

# Testing of Combinational Majority and Minority Logic Networks for Emerging Nanotechnologies

Faizal Karim, André Ivanov, Konrad Walus

**Abstract**—In this paper, we present an extension to the existing Path Oriented DEcision Making (PODEM) algorithm to include the ability to generate test patterns for single stuck-at and bridging faults in majority and minority networks. The work presented in this paper is directly applicable to emergent nanotechnologies such as quantum-dot cellular automata (QCA), resonant tunneling diodes (RTD), single electron tunneling (SET) and tunneling phase logic (TPL). A dynamic probability-based controllability heuristic was developed and used as a guide to make intelligent decisions on which lines to justify during the automatic test pattern generation (ATPG) process. Lastly, a genetic algorithm is used to fill-in the unspecified values in the test patterns produced by the ATPG in order to achieve compaction on the final test set size. The modified PODEM algorithm was tested on a set of Microelectronics Center of North Carolina (MCNC) benchmark circuits. In most cases, the size of the resulting test sets using this method was less than 5% of the total number of possible input patterns.

**Index Terms**—QCA, RTD, SET, TPL, ATPG, Testing, Modelling, SSF, SBF

## I. INTRODUCTION

THE availability of nano fabrication processes [1]–[3] and the demand for smaller, more power efficient technology has led to the research and development of a wide-range of novel computing paradigms at the nanoscale. Many of these emergent technologies such as quantum-dot cellular automata (QCA), resonant tunneling diodes (RTD), single electron tunneling (SET), and tunnel phase logic (TPL) are capable of implementing majority and minority logic [4]–[10]. As a result of this, it is important to be able to effectively model and test for the various faults in these majority/minority based logic circuits - particularly in light of the fact that these nano circuits are difficult to realize without defects. Amongst the numerous potential faults likely to occur in such circuits, single stuck-at faults (SSF) and single bridging faults (SBF) represent a large portion of the general faults that occur in these systems [11]–[13]. In the SSF model, a line in a circuit is assumed to be permanently stuck-at-1 (SA1) or stuck-at-0 (SA0). The SBF model is commonly used when a pair of interconnects is assumed to be shorted. Several algorithms have already been proposed to automatically generate test patterns for SSFs and SBFs in CMOS circuits [14]–[18]. However, it remains to be seen whether or not the conventional CMOS test flow can accommodate the testing of nanotechnologies. To this effect, the Path Oriented DEcision Making (PODEM) algorithm was first introduced as a test methodology for nano scale circuits in [19], where the existing PODEM algorithm was modified generate test patterns for detecting SSFs in majority and

minority networks. In this paper, we expand on the work done in [19] by using the PODEM algorithm to also detect the presence of SBFs within each majority gate in a circuit.

In the last several years, testing and design for test of majority/minority networks has generated considerable interest. Most notably, Gupta *et al* [12], [13] have developed a comprehensive test generation methodology using Boolean satisfiability (SAT) to cover both SSFs and SBFs in majority/minority logic networks and have tested their automatic test pattern generator (ATPG) on a set of Microelectronics Center of North Carolina (MCNC) benchmark circuits. Recently, Fang *et al.* [20], [21] developed a PODEM-like ATPG to detect multiple stuck-at faults across different modules in a triple module redundancy (TMR) architecture, and have tested their algorithm on a number of International Symposium on Circuits and Systems '85 (ISCAS-85) and International Test Conference '99 (ITC-99) benchmark circuits. Tahoori *et al.* [22]–[24] have investigated test set generation and C-testability of 1-D and 2-D circuits in QCA and have suggested test-friendly design architectures that help minimize the number of test vectors required for those majority gate based designs. The work presented in [22]–[24] also discusses defect characterization in QCA circuits and the detection of SSFs. Huang *et al.* [25]–[27] considered the testing of 1-D reversible gate arrays under multiple faults and explored how introducing observe and control lines in the design can decrease the size of test sets while adding only a modest amount of overhead. More recently, a strategy for exhaustive SSF testing of combinational QCA logic was presented in [28]. In that work, the authors are able to exhaustively test for all SSFs in an AND/OR QCA circuit using only 2 test vectors by adding  $2n$  additional majority gates ( $n$  = primary inputs). Given the high area overhead of this design-for-test methodology, it is best suited for circuits with a small number of primary inputs (PI) and not particularly applicable for the general case. Lastly, a pseudo-random pattern generator (PRPG) has been used to study the effects of N-detectability of QCA circuits in [29]. The results in their work have shown that this conventional test technique for CMOS designs can also be effective in QCA based designs as well. However, this method requires a much larger number of test patterns to achieve the same fault coverage obtained in both this work and in [12], [13].

