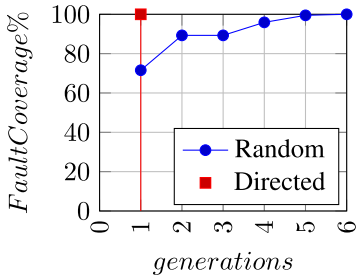

Fig. 12. Variation of fault coverage with *generation* for the circuit *sym*9_148 using GA with random and directed approaches.

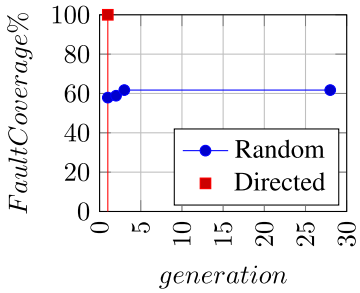

Fig. 13. Variation of fault coverage with *generation* for the circuit *gf*2^10*mult*_109_509 using GA with random and directed approaches.

exact approach with lesser computation time. Test set requirement for full coverage is much less compared to SAT- and PBO-based approaches with an average decrease of 72.73% and 56.76% test vectors, respectively. PMGF results compared with [11] show a decrease of 93.89% test vectors on average. Directed search produced better results than random search in a majority of circuits.

C. Comparison of Results for SaF

Table VI compares values obtained for SaF type faults using GA and exact approach stated in [13]. GA takes much lesser time for computation in all cases with nearly the same number of test vectors.

D. Comparison of GA Incorporating Directed and Random Approach

Figs. 10–13 show the variation of fault coverage percentage with *generation*. Directed approach selects individuals which generally have higher fitness function than most vectors in the gene pool while in random search approach, the probability of selection of the same vector would be $(1/2^n)$ which is a very small value as bit size increases. In the case of *ham*15−109−214, though the initial selection was poorer in GA with the directed approach than in the random approach, it produced maximum coverage in 2 *generation* while random approach required 4 due to constancy in fault coverage% for 3 such generations. Circuits *gf*2^10*mult*_109_509, *sym*9_148, and *cycle*10_2 converge within one generation for directed approach. The random approach in *gf*2^10*mult*_109_509 yields a fault coverage% less than 65% even with 25 *generation* while directed approach produces the result in a single *generation* demonstrating the superiority of directed approach over random when applied to *k*-CNOT circuits with high input variables.

VI. Conclusion

In this paper, a GA-based platform is used to obtain a test set for optimum coverage using less computation time for small, medium, and large circuits. This is the first attempt in reversible literature to use principles of GA for test set generation with optimum fault coverage for a wide class of faults. A GA-based method for fault detection in RCs designed with *k*-CNOT, Peres, and Fredkin gates has been proposed. An algorithm for a directed search method exploring unique properties of RCs as opposed to the random search approach inherent to GA is also presented. Significant improvement is observed with respect to the number of generations, computation time and fault coverage. It is observed that the algorithm worked very well with circuits having small, medium, and large input lines in the case of missing-gate faults. Detection of SaF and BF in large RCs takes longer convergence time owing to the increased fault cardinality in the order of 10^6. It would be interesting to combine circuit partitioning method with GA to catalyze convergence. The fault detection methodology used here can be extended to other subsets of reversible fault models such as additional gate and additional control faults.

To conclude, by using GA proposed in this paper teamed with an appropriate algorithm to direct the initial stage, faster convergence can be ensured. In RCs having few input lines but a large number of gates, the time taken for convergence was found to be relatively higher. Hence, future work would involve exploring ways to decrease the number of test vectors, reduce test generation complexity by using a suitable circuit partitioning technique and to devise an algorithm for

new quantum fault models. Fault diagnosis using GA has also been earmarked for future research.

REFERENCES

[1] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Res. Develop.*, vol. 5, no. 3, pp. 183–191, Jul. 1961.

[2] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Develop.*, vol. 17, no. 6, pp. 525–532, Nov. 1973.

[3] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

[4] C. Taraphdar, T. Chattopadhyay, and J. N. Roy, "Mach–Zehnder interferometer-based all-optical reversible logic gate," *Opt. Laser Technol.*, vol. 42, no. 2, pp. 249–259, 2010.

