ATPG for Reversible Circuits using Technology-Related Fault Models

Jeff S. Allen[1]
[1]Department of Electrical Engineering, Portland State University, Portland, Oregon, 97207-0751, USA
jeffal@cecs.pdx.edu

Jacob D. Biamonte[1,2]
[2]Portland State University Department of Physics Portland, Oregon, 97207-0751, USA
biamonte@ieee.org

Marek A. Perkowski[1,3]
[3]Department of Electronics and Computer Science, Korea Advanced Institute of Science and Technology, Daejeon 305-701, South Korea.
mperkows@ee.pdx.edu

ABSTRACT
We address the problem of test set generation and test set reduction, to first detect, and later localize faults occurring in reversible circuits. Reversible Computation has high promise of low power consumption. Some new fault models are first presented here. An explanation of the new fault models is made based on a physical realization representing the state of the art in the reversible CMOS circuit technology. Evidence is then presented showing that the fault models presented in the current literature are not adequate for existing realizations of reversible logic such as CMOS. We designed a ATPG software package with a friendly graphical user interface to aid experimentation with various fault models. The purpose of this work is to give an overview of our findings and pave the way for a later paper fully addressing the CMOS fault models. The key experimental results are presented.

Categories and Subject Descriptors
D.3.3 [**Verification**]: Design validation and verification, design experiences, emerging technology.

General Terms
Algorithms, Reliability, Verification

Keywords
Test Set Generation, Reversible Computing, CMOS Technology

1. INTRODUCTION

According to Landauer's principle [1], it is possible to construct a computer that dissipates a subjectively small amount of heat. A necessary condition is that no information is lost in the process of propagating logic values from input to output of the circuit. Therefore, logical reversibility represents an important and necessary subclass of computational apparatus that has, in recent times, gained much interest from the EDA community; with the preliminary results reported in [2]. For an introduction on reversible circuits we refer the reader to Markov [3][5].

Various reversible technologies have been proposed, such as Optical [5, p. 287], Quantum [6] CMOS [4], Electrostatic[22], and other switched technologies. Based on the current fabrication capabilities the CMOS realization offers the most immediate implementation prospects, although large power reduction will be gained only in future generation technologies. For example, a 5 bit carry-look-ahead adder has been physically built with results presented in [7]. Although reversible circuits in CMOS technologies do not offer significant losses in power consumption, the promise of low power consumption still makes them very intriguing [8][9][10][11]. Surprisingly little work has been done however to address the errors present in reversible technologies. The main published results are in [12][13][14][23]. In [12] a study is presented that extends the classical stuck-at fault model to reversible circuits. In [14] the missing gate fault model is presented. Here we present a comparison of these models, along with two new fault models; one to complement a previous model and another that addresses the needs of actual technology. We relate these new fault models to logical testing of the dual line switched technologies specifically CMOS technology presented by De Vos in [4] and [7]. It has to be pointed out that these circuits have been fabricated and proven operational in contrast to many other reversible circuit proposals that were adiabatic-reversible rather than physically reversible rather than physically reversible (with no backward mapping) or pure theoretical speculations. We simulated the De Vos circuits using Cadence tools and proved that they operate both from input to output and output to input, being thus truly physically and not only logically reversible. We believe that the current literature on reversible test set generation [12][14], does not include a completely correct fault model, because this model is not good for both quantum and CMOS realizations of reversible circuits. Below, we show realizations of reversible logic circuits in the existing CMOS technology, which demonstrate that the "stuck-at" and "missing gate fault" fault models are insufficient. It should be mentioned that we assume that the switching layout is switch minimized as shown in [4] and therefore path minimized within the gate. Previous research has showed that the fault models from [12],[14] are inadequate also for quantum circuits.

The first thing that must be mentioned about De Vos reversible CMOS circuits is that they have no clock, no ground and no VDD on chip, thus the classical stuck-at concept is not correct for them. Every input, intermediate and output logic variable is repeated twice, as a signal and its negation. To explain the functionality of reversible CMOS realizations, consider Figure 1 (a.), where a signal is represented by both a high and a low component, such that a high signal is [1, 0] and a low signal is value [0, 1]. Gates in this technology flip these signals back and forth, never losing information. However, a stuck at fault internal to the CMOS CNOT gate itself can result in output states such as [0, 0], [1, 1],[Z, Z] and float. The logical test set of the circuit is to set *f* to