The rest of this paper is organized as follows. In Section II we present the implementation details of our PODEM algorithm. The simulation results of our algorithm are provided in Section III and Section IV concludes the paper.

## II. PODEM FOR MAJORITY CIRCUITS

In this section, we discuss the procedure for generating a minimal test set to cover all SSFs and SBFs in majority circuits.

### A. Test Generation Basics

In deterministic ATPG algorithms, there are two main tasks. The first is to excite the target fault, and the second is to propagate this fault effect to a primary output (PO) [30]. In some ATPG algorithms, both the fault-free and faulty logic values are needed, thus composite logic values are used. In these cases, for each signal in a circuit, the values  $v/v_f$  are used where  $v$  represents the fault-free value and  $v_f$  represents the value corresponding to the faulty circuit. In the 5-value algebra proposed in [15], values 0, 1,  $X$ ,  $D$ ,  $\bar{D}$  are considered where 0, 1 and  $X$  are conventional values used to represent false, true and “don’t care”, and  $D$  and  $\bar{D}$  represent the composite logic values 1/0 and 0/1, respectively. This 5-value algebra is used in the PODEM algorithm and will be discussed in further detail in Section II-D. In general, an ATPG algorithm will attempt to propagate the fault-effect ( $D$  or  $\bar{D}$ ) to a PO by recursively justifying all necessary signal lines. During this process, the ATPG algorithm will have to make decisions. If any of these decisions causes a conflict on a given signal line, the ATPG must reverse and change a previous decision. The reversal of a decision is called a backtrack.

### B. Basic Definitions

For convenience, we will only consider non-redundant majority and inverter networks.

**Definition 1** A network  $W$  is considered *redundant* if it contains a fault that is undetectable. Otherwise it is non-redundant [31].

To test any gate input line in a non-redundant majority gate network, we need to sensitize the input by placing the majority gate on threshold, and toggling the input such that a change can be observed at the output.

**Definition 2** A gate input  $i$  is considered to be sensitized if toggling the input toggles the output.

**Definition 3** An  $n$ -input majority/minority gate  $G$  is considered to be on threshold with respect to one of its inputs  $i$ , if the remaining gate inputs are assigned logical values such that  $\frac{n-1}{2}$  inputs hold a logic 0, and the remaining  $\frac{n-1}{2}$  inputs hold a logic 1.

Fig. 1 shows an example of a 3-input majority gate that is on threshold with respect to input line A. Notice that either order of the inputs places a gate on threshold, i.e.,  $M(A,1,0) = M(A,0,1) = A$ .

While the majority gate in Fig. 1 remains fault-free and on threshold, toggling input line A will result in a toggle at the output. If input A is stuck-at-1(0), then the output will

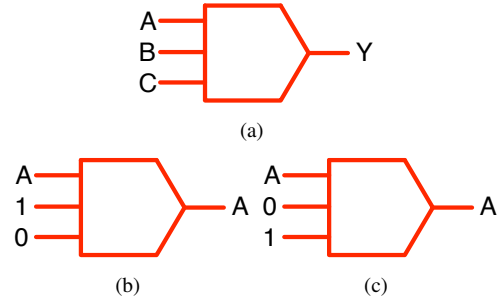


Fig. 1. An example of a majority gate being placed on threshold with respect to input A. Either order of the inputs places a gate on threshold.

also be stuck-at-1(0), and the stuck-at fault will be detected. Detection of bridging faults works the same way. Therefore, if every line in a majority network can be sensitized and toggled, this would be sufficient to test for all SSFs and SBFs in the network.