[5] X. Ma, J. Huang, C. Metra, and F. Lombardi, "Reversible gates and testability of one dimensional arrays of molecular QCA," *J. Electron. Testing*, vol. 24, no. 1, pp. 297–311, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10836-007-5042-2

[6] I. Polian, T. Fiehn, B. Becker, and J. P. Hayes, "A family of logical fault models for reversible circuits," in *Proc. 14th Asian Test Symp. (ATS)*, Kolkata, India, Dec. 2005, pp. 422–427.

[7] A. Mishchenko and M. Perkowski, "Logic synthesis of reversible wave cascades," in *Proc. IEEE/ACM Int. Workshop Logic Synthesis*, Jun. 2002, pp. 197–202.

[8] T. Toffoli, "Reversible computing," in *Proc. Int. Colloquium Automata Lang. Program.*, 1980, pp. 632–644.

[9] E. Fredkin and T. Toffoli, "Conservative logic," *Int. J. Theor. Phys.*, vol. 21, no. 3, pp. 219–253, 1982.

[10] A. Peres, "Reversible logic and quantum computers," *Phys. Rev. A Covering Atomic Mol. Opt. Phys. Quantum Inf.*, vol. 32, no. 6, pp. 3266–3276, 1985.

[11] H. Zhang, R. Wille, and R. Drechsler, "SAT-based ATPG for reversible circuits," in *Proc. IEEE 5th Int. Design Test Workshop*, Abu Dhabi, UAE, Dec. 2010, pp. 149–154.

[12] R. Wille, H. Zhang, and R. Drechsler, "ATPG for reversible circuits using simulation, Boolean satisfiability, and pseudo Boolean optimization," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Chennai, India, Jul. 2011, pp. 120–125.

[13] A. N. Nagamani, S. Ashwin, B. Abhishek, and V. K. Agrawal, "An exact approach for complete test set generation of Toffoli–Fredkin–Peres based reversible circuits," *J. Electron. Testing*, vol. 32, no. 2, pp. 175–196, 2016.

[14] G. J. Woeginger, "Exact algorithms for NP-hard problems: A survey," in *Combinatorial Optimization—Eureka, You Shrink!* Heidelberg, Germany: Springer, 2003, pp. 185–207.

[15] M. Srinivas and L. M. Patnaik, "A simulation-based test generation scheme using genetic algorithms," in *Proc. IEEE 6th Int. Conf. VLSI Design*, Mumbai, India, Jan. 1993, pp. 132–135.

[16] N. Alhagi, M. Hawash, and M. Perkowski, "Synthesis of reversible circuits with no ancilla bits for large reversible functions specified with bit equations," in *Proc. 40th IEEE Int. Symp. Multiple-Valued Logic*, Barcelona, Spain, May 2010, pp. 39–45.

[17] M. B. Tahoori, M. Momenzadeh, J. Huang, and F. Lombardi, "Defects and faults in quantum cellular automata at nano scale," in *Proc. 22nd IEEE VLSI Test Symp.*, Napa County, CA, USA, 2004, pp. 291–296.

[18] D. Y. Feinstein, V. S. S. Nair, and M. A. Thornton, "Advances in quantum computing fault tolerance and testing," in *Proc. HASE*, Dallas, TX, USA, 2007, pp. 369–370.

[19] M. Lukac, M. Kameyama, M. Perkowski, P. Kerntopf, and C. Moraga, "Fault models in reversible and quantum circuits," in *Advances in Unconventional Computing : Volume 1: Theory*, A. Adamatzky, Ed. Cham, Switzerland: Springer, 2017, pp. 475–493, doi: 10.1007/978-3-319-33924-5_19.

[20] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An online resource for reversible functions and reversible circuits," in *Proc. Int. Symp. Multi-Valued Logic*, Dallas, TX, USA, 2008, pp. 220–225.

[21] D. Maslov, G. Dueck, and N. Scott. (2005). *Reversible Logic Synthesis Benchmarks Page*. [Online]. Available: http://www.cs.uvic.ca/∼dmaslov

[22] D. P. Vasudevan, P. K. Lala, J. Di, and J. P. Parkerson, "Reversible-logic design with online testability," *IEEE Trans. Instrum. Meas.*, vol. 55, no. 2, pp. 406–414, Apr. 2006.