value high, toggle the input and then set *f* to low and allow a signal to pass un-changed. This is different than in classical logic circuits discussed by Reddy and past researchers in the field [19,20,21] where every (two-input) EXOR gate in the circuit should have 4 tests (00,01,10,11). The EXOR gate here is not a general function that can fail to all functions other than XNOR [19], but one particular realization built using CMOS switches. The fault model is that each switch can be stuck-short (thus removing the literal), or stuck-open (thus removing the entire connection path from input to output. In Figure 1 (b.) we present a model that is conceptually equivalent to Figure 1 (a.). We denote the complement line of A_k as A_k' and the corresponding output as P_k and P_k', respectively. To logically test this circuit we perform the same set of tests as we did for the circuit from Figure 1 (a). Automatic Test Equipment (ATE) must distinguish correct output values from incorrect output values, so for the circuit given in Figure 1 (a.) any time a value of [0, 0], [1, 1], [Z,Z] or float appears on the output there is a fault in the circuit. To localize we are thus propagating the incorrect states [0,0] and [1,1] through the gold circuit, analogously as path propagation algorithms do with values like stuck-at-0 and stuck-at-1.

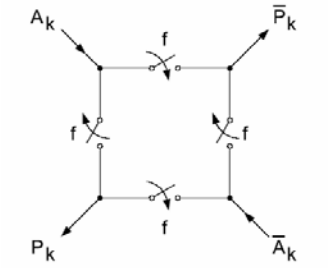

Figure 1 (a.) CNOT GATE, its CMOS realization: The switches are constructed with CMOS transistors; each switch is composed of one n-MOS transistor in parallel with one p-MOS transistor (forming together a transmission gate). When *f* is high, the A_k is routed to P_k' and A_k' is routed to P_k. When *f* is logic 0 the output is A_k routes to P_k while A_k' routes to P_k. An alternate way to describe this circuit is $P_k = f \oplus A_k$, where $\oplus$ represents EXOR.

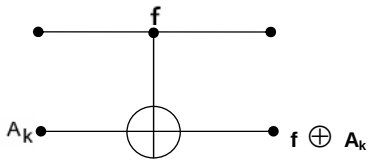

Figure 1 (b.) CNOT GATE: An equivalent to the circuit shown in Figure 1 (a.). This represents the circuit at the level known as the quantum circuits model or program abstraction. When *f* is high, the output of the circuit is inverted. When *f* is logic 0 the output is $P_k = A_k$. An alternate way to describe this circuit is $<P_k, P_k'> = <f \oplus A_k, f \oplus A_k'>$, where $\oplus$ represents EXOR.

2. Fault Models

Recently two fault models for reversible circuits have been introduced, "missing gate" [14], and "stuck-at" [12]. To test for missing gates, k control lines must be set to logic 1. In many cases, full coverage can result in one single test. In [14], in addition to an ATPG method a DFT method is presented in [14] to help reach this lower bound. Using the missing gate fault model for the circuit shown in Figure 2 (a) we apply input test vector (1,1,1) for full coverage. However, this does not account for a control line that is always activated, or from Figure 1 (a.) a switch *f* that is stuck short.

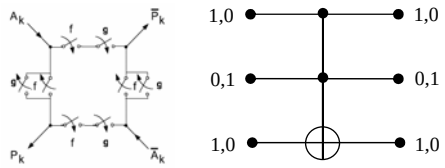

Figure 2 (a.) CCNOT Gate Under Test: An input test vector of (1,0,1) is followed by input test vector of (0,1,0), this test set is complete for the stuck at model, but fails to 'turn on' the CCNOT gate.

The "stuck-at" model in many circuits can be covered in two tests, leaving all gates never activated. For this to happen all CCNOT gates must have k > 1 control lines and set these lines in such a way that the inverted values of the control lines still do not turn on the gate. Assume a test set for stuck-at model that covers all of the faults in two tests, for every vector from this set all control bits when applied as well when inverted do not turn the gate on. Please look now at Figure 2 (a.) and Figure 2 (b.) for examples of complete equivalent test sets for the stuck-at model. Consider Figure 2 (a.) again so we can explain the structure of a CCNOT gate. For the CCNOT gate we simply create two sets of duplicate logical paths, one pair of one *g* switch in series with switch *f* to create two conjunctive paths (top and bottom), and another pair with one *g'* switch in parallel with one *f'* switch forming disjunctive paths (left and right). We must have a logical AND of *g* and *f* to swap the target A_k, but the test set from both Figure 2 (a.) and Figure 2 (b.) misses a critical path within the gate. Thus the stuck-at model proposed in [12] as well in [23], are both incomplete for CMOS technology. In other words, the logic value A_k [1,0] is never swapped to its inverse based on the path the input tests (1,0,1) and (0,1,0) take. This means that the stuck-at model is not complete for CMOS technology.