**Definition 4** A *primitive D cube of failure* (PDCF) is a set of inputs that sensitizes a path for a particular fault within a circuit.

Since we are considering networks of majority gates and inverters, it is useful to consider the unique testing features provided by these networks that cannot be achieved practically in standard CMOS circuits. Consider a 3-input majority gate with input lines A, B, and C and an output Y such that  $Y = AB + AC + BC$ . The test vectors listed in Table I provide the PDCFs to test a single 3-input majority gate for all possible SSFs and SBFs on its inputs. For example, the test vector {001} tests for input lines A and B stuck-at-1 (SA1), input line C dominating either input line A or B with C=1 ( $C(1) \rightarrow A$ ,  $C(1) \rightarrow B$ ), and either input line A or B dominating the other with its complement value with A,B = 0 ( $\bar{A}(0) \rightarrow B$ ,  $\bar{B}(0) \rightarrow A$ ). It is assumed that the logical value of the dominating input line can play a role in its ability to cause a fault on a neighboring interconnect and thus, separate input patterns are used to test for each of the two possible bridging faults, i.e., {001} will test for  $C(1) \rightarrow A$ , while {110} will test for  $C(0) \rightarrow A$ .

The number of PDCFs increases with the size of the majority/minority gate as necessary to test for the additional potential stuck-at and bridging faults that can occur.

TABLE I  
PDCFs FOR A 3-INPUT MAJORITY GATE

| ABC | SSF Detected     | SBF Detected                                                                                          |
|-----|------------------|-------------------------------------------------------------------------------------------------------|
| 001 | A (SA1), B (SA1) | $C(1) \rightarrow A$ , $C(1) \rightarrow B$ , $\bar{A}(0) \rightarrow B$ , $\bar{B}(0) \rightarrow A$ |
| 010 | A (SA1), C (SA1) | $B(1) \rightarrow A$ , $B(1) \rightarrow C$ , $\bar{A}(0) \rightarrow C$ , $\bar{C}(0) \rightarrow A$ |
| 011 | B (SA0), C (SA0) | $A(0) \rightarrow B$ , $A(0) \rightarrow C$ , $\bar{B}(1) \rightarrow C$ , $\bar{C}(1) \rightarrow B$ |
| 100 | B (SA1), C (SA1) | $A(1) \rightarrow B$ , $A(1) \rightarrow C$ , $\bar{B}(0) \rightarrow C$ , $\bar{C}(0) \rightarrow B$ |
| 101 | A (SA0), C (SA0) | $B(0) \rightarrow A$ , $B(0) \rightarrow C$ , $\bar{A}(1) \rightarrow C$ , $\bar{C}(1) \rightarrow A$ |
| 110 | A (SA0), B (SA0) | $C(0) \rightarrow A$ , $C(0) \rightarrow B$ , $\bar{A}(1) \rightarrow B$ , $\bar{B}(1) \rightarrow A$ |

By quick inspection, it is easy to see that the bottom half of the test set shown in Table I is simply the complement of the top half. This is a result of the following property of

majority circuits:

**Property 1.** Let a 3-input majority gate,  $G$ , have inputs  $a, b$ , and  $c$  (for input lines A, B and C, respectively) and an output,  $y$ . If a stuck-at- $v$  fault on any line (or a bridging fault between any two input lines, say  $A(a) \rightarrow B$ ) is detected by input pattern  $abc$ , then the stuck-at- $v$  fault on that line (or the  $A(a') \rightarrow B$  fault) is detectable by the input pattern  $a'b'c'$  (where  $x'$  is the complement of input  $x$ ) [22].

Thus, according to Property 1, for a line  $s$ , it is only necessary to generate a test pattern to detect a stuck-at- $v$  fault since the test pattern to detect the stuck-at- $v'$  fault can be generated by simply complementing all the bits from the original test pattern. The same can be said when generating test vectors for bridging faults.

