An efficient greedy algorithm to create adaptive trees for fault localization in binary reversible circuits

Kavitha Ramasamy, Radhika Tagare, Ed Perkins and Marek Perkowski
Department of Electrical and Computer Engineering, Portland State University,
Portland, Oregon, 97207-0751, mperkows@ee.pdx.edu

Abstract

There is recently an interest in test generation for reversible circuits, but nothing has been published about fault localization in such circuits. The paper deals with fault localization for binary permutative reversible circuits. We concentrate on functional test based fault localization, and we generate an adaptive tree that depicts those faults that can be detected and localized. A striking property of reversible circuits is that they exhibit "symmetric" adaptive trees. This helps considerably by being able to generate only half of the tree, and the other half is created as the mirror image of the first half. Because each test covers half faults, it is relatively easy to generate the tree using a greedy algorithm. Thus the problem of fault localization of reversible circuits is easier than the same problem for standard irreversible circuits.

1. INTRODUCTION

Low power consumption is a major issue in VLSI circuits today. As the transistor size decreases, power consumption and heat dissipation become two major problems for the IC designers. Techniques like voltage scaling, low power layout are already in practice to obtain circuits with low power. The motivation for studying reversible adiabatic circuits comes mainly from this increasing demand for low energy dissipation computation [5]. Reversible circuits play also a vital role in quantum computing [6] and emerging nanotechnologies [7, 8, 9]. To ensure the proper functionality and the durability of an integrated circuit; testing and failure analysis are extremely important during and after its design and manufacturing. The main idea behind fault localization in future highly parallel redundant logic systems is self-repair based on localization and replacement of faulty modules. We show that this task is easier when the circuit is reversible.

2. TEST GENERATION AND FAULT LOCALIZATION

Testing of a circuit verifies the correctness of the hardware. Any circuit can be tested either by parametric or by functional tests. Parametric tests include verifying several parameters of a circuit like its AC, DC parameters, maximum current, maximum power dissipation etc. Functional tests include verifying if the circuit under consideration functions exactly as desired. An undesired instance in the circuit is referred to as a *fault*. Thus they may occur due to logical flaw in the behavioral or structural design of the integrated circuit (IC) categorized as single/multiple stuck at faults or due to some physical flaws in the manufacturing process of the IC like bridging or delay faults. Whereas a fixed logical value ('0' or '1') at one or more nodes of a circuit is the cause for single/multiple "stuck-at faults". A model that depicts the failures in the functional behavior of the circuit is called as fault model. Most commonly used fault model is the "single stuck-at" fault model.

There are two aspects to testing a circuit. One is Fault Detection and the other is termed Fault Localization. Detecting the presence of fault in a circuit is called fault detection whereas finding the exact location of this fault comes under the name of fault localization. So far, nothing has been published on self-repair and fault-localization of reversible circuits, although it is a common agreement that future technologies will be both low power and fault-tolerant. The paper uses a novel approach to choose an input test vector depending on the choice of circuit partitions. A counter-example test vector at this chosen partition is applied to find the corresponding test vector at the inputs, and then the test vector is checked for its coverage measure. Our goal is to try to use this approach to locate booth the unique and equivalent faults in reversible circuits, while trying to generate a symmetric adaptive tree.

In our approach we focus on functional test based fault localization to locate single stuck at faults in binary reversible circuits.

3. PREVIOUS WORK ON TESTING BINARY REVERSIBLE CIRCUITS

3.1. Test generation for reversible circuits and design for test of reversible circuits.

Patel et al., [1] use a direct approach to generate set of test vectors to detect all faults in a reversible logic circuit by decomposing larger circuits into smaller sub-circuits (block partitioning). They formulate finding the minimal test set as an Integer Linear Programming (ILP) problem. Single stuck-at fault model is used to detect faults in internal lines and primary

input and output lines of the circuit. Their main contribution are the following observations regarding reversible circuits:

i) Any test set that is complete for the single stuck-at fault model is also complete for multiple stuck-at fault model.
ii) Each test vector covers exactly half of the faults, and each fault is covered by exactly half of the possible test vectors.

Ugur Kalay et al., [2] use universal test set to detect faults in AND-EXOR based circuits. They too use a similar fault model as Patel et al. This method can be adopted for a special type of reversible cascades that are based on ESOP circuits [4]. Both [1] and [2] focus only on fault detection. Because of importance of fault localization, we extend these works towards this new aspect of reversible circuits.

