Test Set Generation and Fault Localization Software for Reversible Circuits

Dean Pierce, Jacob Biamonte, Marek Perkowski

Portland State University
Department of Electrical and Computer Engineering
Portland, Oregon – 97201, USA

piercede@pdx.edu, biamonte@cecs.pdx.edu, mperkows@ee.pdx.edu

Abstract

We discuss some properties of reversible circuits that allow them to be tested more efficiently than their classical counterparts, and give an analysis of currently proposed fault models. We also present an efficient algorithm that can be used to generate fault localization trees for large circuits.

1 Introduction

Reversible circuits are circuits that have an equal number of inputs and outputs. Mathematically, the inputs of a reversible circuit bijectively map onto its outputs, so no information is lost. Physically, this means that a circuit can potentially operate with extremely low power consumption [9]. Reversible circuits have also been shown to have many practical applications in other fields such as signal analysis, cryptography, code converters, spectral transforms, and image processing. It has also been suggested that classical circuits can be converted to reversible circuits for increased testability, which is due to the improved controllability and observability in the circuit [15].

Although initial interest in reversible circuits is from their low power consumption and reliability in future nano-technologies, their high testability adds one more argument to their acceptance as a viable technology. An important part of the testing of circuitry is determining what kinds of faults are possible, and where they are likely to occur. The stuck-at fault model [2] was an earlier fault model that was proposed for the testing of reversible circuitry. More recently a new fault model [3] was proposed in which a gate might never get activated, and was suggested to be more relevant to the technology.

This paper covers a method to generate **fault tables**, and from them, **fault localization trees**. An algorithm for building fault localization trees is covered in depth, and shown by example. Also proposed is a new method of storing fault tables with decision diagrams, which makes it possible to index fault tables that are much larger than those previously possible.

2 Fault Models

There are many places that a fault could be located at in a circuit, and it is hard to determine which places errors would be likely to occur because there does not exist a commonly agreed upon technology for building reversible circuitry. Similarly, different fault models will need to be introduced based on the physical technology, because it is important to both be able to test for all likely faults accurately, and to have a concise test set so that testing can be performed efficiently. These are the currently proposed fault models for reversible circuitry.

Stuck At

The **stuck at fault model** [2] assumes that there is a problem with one of the horizontal wires on the circuit. A passing bit can either get stuck at value zero, or stuck at value one. This is the only fault model discussed in this paper that destroys the reversibility of a circuit, because data is lost as it passes over the fault. A circuit with **n** bits and **m** stages will have $(m+1)*n$ locations where a fault might occur. Since the stuck at fault model covers two types of faults at each location where a fault might occur, the total number of faults covered by the model is $(m+1)*2n$. Depending on the layout of the circuit, many of these faults might be equivalent.

Missing Gate

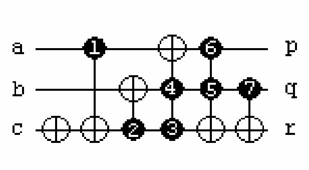
When a control on a controlled gate in a reversible circuit is damaged in such a way that it can never be turned on, the gate that is being controlled can never be activated. An equivalent circuit diagram would look the same, but with that one gate removed. The full set of these faults are known as the **missing gate fault model** [3]. The number of faults in this model will be **m**, one fault for every gate that could be missing

Broken Control

When a control is damaged in such a way that it is always measured as "on", then it is analogous to there just being a wire instead of a control at that part of the circuit. The number of faults in this model is equal to the number of controls in the circuit. This fault model has not yet been introduced in the literature, but will be referred to in this paper as the **broken control fault model**. This fault model has also been referred to as the "missing control" fault model.

3 Fault Table

Traditionally, the first step towards determining an appropriate test sequence for a given circuit is to build a complete fault table [17]. For didactic reasons, we will apply this concept for reversible circuits below. A fault table is a set of truth tables that spans a given fault model. Each column in the table represents a possible variation of the circuit based on where the fault is located, and what type of fault it is. Identical columns (equivalent faults) are combined, thus reducing the number of columns in the table. The resulting table will have 2^n rows, where n is the number of bits in the circuit. The number of columns in the table is determined by the number of applicable faults based on the fault model, plus one for the good circuit. For reversible circuitry, each entry in the table is a number between 0 and $2^n - 1$, which represents the output of the good or faulty circuit.