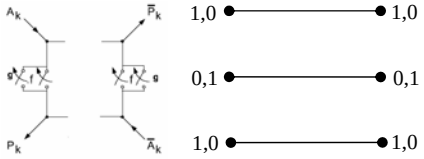

Figure 2 (b.) Missing CCNOT Gate Under Test Containing Fault: An input test vector of (1,0,1) is followed by input test vector of (0,1,0), this test set is complete for the stuck at model.

The other previous fault model under investigation the "missing gate" fault model (MGF)[14], can be covered in some circuits in as few as one tests [14], making it an attractive model. This model checks to ensure that the gate can be activated. In dual-line switched technologies this fault model checks the existence of each gate's inverting path. Essentially it ensures that the gate can be "turned on", it does not however ensure that the gate can be "turned off" and therefore this model does not distinguish a NOT gate from a CNOT, CCNOT, etc. Consider again Figure 2(b) with the open along the NOT path, the MGF will uncover this fault. It should be obvious to the reader that with only two of four+ paths checked the test is not complete for the gate.

With the "missing gate" fault model covering the inverting path, and the "stuck-at" fault model testing a minimum of one non-inverting paths it would appear that the "missing gate" and "stuck-at" models are complementary and when merged would be sufficient. Together the two faults test that each gate can turn on and off both paths separately, and that both logic values can be produced. The fact that the gate is capable of separate traversal of both paths ensures that no switch is "stuck-short". To assist in analyzing the cover produced by the two models we define and explain the first fault model used in this work:

Lock Gate Fault (LGF): Ensures the existence of a non-inverting path along NOT gate, the logical complement to the "missing gate" fault model.

The Lock Gate Fault (LGF) is essentially the inverse to the "missing gate" and is always covered by the "stuck-at". There exist no two MGF tests that are the inverse of one another. This means that the LGF model is included within the "stuck-at" model. This does not take away from usefulness in investigating fault models for reversible circuits as it does represent the existence of a path along one half of the four data paths for any gate and completes the test for "stuck-short" switches.

To further assist in analyzing the two fault models and their effect on the gate under operation, we have added fault models directly related to dual line technology switches. In doing so we introduce the second fault model used in this work:

Control Switch Stuck Fault (CSSF): Broken control switch, either stuck-short or stuck-open.

Vector<f,g,A>	CSSF (f @ 1)	CSSF (g @ 1)	CSSF (f @ 0)	CSSF (g @ 0)
0,0,dc				
0,1,dc	X			
1,0,dc		X		
1,1,dc			X	X

Figure 3 CSSF CCNOT Gate Fault Table: Only one test exists for any Control Switch Stuck Fault. As the size of k lines increases the gate gains only one test, an extra "one cold" test in order to focus on a single switch set.

Although the Control Switch Stuck Fault is one in which a switch is either stuck-open, stuck-closed, or missing, it is denoted by Control-Switch-Stuck-at-1, *CSSF-@-1*, and Control-Switch-Stuck-at-0, *CSSF-@-0*. This is to help localize to which path the error belongs. Control-switch stuck-at-0 can be either a stuck-open along one of the two (active) inverting paths, or a stuck-short along one of the two (inactive) non-inverting paths. Whereas the Control-Switch-Stuck-at-1 is either; a stuck-short along one of the two (inactive) inverting paths or a stuck-open along one of the two (active) non-inverting paths. The CSSF, Figure 3, unfortunately lacks the forgiving properties of the previous fault models and there are 2 * (k + 1) tests existing and (k + 1) required tests to test a k CCNOT gate. On the positive side there is a benefit of not ever caring about the value on the controlled line. The CSSF-@-0 test vectors are identical to those for the MGF.

3. ALGORITHM

To analyze the fault models a reversible circuit testing program was written to test for all "stuck-at", "missing gate" and the previously introduced "lock gate" models. To represent the physical implementation for dual-line reversible technology we add the previously mentioned control switch stuck fault model. The goal of the software package was two-fold, one to quickly create near optimal test patterns for the three fault models, the second to check the actual cover for the technology at the gate's internal switches.

The circuit is represented as a netlist containing fault information through which we propagate signals bi-directionally, to build test vectors as well as detect faults. The state of our circuit is represented by a binary input vector, and the gates act on this vector as operators would in the state space of the system.

Test Vector	In Stk @ 0	In Stk @ 1	MGF	LGF	Out Stk @ 0	Out Stk @ 1
00		X		X		X
01	X			X	X	
10		X	X		X	
11	X		X			X

Figure 4 (a), A single CNOT Fault Table: As can be seen with this fault model any test covers half of the faults, but the overlap is in such a way where the first test will get 3, the second test 2, and either remaining test will cover the last one fault.