3.2. Fault Localization of irreversible circuits

The two most popular approaches to Fault Localization are *A) Preset Method* and *B) Adaptive Tree Method* [3]. In literature, both methods starts from a fault table which has tests (input vectors) as rows and all possible faults as columns. The goal of preset method is to find a minimal test set to locate all the faults in the circuit. The minimal test set is found using the algorithm to solve the covering problem which limits the applicability of this approach to relatively small problems. Therefore we concentrate on only the Adaptive Tree Method in this research. To speed up the method and allow it to be used without generating all tests, we created an algorithm that does not create the fault table at all. This way, adaptive trees can be created for large circuits.

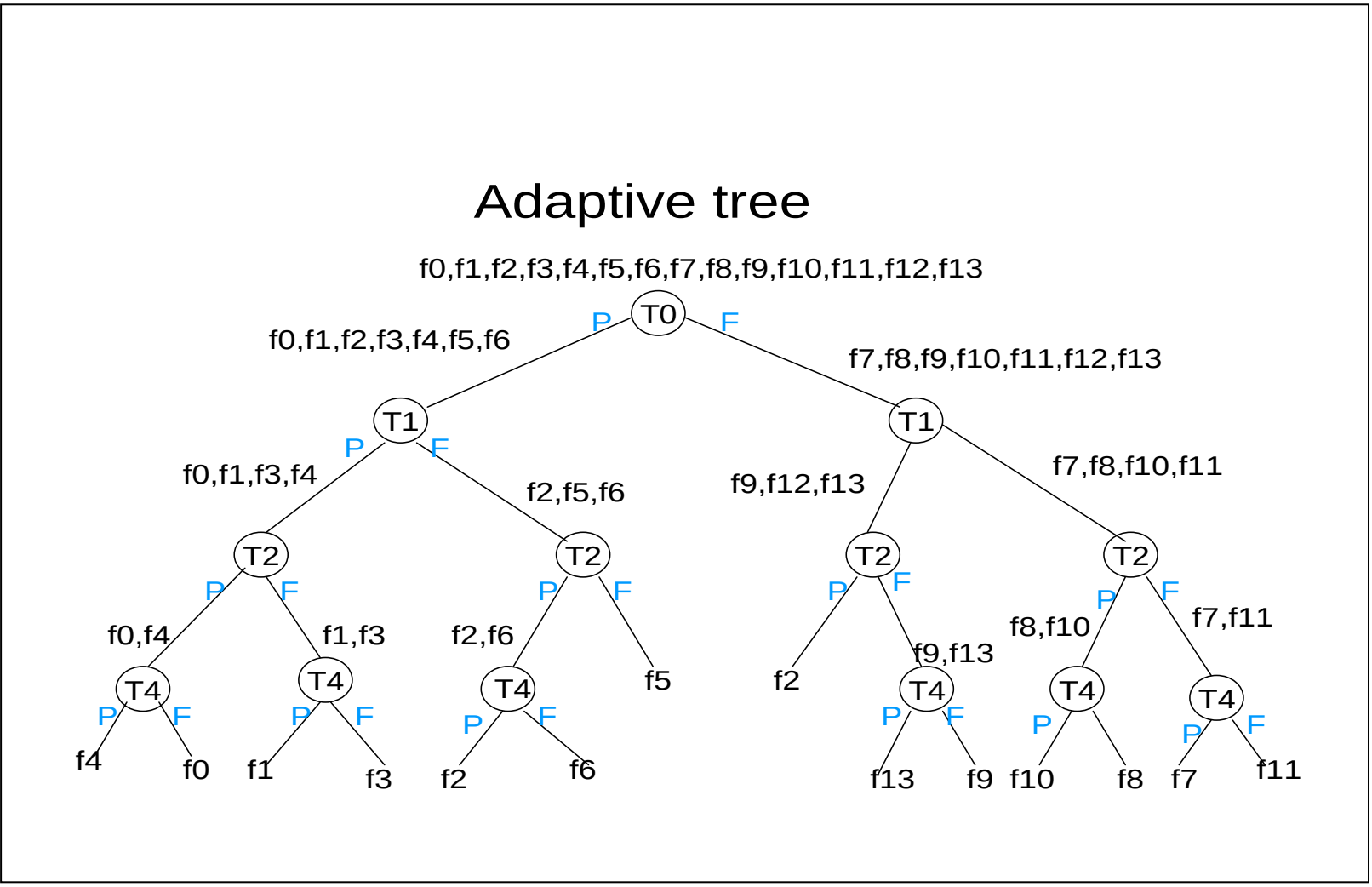
The Adaptive Tree Method. Adaptive tree is represented by a directed tree data structure. Fig. 1 shows an example of an Adaptive Tree. In the figure, the nodes correspond to various tests and the branches correspond to different circuit responses to these tests. Selection of a test to be applied on the current level is determined by the response of the circuit to the chosen test in the previous level of the tree. The choice of the test at each node is based on the following rule: at every node choose a test that partitions the incoming subset/set of faults into the balanced subsets of faults (i.e. with their cardinalities as close as possible). (Such choice creates a well-balanced tree, thus the tree allows for the fastest fault localization assuming equal faults probabilities). One subset is of the faults that can be detected by this test and the other is of those which can not be detected by this test. This procedure is continued until every branch ends with a single fault. This is the fault that can be located exactly by sequentially applying the tests at every node on the path reaching the fault; from top to bottom of the adaptive tree. Observe that in case of non-reversible circuit the tree is created based on the fault table. At each level of recursive tree generation, an attempt is made to choose a row in this table that approximately covers half of the faults remaining in this node. This requires first to create the table which may be very large, and next, it is difficult to select the good row, since there are many candidates and often none is close to covering half faults. In reversible circuits, because of property ii), every test covers half faults, thus tree generation is easier and moreover, it does not require creating the table, which substantially increases the efficiency.