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

input	output
000	111
001	000
010	001
011	011
100	100
101	010
110	110
111	101

Figure 1 - 3_17 gate

Figure 2 - Matrix representation of a 3_17 gate

Figure 3 - Truth table

As an example, we will present a fault table for a 3_17 [19] circuit using the broken control fault model. **Figure 1** shows the circuit diagram for a 3_17 circuit in quantum array notation. All of the controls have been labeled accordingly. Since the **broken control fault model** is being used, each control marks a place in the circuit where a fault might occur. In **Figure 2**, the circuit is shown as a permutation matrix. From this matrix, a truth table can easily be derived, as shown in **Figure 3**. The truth table lists the inputs of the circuit, and then lists the appropriate output.

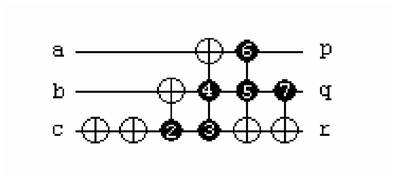


Figure 4 - 3_17 gate with fault

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 5 - Matrix representation of a 3_17 gate with fault

input	output
000	000
001	111
010	011
011	001
100	100
101	010
110	110
111	101

Figure 6 - Truth table with fault

If fault occurs on the first control, it would be as if the control had "disappeared". Notice that the second stage of the circuit has changed in **Figure 4** when compared with **Figure 1**. The matrix that would describe the function would change is shown in **Figure 5**, and the corresponding truth table is shown in **Figure 6**.

By calculating the truth tables for every possible fault in the fault model, the fault table is generated. The complete fault table for the 3_17 gate with the broken control fault model is shown in **Figure 7**. All of the binary values have been replaced by their decimal equivalents. The columns of the table represent where the fault is. The column labeled GC (Good Circuit) is the truth table shown in **Figure 3**, and represents the truth table if there are no faults in the circuit. The rows correspond to the inputs.

	GC	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	7	0	7	7	7	7	7	7
1	0	7	3	0	0	0	0	1
2	1	3	1	1	5	1	1	0
3	3	1	0	6	3	3	2	3
4	4	4	6	4	4	5	4	5
5	2	2	2	2	2	2	3	2
6	6	6	4	3	6	6	6	6
7	5	5	5	5	1	4	5	4

Figure 7 - Fault Table for 3_17 circuit covered by the broken control fault model

Building and minimizing complete fault tables [17] is usually avoided because of the exponential increase in the number of input bits that are added to a circuit. Such computationally difficult data structures can be stored much more efficiently as decision diagrams, and still have the quick indexing capability that allows it to be called a fault table. The software that was developed for this project is capable of efficiently storing fault tables with up to 2^{300} virtual rows using decision diagrams [1], which can be indexed as if the entire column were actually created.

Fault tables like this can be generated for non-reversible circuits, although the restricted numbers of outputs in many irreversible circuits reduce their testability. For each irreversible circuit F, we will be able to compare its testability and fault localization ability with its reversible equivalent F_{rev} obtained by converting the irreversible circuit into a reversible circuit.

4 Fault Localization Tree

The localization of a fault in a circuit can be done by sending a sequence of input values as tests, and observing the corresponding outputs. With this method, one can determine the validity of the circuit, or location of the fault, in a relatively small number of tests compared to the classical fault diagnosis methods, which utilize binary branching in their localization trees [15,17]. A good way to represent a set of test sequences is with a fault localization tree.

A **fault localization tree** (**Figure 8**) is a generic base tree abstraction. Each node contains a suggestion of what test to select next, and it is traversed based on the output value obtained by the circuit. The tree in **Figure 8** is constructed from the fault localization table in **Figure 7**. A circuit that has been physically manufactured may contain errors, and with the aide of a fault localization tree, those errors can be quickly localized. The 3_17 circuit is not very complex, but the methods discussed here can be applied to circuits of large size.