Test Vector	In Stk @ 0	In Stk @ 1	MGF	LGF	Out Stk @ 0	Out Stk @ 1
000		X		X		X
001	X			X	X	
010		X		X		X
011	X			X	X	
100		X		X		X
101	X			X	X	
110		X	X		X	
111	X		X			X

Figure 4 (b.) CCNOT Gate Fault Table: This is like the single k line CNOT example except there are two sets of three equivalent tests. The set of equivalent tests increases, as k increases the number of equivalent tests increases to two sets of $2^{(k+1)} - 2$ /2 equivalent tests. The MGF tests however, do not gain equivalent tests and always have only two vectors to cover. This makes those Toffoli gates that have the most control inputs the most critical to test. The fact that there are extra equivalent tests to cover the LGF model and the only requirement is applying a test that is not testing for the "missing gate" model, one just needs to set any control bit low. For example <*, 0, *> or <0, *, *> are adequate for setting the LGF test vector.

As can be seen in Figure 4(a), any arbitrary first test vector will cover 3 faults, with the second test vector will cover exactly 2 faults, the final fault covered by either of the two remaining test vectors. Fault coverage is determined by the interaction of the test vector as it propagates through the netlist representing the circuit. The propagation acts in both directions setting minimal bits to test toward a prioritized list of the possible faults covered until a full test vector is created. During localization the error result is propagated in the reverse direction, *making use of the intrinsic properties of reversibility*, and the error is compared against the correct vector for each stage. This method is surprisingly fast, loading and running all *.tfc circuits on Maslov's site [18] in less than 2 minutes and generated a complete set for a 1500+ gate circuit, hwb9-1544, in less than 1 minute, *using a 1.8Ghz w/512 MB P4 computer*. The speed of our test set generation and its efficiency are due to heuristics based on the analysis of test vector coverage patterns, common AI programming techniques, along with the luxury that absolute minimal cover is not a requirement for this study.

To understand the basics of the heuristics used for prioritization and selection of test vectors we look to Figure 4(a) and Figure 4(b) displaying test vector coverage for the CNOT and CCNOT gate respectively. As can be seen in Figure 4(a) any three of the four possible tests are required to cover a CNOT gate. As the number of control lines is increased by one to a CCNOT gate Figure 4(b), an interesting growth pattern emerges. The number of vectors covering the MGF fault is always two whereas the number of test vectors covering the LGF fault increases. Each of the LGF test vectors from 4(a) is repeated $(2^{(k)} - 1)$ times and requires only the setting of one bit to test. This makes MGF a more infrequent test as the number of controls increases, while for the other fault models the number of vectors that can be used as tests increase with k. The nature of the growth as well as the ease of testing the LGF faults are important factors for the order of selecting test vectors for this path propagating technique. This order for test vector prioritizations is as follows:

1. Select vector for Missing Gate Fault model [always only 2 possible tests].
2. Select vector for Lock Gate Fault model [$2^{(k+1)} - 2$ possible tests].
3. Vectors are selected towards any of remaining gate-wise tests for untested vectors, of coverage one, generally input s0 or s1 or output s0 or s1. [$2^{(k+1)}$ possible tests].

For a circuit with i input lines, at any stage with k control lines there exist exactly $2^{(i - k)}$ possible input vector variants that will cover the same stage-wise test vector, example; A CNOT gate, a two input two output gate, existing within a four input circuit has two lines that are not used by the CNOT gate two don't cares, leaving four vectors combined with the stage test vector from figure 3(a). This relation between the number of input lines and control lines as well as the sparsity of the MGF test as k increases illustrate the MGF for the stage with the most control lines as a critical path.

Below more details of our algorithm are presented. Paper [12] used an algorithm which breaks the circuit into smaller disjoint sub-circuit groups and sets bits accordingly for each group, merging the groups to create test vectors. The approach taken here is based on a single signal vector that propagates through the circuit attempting to set a prioritized list of tests while maintaining the largest set of possible tests for the next stage. The signal vector moves forward and backward through the circuit setting appropriate bits for test and leaving as many of the lesser tested lines as possible in the don't care state.

As was shown earlier the first test on a gate will cover exactly three of the six faults, this is universal for all gates in the system and means that the first test will cover half of the faults irregardless of the test vector. Although each test vector has an equal number of faults covered in the initial fault table, the next selected tests are not equivalent so that the variants that cover the most critical remaining faults should be promoted first. Without building the entire fault table, a group of variant tests covering the most critical test can be easily achieved by starting at the stage containing the most infrequent remaining test. This is the largest Toffoli gate still in need of MGF test cover, in cases where all MGF have been previously tested, the smallest Toffoli in need of LGF test is chosen as the stage from which to start, if all MGF and LGF have been tested the first untested stuck-at stage in the circuit is selected as the final "clean up" point.

Once the initial starting stage is selected the program determines which direction to initially propagate the signal vector through the circuit. It does this by performing a *local blocking analysis function*, explained below.