[23] S. N. Mahammad and K. Veezhinathan, "Constructing online testable circuits using reversible logic," *IEEE Trans. Instrum. Meas.*, vol. 59, no. 1, pp. 101–109, Jan. 2010.

[24] D. P. Vasudevan, P. K. Lala, and J. P. Parkerson, "Online testable reversible logic circuit design using NAND blocks," in *Proc. 19th IEEE Int. Symp. Defect Fault Tolerance VLSI Syst. (DFT)*, Cannes, France, 2004, pp. 324–331.

[25] N. Farazmand, M. Zamani, and M. B. Tahoori, "Online fault testing of reversible logic using dual rail coding," in *Proc. IEEE 16th Int. On-Line Testing Symp. (IOLTS)*, 2010, pp. 204–205.

[26] B. Sen, J. Das, and B. K. Sikdar, "A DFT methodology targeting online testing of reversible circuit," in *Proc. Int. Conf. Devices Circuits Syst. (ICDCS)*, Coimbatore, India, Mar. 2012, pp. 689–693.

[27] K. N. Patel, J. P. Hayes, and I. L. Markov, "Fault testing for reversible circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 8, pp. 1220–1230, Aug. 2004.

[28] J. P. Hayes, I. Polian, and B. Becker, "Testing for missing-gate faults in reversible circuits," in *Proc. 13th Asian Test Symp.*, Nov. 2004, pp. 100–105.

[29] X. Fang-Ying, C. Han-Wu, L. Wen-Jie, and L. Zhi-Giang, "Fault detection for single and multiple missing-gate faults in reversible circuits," in *Proc. IEEE Congr. Evol. Comput.*, Hong Kong, Jun. 2008, pp. 131–135.

[30] M. Zamani, N. Farazmand, and M. B. Tahoori, "Fault masking and diagnosis in reversible circuits," in *Proc. 16th IEEE Eur. Test Symp. (ETS)*, Trondheim, Norway, 2011, pp. 69–74.

[31] D. K. Kole, H. Rahaman, D. K. Das, and B. B. Bhattacharya, "Derivation of test set for detecting multiple missing-gate faults in reversible circuits," *Comput. Elect. Eng.*, vol. 39, no. 2, pp. 225–236, 2013.

[32] H. Rahaman, D. K. Kole, D. K. Das, and B. B. Bhattacharya, "On the detection of missing-gate faults in reversible circuits by a universal test set," in *Proc. 21st Int. Conf. VLSI Design (VLSID)*, Hyderabad, India, 2008, pp. 163–168.

[33] J. Zhong and J. C. Muzio, "Analyzing fault models for reversible logic circuits," in *Proc. IEEE Int. Conf. Evol. Comput.*, Vancouver, BC, Canada, 2006, pp. 2422–2427.

[34] P. Sarkar and S. Chakrabarti, "Universal test set for bridging fault detection in reversible circuit," in *Proc. 3rd Int. Design Test Workshop*, Monastir, Tunisia, Dec. 2008, pp. 51–56.

[35] A. N. Nagamani, B. Abhishek, and V. K. Agrawal, "Deterministic approach for bridging fault detection in Peres–Fredkin and Toffoli based reversible circuits," in *Proc. IEEE Int. Conf. Comput. Intell. Comput. Res. (ICCIC)*, Madurai, India, Dec. 2015, pp. 1–6.

[36] M. Zamani and M. B. Tahoori, "Online missing/repeated gate faults detection in reversible circuits," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Vancouver, BC, Canada, 2011, pp. 435–442.

[37] A. Deb *et al.*, "Reversible synthesis of symmetric functions with a simple regular structure and easy testability," *ACM J. Emerg. Technol. Comput. Syst. (JETC)*, vol. 12, no. 4, 2016, Art. no. 34.

[38] G. Ascia, V. Catania, and M. Palesi, "A GA-based design space exploration framework for parameterized system-on-a-chip platforms," *IEEE Trans. Evol. Comput.*, vol. 8, no. 4, pp. 329–346, Aug. 2004.