In **Figure 8**, the circles represent the suggested input. Below the circle is a list of possible answers. The tree is then traversed until a square is reached, which represents the location of a fault in the circuit.

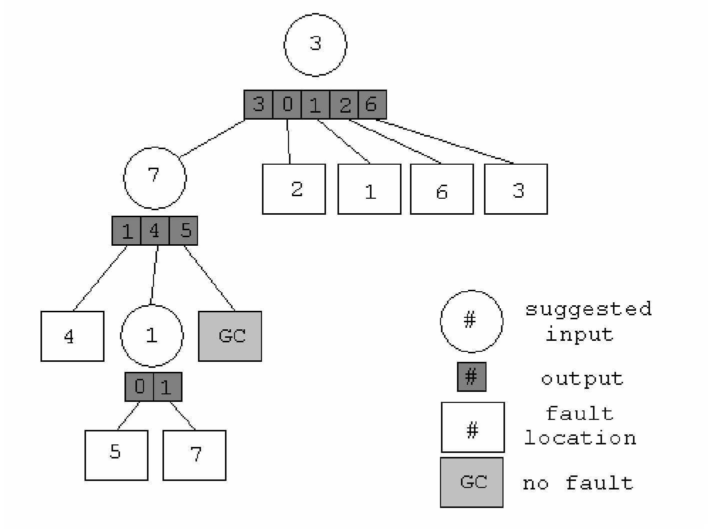


Figure 8 - Fault Localization Tree
for the fault table shown in Figure 7

As an example of how a fault localization tree is used, imagine that a 3_17 circuit has been manufactured where the fourth control is permanently on. When the circuit is tested, the first input is 3. Since the fourth control is missing, the output will be 3. This result can be observed in **Figure 7**. The tree is then traversed down to the next suggested input, which is 7. The output is 1, so the fault has been localized at the fourth control.

It is important to realize that a single fault table can be used to generate many different fault localization trees. Many algorithms are known from the literature which can be used to generate fault localization trees, but few are non-binary, and none are for truly large **circuits under test**.

5 The Tree Generation Algorithm

Observing the large range of outputs for any test, trees can be made that are exponentially shallower than the trees with binary branching. As a proof of concept of this idea, we created a software package named RFault that takes decision diagram encoded fault tables, and converts them to fault localization trees (as shown in **Figure 8**), which are also called adaptive trees [15,17].

An important part of RFault uses a recursive algorithm to break down the list of faults until each fault has its own unique test sequence. The great advantage to this algorithm is that any given test has the potential to break the list down into up to 2^n new lists, where **n** is the number of bits in the circuit. This means that using this type of localization, the tree is shallower than commonly used methods that would only branch twice per node.

The recursive insertion algorithm that builds the fault localization tree has four stages. There is the “leaf” stage, the “best test” stage, the “re-sort” stage, and the “recursion” stage. The function will get called one time for every node in the tree, and will end its current branch of recursion if it has generalized a path that will localize a fault. The following pseudo code describes the recursive algorithm that is used to generate the fault localization tree.

5.1 The Leaf Stage

During the leaf stage, the algorithm calculates how many faults are listed in the list of faults. If only one is listed, then the current node becomes a leaf, and this branch of the recursion is completed.

5.2 The Best Test Stage

There is more than one entry in the fault table to localize between, so now it is the task of the program to determine which next test should be selected. There are many ways that this can be done, but RFault uses one of two methods.

If the good circuit is in the list of faults, RFault checks each possible input against the current list of fault entries, and decides upon the test that results in the output of the good circuit having the least in common with the outputs of the other circuits. With this method, the size of the list containing the good circuit is reduced to a list of one element as soon as possible.

If the good circuit is not in the list, RFault will select the test that gives the most diverse output, thus causing the tree to split into the maximum number of branches. The ideal case would be to find an input where the outputs for all remaining fault entries.