For each line with a control bit at the selected starting stage a quick count of stages to the first CNOT, CCNOT, etc in the circuit is performed. The minimum count for each direction is found, the propagation of the simulated test vector signal will then flow in the other direction, the side of the largest minimum count.

When a CNOT, CCNOT, etc. is encountered the gate must be either turned on or turned off, lines may need to be set without gain of fault coverage. As this is reversible circuit with one to one mappings throughout the circuit, the cost of setting a bit divides the number of possible tests in half for all stages of the circuit. This of course does not apply to handling of the test overlap.

The program then steps through the circuit setting lines for the appropriate test based on the aforementioned priorities. When setting bits for the LGF test, only one bit needs to be set. This bit is determined through a line weighting system which is designed to find which bit will prevent the fewest tests once it is set. In cases where no test can be performed, no bits are set. Irregardless of the existence of a possible test, if the bit under control, the NOT line, has been given a value the gate must either be activated or deactivated in order to propagate. The same line weighting function as used for turning off a gate for the LGF test is used with an additional option of activating the gate.

Once the algorithm reaches either the input or output of the circuit the current vector along with don't cares is simulated back to the starting point. From there the system steps to the opposing end of the circuit. Upon reaching the opposite side, any remaining bits that can be set to test for any remaining stuck-at tests are set appropriately. Any bits still unset are set randomly to fill the test vector.

In cases where the actual circuit's output value differs from the value expected, the software switches its behavior to localize a fault. The software is notified by the experimenter of the difference between the expected output and that of the actual circuit output. The incorrect output is propagated through the system in parallel with the expected output. Assuming a single fault within the system, stages where only the gates for which the controlled bit and bits along the control line are incorrect and have not passed test are flagged as possible errors. This is for localization of MGF, LGF, and "stuck-at" fault models. This method cannot be applied for the CSSF model because for this model the expected output results depend on specifics of implementation.

4. Experimental Results

In this section we first present some data generated by our software package and then illustrate the combined percentage covers for the missing gate, stuck-at, and locked gate faults against the cover of the CSSF for switching technology. In Figure 5 (a) we can see the family of curves approaching full combined coverage for the missing gate, stuck-at and locked gate models. In Figure 5(b) the percentage cover of the same tests

applied to the Control Switch Stuck Fault is shown. To give the reader more exact details see Tables 1,2 and 3 in the appendix.

With the exception of benchmark function xor5d1 none of the complete tests for the combined fault models completely test the functional parts of gates, the switches within the circuit. This is only due to the fact that the xor5d1 is made entirely of Feynman gates. Not surprisingly, a relationship was found between circuits containing gates with a large number of control lines and a decrease in CSSF coverage percentages.

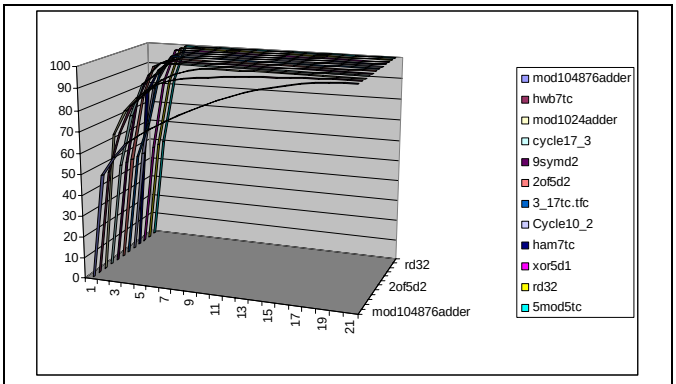

Figure 5(a): Graphical representation of combined MGF, LGF, and stuck-at percentage cover for several members of the Maslov Benchmarks. The X axis represents test count, the Y axis represents the percentage cover.

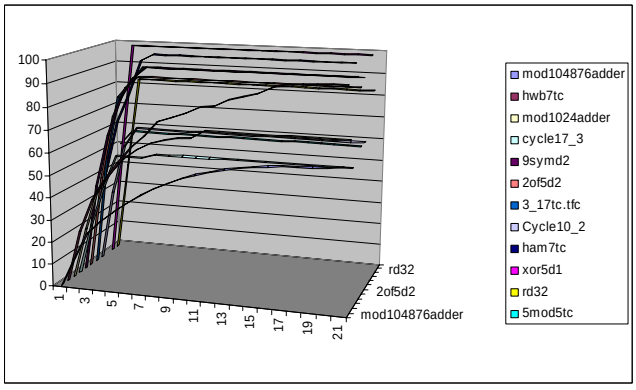

Figure 5(b): Graphical representation of percentage cover for same test vectors applied to the CSSF model for the same reversible functions from the Maslov Benchmarks. The X axis represents test count, the Y axis represents the percentage cover.