Note: Observe in the Figure that fault f4 passes through the tests at every level. Hence this path in the adaptive tree corresponds to the faultless circuit.

4. ADAPTIVE TREE GENERATION FOR REVERSIBLE CIRCUITS.

We focus here on functional test based Fault Localization to locate single stuck-at faults in binary reversible circuits which particularly comprise *of Toffoli, Feynman and NOT gates.* We assume circuits with faults only in internal lines, primary input lines and output lines of the circuit. The circuit is analyzed by partitioning it into sub-circuits assuming one gate per every wire of the partition. Our approach to fault localization is based on the greedy heuristics for the adaptive tree method. The two already mentioned characteristics of the reversible circuits by Patel et al [1] make reversible circuits exhibit a nice symmetry.

We thus can get a balanced adaptive tree as shown above in Fig. 1. This tree is said to be symmetric because for a particular level the same test vector can be used for partitioning at every node. Due to this symmetry property observed in reversible circuits, adaptive trees for reversible circuits exhibit a special mirror image property when folded over the test at level 0. Another advantage of Adaptive Tree approach is that it avoids creation of the entire fault cover table for every test vector and all possible faults. Thus saving time and memory space wasted otherwise, by the Preset method; where all the tests need to be applied regardless of the circuit output. So we also require less number of test vectors generated for our approach.



5. SIMULATOR OF REVERSIBLE CIRCUITS.

We designed a simulator to analyze binary reversible circuits by mainly evaluating the circuit output for a particular input test vector applied to it. The simulator is designed to scan the circuit description first. Therefore it can operate in two directions, either forward or backward. It operates in forward direction to find the output of the circuit at every node of every partition in the circuit. The simulator operates in the backward direction to find the input test vector corresponding to counter-example vector applied at some partition of the circuit. Observe, that it is again the property of reversible circuits that allows for easy simulation of a circuit in both directions and in exactly the same way, because for each n-bit reversible cell F there exist a unique inverse cell $F^{-1.}$ Moreover, the Feynman and Toffoli gates that we use are their own inverses which speeds-up the simulation further.

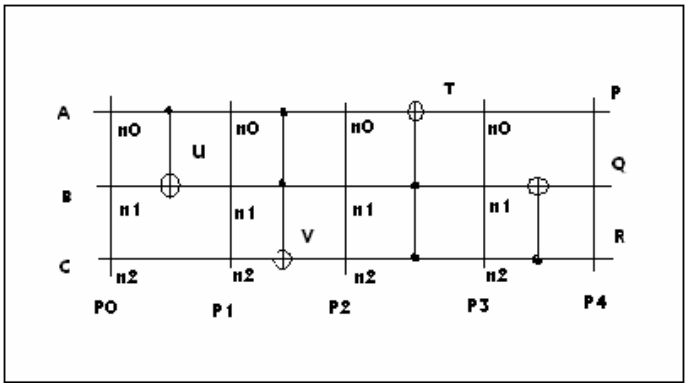


Fig. 2. Example of a binary reversible circuit

In Fig. 2 A, B and C are the basic input wires. P0, P1, P2, P3, P4 are the partitions such that only one gate is covered per partition. P, Q, R are the final circuit outputs. The output table representing the logical values at each node of every partition can thus be written in the form of an array, ***out[P][N],*** where N is the number of wires, and P is the number of partitions.

Algorithm for Fault Localization

1) The fault table is created and updated incrementally together with generating the adaptive tree. It is built as the tree is expanded from the root to leafs by adding new nodes (this is a free tree, variables are not ordered). This fault table describes only the s-a-0 faults covered by a particular test vector. For every level we update this fault table to represent only the remaining uncovered s-a-0 faults for the entire circuit.