In both of these cases, if multiple tests are equally valuable, then one is chosen at random. This will rarely result in a minimal tree, but it prevents the overhead that would be required by a backtracking algorithm. The effects can be balanced out by constructing the tree multiple times (using random choices), and selecting the best result. In general, it is best to first select the tree based on the shortest number of tests to verify the good circuit, and if multiple trees have the same number of tests to verify the good circuit, then the tree with the smallest average number of tests should be selected.

	GC	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	**7**	0	**7**	**7**	**7**	**7**	**7**	**7**
1	**0**	7	3	**0**	**0**	**0**	**0**	1
2	**1**	3	**1**	**1**	5	**1**	**1**	0
3	**3**	1	0	6	**3**	**3**	2	**3**
4	**4**	**4**	6	**4**	**4**	5	**4**	5
5	**2**	**2**	**2**	**2**	**2**	**2**	3	**2**
6	**6**	**6**	4	3	**6**	**6**	**6**	**6**
7	**5**	**5**	**5**	**5**	1	4	**5**	4

Figure 9- Truth table for the 3_17 circuit with the broken control fault model.

As an example, the entire fault table shown in **Figure 7** can be looked at to determine which test would be the best. The good circuit is listed in the list of possible faults, so the first method must be used. All outputs matching the good circuit have been bolded for emphasis in **Figure 9**. The goal is to find the test to isolate the good circuit as early in the tree as possible, so counting along the rows, the value of each row can be determined. If the input is zero, and there are no faults in the circuit, the output will be 7. There are six other columns in the fault table where the result will also be 7 when provided with the input of zero.

Repeating this process, it is noticed that if the input is a 3, then there are only three other columns covered by the broken gate fault model which match with the output of the good circuit. Since this test distinguishes the good circuit column as much as possible, the test 3 is selected as the **best test** for this set of faults. This decision is reflected in Figure 8, where 3 is the first suggested input test to be applied.

5.3 The Re-Sort Stage

Once the algorithm has calculated the best test, it is then used to break down the current list of fault entries, into a set of smaller lists of fault entries. RFault will go down the list of fault entries, and sort them out based on their output when given the **best test**. The number of new lists is the number of times the current node will split.

In the example, 3 was selected as the best test. By looking at the fault localization table, it is observed that the input of 3 will produce one of five distinct outputs. This means that the list of faults is split up into five smaller lists of faults. These smaller lists are shown in **Figure 10**.

	GC	**4**	**5**	**7**
0	7	7	7	7
1	0	0	0	1
2	1	5	1	0
3	**3**	**3**	**3**	**3**
4	4	4	5	5
5	2	2	2	2
6	6	6	6	6
7	5	1	4	4

Output =3

	2
0	7
1	3
2	1
3	**0**
4	6
5	2
6	4
7	5

Output=0

	1
0	0
1	7
2	3
3	**1**
4	4
5	2
6	6
7	5

Output=1

	6
0	7
1	0
2	1
3	**2**
4	4
5	3
6	6
7	5

Output=2

	3
0	7
1	0
2	1
3	**6**
4	4
5	2
6	3
7	5

Output=6

Figure 10 - New lists of faults, divided based on output of the test 3

5.4 The Recursion Stage

The algorithm now calls itself on each of the new smaller lists. Notice that the sum of the numbers of elements in each of the smaller lists is equal to the number of elements in the larger list. This whole process takes very little memory, and can be done at relatively high speeds.

6 A New Way to Store Fault Tables

Classically, a Fault Table would be stored as an array of arrays. It has been noted, however, that the functionality of a circuit can be represented in the form of a matrix, and matrices can be stored as decision diagrams [1]. If the fault table were stored as an array of decision diagrams, the amount of memory needed to store an entire fault table would be drastically smaller than that required to store a standard fault table as an array. Each column would be stored as a decision diagram instead of an array with 2^n entries.

The decision diagrams can be multiplied together, compared, and indexed very efficiently, all with minimal memory usage [1]. By taking advantage of these features, there is never a point where 2^n operations need to be done, allowing for much larger fault tables can be built.