Figures 6 (a.) and 6 (b.) show screen shots of a circuit under test. In Figure 6 (a.) one can see a small display window that shows percentage coverage of each of the fault models after each of the fault models. We found that the missing gate and stuck at models miss many faults; note the control squares with green represent the untested switches (the colors are not seen in this text).

Figure 6 (c.) illustrates a selection between the different libraries of gates present in the software working directory. Our software takes a common reversible description language, .tfc used by MMD [18] so it is very easy to generate test plans for new circuits.

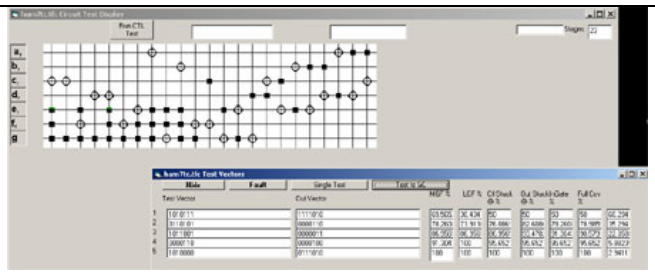

Figure 6 (a.): Screen shot of graphical interface.

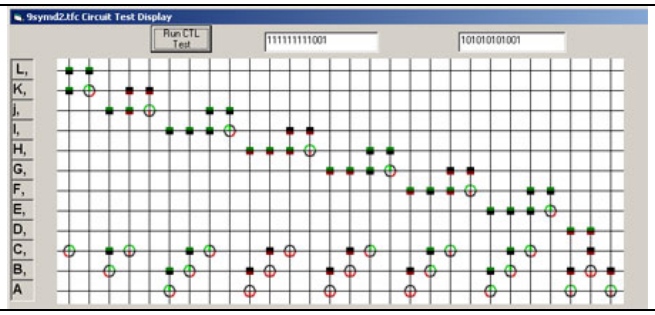

Figure 6 (b.) Stuck high and low faults are represented in green and red respectively. With the gate errors along the vertical axis, and the wire stuck at errors represented in the same manner at the front and back of the CNOT gate. Each vector that propagates though the circuit changes the colors to black if a fault is covered.

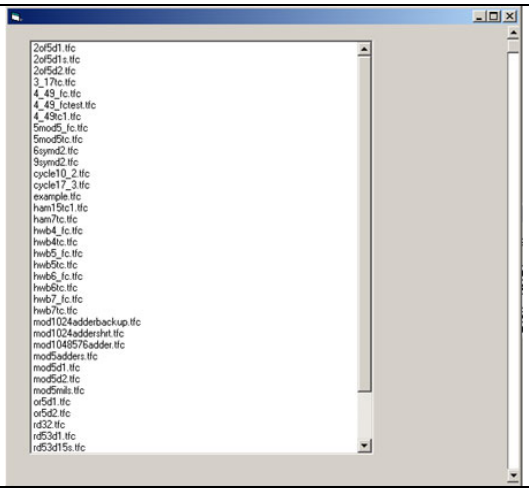

Figure 6 (c.) Selection window for selecting circuits to be tested.

The comparison of tests to cover for the MGF, LGF and Stuck at fault models for Maslov reversible benchmarks [18] is shown in Table 1. Table 2 includes the fault coverage percentage used for visualization of few larger examples. Table 3 represents percentage cover for the test vectors used in Table 2 applied to the CSSF model.

5. CONCLUSION

This paper presents a very practical approach to reversible fault detection, that is based on both old and new fault models we have shown also a direct relation of these fault models to the current CMOS reversible computing technology. For the combined fault models, the software creates test sets larger but more complete than any earlier model alone. However it has also been shown that the combined models are not adequate for dual-line switching

technologies such as CMOS. In order to test these technologies one has to focus on the control switches rather than the concept of a controlled inverter.

In future works we plan to complete the test set generator for the Control Switch Stuck Fault model. This is something of a challenge to program without the use of a fault table as done with the previous models due to the lack of structures that lend themselves to straight forward heuristics. As the possible error output is not a pair of valid complementary words we also plan to use a multi-valued netlist localization against expected complementary pair binary outputs.

6. ACKNOWLEDGMENTS

Portland State University provided funding, resources, space and support for this project. Some support by Ronald E. McNair Post baccalaureate Achievement Program of Portland State University, and the Korean Advanced Institute of Science and Technology, was given during parts of this project. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies of endorsements, either expressed or implied, of the Funding agencies.