2) Level 0: For any reversible circuit, test vector T0 detects all s-a-1 faults. Hence we always choose T0 for the Level 0.

3) For all Levels ahead :
   a) Select a partition for which the fault table shows maximum number of uncovered faults and mark that partition as checked.
   b) Find the counter test vector at that partition.
   c) Backtrack in the circuit from this point using the backward simulator function, to find the corresponding input test vector.
   d) Apply this input test vector to the given circuit to find the output using the forward simulator function.
   e) Get the output table : out[P][N]
   f) Check if this input test vector divides the uncovered faults (looking at the fault table) from the previous level into half.
   g) If so, the test is good. Then check if the same test holds good for all other nodes in the same level.
      - if not then discard the test. Go to step h)
      - if good go to step i)
   h) If the test is not good,
      - If all partitions are not checked, then choose a partition which is next maximum and repeat step b) onwards.
      - If all partitions are checked, then choose the input test vector which divides the uncovered faults into nearly half subsets.
      - Update the fault table by marking the covered faults by this chosen test vector. Go to step i)
   i) Repeat steps a) through i) until all distinct s-a-0 faults except one, from the fault table are covered for every node in that level.

6. CONSIDERING EQUIVALENT FAULTS

When the output at two or more nodes of the circuit is identical for any input test vector applied to the circuit, then those nodes are said to be equivalent nodes. In our algorithm we deal with equivalent nodes which are adjacent to each other. In other words these nodes are nothing but a wire separated by the logical partitions P0, P1 etc. For example in the given circuit in Fig. 2 nodes n0 at partitions P0 and P1 are the equivalent nodes. In our incremental fault table we represent the equivalent nodes either by a '2' or '3' depending on its logical value '0' or '1'. While considering the uncovered faults in a partition, we count the equivalent faults too; if uncovered until the moment.

Particulars to be noted

1) If there are N wires, then 2^N is the maximum possible input test vector. We restrict ourselves to a certain number of test vectors as we choose a test vector depending on choice of partition. Thus, the number of actual possible input test vectors for a particular circuit will depend on the number of partitions. Hence we can actually use only 2^P input test vectors.
2) The major consequence is that while doing so we might loose on some good test vectors at a particular node, which exactly divide the faults into two equal subsets of covered and uncovered faults.
3) Also another consequence is that we might loose on some good test vectors at a particular level, where in same test vector can be applied at every node in that level.
4) The effect of all these is that the adaptive tree will not be balanced.

7. FUTURE WORK

It is assumed in the algorithm that all stuck-at-one faults are covered by a test vector which is all zeros; denoted by T0. But this holds true under the assumption that the circuit under test does not include any NOT gates i.e. inverters. Currently we are redesigning the algorithm to take NOT gates also into account.

In future, we also plan to modify our algorithm to incorporate for fault localization in binary and multi-valued quantum circuits.

8. CONCLUSIONS

The program was tested on several reversible circuits from the literature [10]. Because of the lack of large benchmarks, we have to create such circuits randomly, which is perhaps not a good idea, but nothing better can be done since there are no good synthesizers so far for very large reversible functions. From the examples of the not more than 8 qubit circuits that we analyzed it seems that the symmetric property holds true. The tree obtained is mostly balanced in these cases. Further testing and analysis of data is necessary on larger examples.

REFERENCES

1. K.N. Patel, J.P. Hayes and I. Markov, "Fault testing for reversible circuits," Proc. *VLSI Test Symp.* (*VTS 03*), Napa, CA, pp. 410–416, April 2003
2. U. Kalay, N. Venkataramaiah, A. Mishchenko, D. V. Hall, and M. A. Perkowski, "Highly Testable Finite State Machines Based on EXOR Logic", *PACRIM'99 7th IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Victoria, B.C., Canada, August 23-25, 1999
3. Z. Kohavi, "Switching and Finite Automata Theory, *McGraw Hill*, 1978.
4. A. Mishchenko and M. Perkowski**,** ``Fast Heuristic Minimization of Exclusive Sums-of-Products," *Proc. RM'2001 Workshop*, August 2001
5. C. Bennett, "Logic Reversibility of Computation," *IBM J. Res. Dev.* 17:525-532, 1973.
6. M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.
7. R. C. Merkle. Reversible electronic logic using switches. *Nanotechnology*, 4: pp. 21-40, 1993.
8. R. C. Merkle. Two types of mechanical reversible logic. *Nanotechnology*, 4: pp. 114-131,1993.
9. R. C. Merkle and K. E. Drexler. Helical logic. *Nanotechnology*, 7: pp. 325-339, 1996.
10. D. Maslov, Reversible Logic Synthesis, Ph.D. Thesis, University of New Brunswick, 2003.