7 Conclusion

We showed a method to create trees of test sequences used to localize a fault in a reversible circuit. The method based on the Fault Table is general and does not depend on any particular fault model.

Reversible circuits have been shown to be more testable because the maximum amount of relevant information is gained from a single test. Since more information can be gained, then the number of branches in the tree is much larger than can be obtained from nonreversible circuits. If the number of inputs of a circuit is larger than the number of outputs, information is lost, requiring one to perform more tests to gain the same amount of information about the circuit.

Using the algorithms discussed in this paper, especially the use of decision diagrams in fault table construction, very efficient fault localization systems can be created. Although fault localization trees have a great advantage when localizing faults in reversible circuits, this method can be applied to test any digital circuit technology.

Future research will involve a comparison of minimal test sets and fault localization trees for irreversible circuits of large size, and their counterpart reversible circuits obtained from irreversible-to-reversible conversion.

8 Acknowledgements

George F. Viamontes made modifications to QuIDDPro adding special functions so it could be used for this project. He also provided invaluable guidance which led to many large advances in our research. Portland State University provided funding, space and materials needed to realize this project.

9 References

1. G. F. Viamontes, I. L. Markov, and J. P. Hayes, Improving gate-level simulation of quantum circuits, *Quantum Information Processing, vol. 2(5), pp. 347-380, October 2003 http://xxx.lanl.gov/abs/quant-ph/0309060*
2. K. N. Patel, J. P. Hayes and I. L. Markov, Fault Testing for Reversible Circuits, *quant-ph/0404003*
3. J.P. Hayes, I. Polian and B. Becker, Testing for Missing-Gate Faults in Reversible Circuits, *13th Asian Test Symposium (ATS'04)*
4. H-J. Wunderlich, S. Hellebrand, The Pseudo-Exhaustive Test of Sequential Circuits, *Proceedings IEEE International Test Conference, Washington, DC, 1989*
5. C. Landrault, Test and Design For Test, *www.ee.pdx.edu/~mperkows*, Translated by M. A. Perkowski
6. J. Biamonte, M. Perkowski, Principles of Quantum Fault Detection, *Portland State University INQ research conference, June 08, 2004*
7. J. Biamonte, M. Perkowski, Principles of Quantum Fault Diagnostics, *to appear in McNair research Journal,* Issue 1, Volume 1, 2004
8. E. McCluskey and Ch-W. Tseng, Stuck-Fault Tests vs. Actual Defects, 1997
9. M. Nielsen, I. Chuang, Quantum Computing and Quantum Information, *Cambridge University Press, 2000*
10. C. Williams, S. Clearwater, Explorations in Quantum Computing, *Springer Press, 1997*
11. R. C. Merkle, Reversible electronic logic using switches, *Nanotechnology, 4: pp. 21-40, 1993*
12. R. C. Merkle, Two types of mechanical reversible logic, *Nanotechnology, 4: pp. 114-131,1993*
13. Fredkin, T. Toffoli, Conservative Logic, *MIT Laboratory for Computer Science 545 Technology Square, Cambridge, Massachusetts 02139*
14. W. Zurek, Reversibility and Stability of Information Processing Systems, *Physical Review Letters, Vol. 53, pp. 391-394, 1984*
15. K. Ramasamy, R. Tagare, E. Perkins and M. Perkowski, Fault localization in reversible circuits is easier than for classical circuits, *Proceedings of the International Workshop on Logic and Synthesis, June 2004.*
16. P. Beraldi, A. Ruszczyński, The Probabilistic Set Covering Problem, *Operations Research © 2002 INFORMS, Vol.50, No.6, November–December 2002, pp. 956–967*
17. Z. Kohavi. Switching and Finite Automata Theory. McGraw-Hill, 1970.
18. Friedman, A. D., and P. R. Menon, Fault Detection in Digital Circuits, Prentice-Hall, 1971.
19. D. Maslov, Reversible Benchmarks, http://www.cs.uvic.ca/~dmaslov/