7. REFERENCES

[1] R. Landauer: Irreversibility and heat generation in the computing process. I.B.M. Journal of Research and Development 5 (1961) 183-191

[2] E. Fredkin and T. Toffoli: Conservative logic. Int. Journal of Theoretical Physics 21 (1982) 219-253

[3] I. L. Markov, ``An Introduction to Reversible Circuits" IWLS, Laguna Beach, CA, May 2003 (invited), http://www.eecs.umich.edu/~imarkov/pubs/

[4] A. De Vos, B. Desoete, A. Adamski, P. Pietrzak, M. Sibinski, and T. Widerski: Design of reversible logic circuits by means of control gates. In: D. Soudris, P. Pirsch, and E. Barke (eds.): Proc. 10 th Int. Workshop Patmos, Gottingen (Sept. 2000) 255-264

[5] M. Nielsen and I. Chuang, Quantum Computation and Quantum Information, Cambridge Univ. Press, 2000.

[6] D. Vion, A. Aassime, A. Cottet, P. Joyez, H. Pothier, C. Urbina, D. Esteve, M.H. Devoret, Manipulating the Quantum State of an Electrical Circuit, arXiv:cond-mat/0304232, v2 15 Apr 2003.

[7] B. Desoete and A. De Vos. A Reversible Carry-Look-Ahead Adder Using Control Gates. Integration, The VLSI Journal, vol. 33, pp. 89–104, 2002.

[8] R.I. Bahar, J. Mundy and J. Chan, A Probabilistic Based Design Methodology for Nanoscale Computation, ICCAD, 2003, pp. 480-486.

[9] J. Han, and P. Jonker, A System Architecture Solution for Unreliable Nanoelectronic Devices, IEEE Trans. on Nanotechnology, vol. 1, December 2002 pp. 201-208.

[10] J. Kim, J-S. Lee and S. Lee, Implementing unitary operators in quantum computation, Phys. Rev. A 61 (2000) 032312, quant-ph/9908052

[11] J. Kim, J-S. Lee, S. Lee and C. Cheong, Implementation of the refined Deutsch-Jozsa algorithm on a 3-bit NMR quantum computer, Phys. Rev. A 62, 022312 (2000), http://arxiv.org/abs/quant-ph/9910015

[12] K. N. Patel, J. P. Hayes and I. L. Markov, Fault Testing for Reversible Circuits, IEEE Trans. on CAD, 23(8), pp. 1220-1230, August 2004, quant-ph/0404003

[13] J. Biamonte and M. Perkowski, Testing a Quantum Computer, Proceedings of, KAIS, Workshop on Quantum Information Sicence, Seoul Korea, August 29th - 31st, 2004. http://arxiv.org/abs/quant-ph/0501108

[14] J.P. Hayes, I. Polian, B. Becker, Testing for Missing-Gate Faults in Reversible Circuits, Proc. Asian Test Symposium, Taiwain, November 2004.

[15] W. Zurek, Reversibility and Stability of Information Processing Systems, Physical Review Letters, Vol. 53, pp. 391-394, 1984.

[16] D. Maslov and G. Dueck. Reversible Cascades with Minimal Garbage. IEEE Transactions on CAD, vol. 23, issue 11, Nov. 2004, pp. 1497-1509.

[17] C. Landrault, Test and Design For Test, www.ee.pdx.edu/~mperkows Translated by M. A. Perkowski.

[18] Maslov Reversible Logic Benchmarks: http://www.cs.uvic.ca/dmaslov/

[19] S.M. Reddy, “Easily Testable Realizations for Logic Functions,” IEEE Trans. Computers, vol. 21, no. 11, pp. 1,183-1,188, Nov. 1972.

[20] D.K. Pradhan, “Universal Test Sets for Multiple Fault Detection in AND-EXOR Arrays,” IEEE Trans. Computers, vol. 27, no. 2, pp. 181-187, Feb. 1978.

[21] U. Kalay, D.V. Hall, M. A. Perkowski: A Minimal Universal Test Set for Self-Test of EXOR-Sum-of-Products Circuits. pp. 267-276, Volume 49, Number 3, March 2000

[22] R. C. Merkle, Reversible electronic logic using switches, Nanotechnology, 4: pp. 21-40, 1993

[23] E. Perkins et al, Fault Testing for Reversible Circuits is Easier, IWLS 2004.