[39] C. Bernardeschi, L. Cassano, M. G. C. A. Cimino, and A. Domenici, "GABES: A genetic algorithm based environment for SEU testing in SRAM-FPGAs," *J. Syst. Archit.*, vol. 59, no. 10, pp. 1243–1254, 2013.

[40] F. Corno, P. Prinetto, M. Rebaudengo, and M. S. Reorda, "GATTO: A genetic algorithm for automatic test pattern generation for large synchronous sequential circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 8, pp. 991–1000, Aug. 1996.

[41] D. G. Saab, Y. G. Saab, and J. A. Abraham, "Automatic test vector cultivation for sequential VLSI circuits using genetic algorithms," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 10, pp. 1278–1285, Oct. 1996.

[42] A. Paler, I. Polian, and J. P. Hayes, "Detection and diagnosis of faulty quantum circuits," in *Proc. 17th Asia South Pac. Design Autom. Conf.*, Sydney, NSW, Australia, Jan. 2012, pp. 181–186.

[43] I. Polian and J. P. Hayes, "Advanced modeling of faults in reversible circuits," in *Proc. Design Test Symp. East-West (EWDTS)*, St. Petersburg, Russia, Sep. 2010, pp. 376–381.

[44] S. Sivanandam and S. Deepa, *Introduction to Genetic Algorithms*. Heidelberg, Germany: Springer-Verlag, 2007. [Online]. Available: http://www.springer.com/in/book/9783540731894

[45] A. C. Koenig, *A Study of Mutation Methods for Evolutionary Algorithms*, Univ. at Missouri-Rolla, Rolla, MO, USA, 2002.

[46] K. Deb and D. Deb, "Analysing mutation schemes for real-parameter genetic algorithms," *Int. J. Artif. Intell. Soft Comput.*, vol. 4, no. 1, pp. 1–28, 2014.

[47] K. Ramasamy, R. Tagare, E. Perkins, and M. Perkowski, "Fault localization in reversible circuits is easier than for classical circuits," in *Proc. Int. Workshop Logic Synthesis (IWLS)*, Temecula, CA, USA, 2004.

[48] H. Thapliyal, N. Ranganathan, and S. Kotiyal, "Design of testable reversible sequential circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 7, pp. 1201–1209, Jul. 2013.
[49] H. Rahaman, D. K. K. D. K. Kole, D. K. Das, and B. B. Bhattacharya, "Optimum test set for bridging fault detection in reversible circuits," in *Proc. 16th Asian Test Symp. (ATS)*, Beijing, China, Oct. 2007, pp. 125–128.

A. N. Nagamani (M'15) received the M.Tech. degree from VTU, Belgaum, India, where she is currently pursuing the Ph.D. degree in reversible circuit optimization and testing.

She is an Assistant Professor with the Department of Electronics and Communication Engineering, PES Institute of Technology, Bengaluru, India. Her current research interests include reversible logic optimization and testing, low power digital very large-scale integration (VLSI), analog and mixed signal design, and VLSI architecture for optimized performance. She has published several publications in the above mentioned areas in the peer reviewed conferences and journals.

S. N. Anuktha is currently pursuing the graduate degree in electronics and communications engineering with the PES Institute of Technology, Bengaluru, India.

Her current research interests include reversible logic, digital communication, and very large-scale integration for signal processing.

N. Nanditha is currently pursuing the graduate degree in electronics and communications engineering with the PES Institute of Technology, Bengaluru, India.

Her current research interests include reversible logic, memory design, and very large-scale integration for signal processing.

Vinod Kumar Agrawal (SM'07) received the M.Tech. degree from the Indian Institute of Technology Kanpur, Kanpur, India, and the Ph.D. degree from the Department of Computer Science, Indian Institute of Science, Bengaluru, India, in 1986.

He joined ISRO Satellite Center, Bengaluru, in 1978, where he was the Group Director of the Control Systems Group. He is currently the Director for the Crucible of Research and Innovation Laboratory, PES Institute of Technology, Bengaluru, India. His expertise is in Petri nets and evolutionary computing. His current research interests include emerging technologies for circuit design and optimization. He has published several papers in peer reviewed conferences and journals in his specialized fields.