Name	Model1	Gate Count	Bits	MGF	LGF	I/OSA-0-1	All Models
2of5	GT	18	6	4	2	4	4
2of5	GT	15	6	3	2	4	4
2of5	NCT	12	7	3	2	5	5
rd32	NCT	4	4	2	2	3	3
3_17	NCT	6	3	2	2	4	4
4_49	GT	16	4	3	3	4	4
6sym	NCT	20	10	4	3	5	5
9sym	NCT	28	12	4	3	5	5
4mod5	NCT	8	5	1	2	3	3
4mod5	NCT	9	5	1	2	4	5
5mod5	GT	17	6	1	2	3	3
5mod5	GT	10	6	2	2	4	4
cycle10_2	GT	19	12	2	3	5	5
cycle17_3	GT	48	20	3	3	6	6
ham7	GT	23	7	4	3	4	4
ham15	GT	132	15	11	7	12	12
hwb4	GT	17	4	3	4	4	4
hwb4	GT&GF	11	4	2	2	4	4
hwb5	GT	55	5	6	7	7	7
hwb6	GT	126	6	12	5	11	12
hwb7	GT	289	7	19	11	13	19
mod5adder	GT	21	6	3	2	4	4
mod1024adder	GT	25	20	3	3	5	5
rd53	GT	4	7	2	2	3	3
rd53	GT	12	7	3	2	4	4
rd73	NCT	20	10	3	3	4	4
rd84	NCT	28	15	5	2	6	6
xor5	NCT	4	3	1	2	3	3

Table 1: # of tests required to cover fault. For Fault models “Missing Gate”, “Locked Gate”, “stuck-at”, and combination of all models

Test Count	mod104876-adder	hwb7tc	mod1024adder	cycle17_3	9symd2	2of5d2	3_17tc.tfc	Cycle10_2	ham7tc	xor5d1	rd32	5mod5tc
0	0	0	0	0	0	0	0	0	0	0	0	0
1	50	50	50	50	50	50	50	50	78.985	50	50	50
2	59.047	69.26	66.36	74.305	79.76	80.55	66.66	83	90.579	83.33	83.33	83.33
3	66.11	81.891	78.18	89.93	92.857	95.83	91.66	93	95.652	100	100	100
4	70.158	87.427	84.545	94.44	98.809	98.611	100	100	100	100	100	100
5	73.809	92.56	89.393	99.652	100	100	100	100	100	100	100	100
6	77.222	94.75	93.33	100	100	100	100	100	100	100	100	100
7	80.39	96.309	96.363	100	100	100	100	100	100	100	100	100
8	83.33	96.885	98.48	100	100	100	100	100	100	100	100	100
9	86.03	97.52	99.6963	100	100	100	100	100	100	100	100	100
10	88.49	98.327	100	100	100	100	100	100	100	100	100	100
11	90.714	98.442	100	100	100	100	100	100	100	100	100	100
12	92.698	98.904	100	100	100	100	100	100	100	100	100	100
13	94.444	98.961	100	100	100	100	100	100	100	100	100	100
14	95.952	99.192	100	100	100	100	100	100	100	100	100	100
15	97.22	99.711	100	100	100	100	100	100	100	100	100	100
16	98.253	99.884	100	100	100	100	100	100	100	100	100	100
17	99.047	99.942	100	100	100	100	100	100	100	100	100	100
18	99.603	100	100	100	100	100	100	100	100	100	100	100
19	99.92	100	100	100	100	100	100	100	100	100	100	100
20	100	100	100	100	100	100	100	100	100	100	100	100

Table 2: Percentage cover for combination of MGF, Stuck-at, and LGF models

Test Count	mod104876adder	hwb7tc	mod1024adder	cycle17_3	9symd2	2of5d2	3_17tc.tfc	Cycle10_2	ham7tc	xor5d1	rd32	5mod5tc
0	0	0	0	0	0	0	0	0	0	0	0	0
1	12.987	23.379	22.727	22.493	39.583	42.105	50	32	39.705	50	41.36	50
2	24.707	38.31	41.136	40.953	63.541	71.052	78.571	54.5	64.705	100	83.33	56.578
3	30.292	47.222	49.545	55.623	80.208	84.21	85.714	63.5	77.941	100	83.33	56.578
4	35.259	55.729	55.909	55.623	89.583	89.473	92.857	63.5	94.117	100	83.33	56.578
5	39.642	62.789	60.454	55.623	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
6	43.506	68.287	63.636	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
7	46.883	74.594	65.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
8	49.805	76.851	66.818	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
9	52.305	78.935	67.272	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
10	54.415	82.002	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
11	56.168	82.986	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
12	57.597	85.879	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
13	58.733	87.152	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
14	59.61	89.12	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
15	60.259	91.724	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
16	60.715	93.344	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
17	61.006	93.807	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
18	61.168	94.155	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
19	61.233	94.155	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578
20	62.37	94.155	70.681	57.579	94.791	89.473	92.857	63.5	97.058	100	83.33	56.578

Table 3: Percentage cover for CSSF model using test vectors from table 2, it should be noted that no new test vectors are created after 100% coverage from table 1. First test cover percentages may be high due to the fact MGF covers CSSF at 0.