### C. Testability Analysis

During the test pattern generation process, the PODEM algorithm will need to make decisions at the PIs. In these instances, it is helpful to use testability measures as a guide to make more intelligent decisions, and thus avoid potential conflicts and backtracking. In this work, a probability-based testability analysis heuristic is considered. For a random input pattern, the following three measures for each signal  $s$  in a combinational circuit can be calculated [30]:

- 1)  $C0(s)$  - probability-based 0-controllability of  $s$
- 2)  $C1(s)$  - probability-based 1-controllability of  $s$
- 3)  $O(s)$  - probability-based observability of  $s$

In this work, only  $C0(s)$  and  $C1(s)$  - the probability of controlling signal,  $s$ , to a 0 or 1 from a PI, respectively - are considered. The  $C0$  and  $C1$  probabilities for every node in the circuit are initialized to 0.5 and are adjusted at the completion of each iteration of the PODEM algorithm. For instance, consider the majority gate shown in Fig. 2(a). All nodes are initially set to 0.5/0.5 (0-controllability/1-controllability). Once input line  $A$  is set to a logic 1, the controllability measures are updated to reflect that change as shown in Fig. 2(b). Note, that for each signal,  $s$ , in the circuit,  $C0(s) + C1(s) = 1$ .

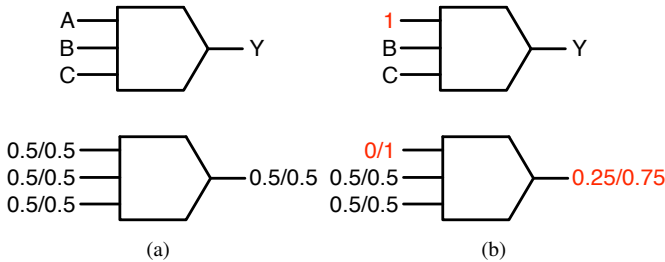


Fig. 2. Calculating probability-based controllability measures. The 2-value tuple  $v_1/v_2$  of each line represents the signal's 0-controllability ( $v_1$ ) and 1-controllability ( $v_2$ ).

### D. High Level Overview

At the onset of the ATPG, all nodes in the circuit are set to  $X$  (don't care). In [15], the author defines what is known as the  $D$ -frontier. The  $D$ -frontier consists of all the gates in the circuit whose output value is  $X$  and has a fault-effect ( $D$  or  $\bar{D}$ ) at one or more of its inputs. The PODEM algorithm places a  $D$  (or  $\bar{D}$ ) at the site of the target fault  $f$  and then tries to propagate the  $D$ -frontier to a PO by making decisions only at the PIs. If at any point the  $D$ -frontier becomes an empty set prior to reaching the output, the algorithm must backtrack and try a different path. The basic flow of the modified PODEM algorithm is shown in Functions 1 and 2 [30] and illustrated in Fig. 3. The most significant change made in the modified algorithm is that in order to cover all SBFs, the algorithm applies the set of PDCFs (for instance, the PDCFs for 3-input majority gates shown in Table I) to each majority gate in the circuit as opposed to simply assigning values to individual nodes. As such, for a given majority gate under test, the search begins by assigning each node of this majority gate a binary value as determined by one of the PDCFs, and backtracing from this node  $a$  to a PI via the best path. The discussed controllability measures are used here to determine which path is regarded as the best. This continues until each of the input nodes of the majority gate are assigned. Once this is complete, the algorithm is to check to see if the fault effect has been observed at a PO. If not, then the search continues by systematically selecting new objectives and backtracing from those objectives to an unassigned PI. As the algorithm progresses, more PIs get assigned logic values. If at any point the  $D$ -frontier becomes empty an empty set, the backtracking mechanism proceeds by reversing the most recent decision. If reversing the most recent decision also causes a conflict, then the next most recent decision is reversed and so on until either there is no longer a conflict or no more reversals are possible. In the latter case, the fault is considered to be undetectable and the circuit is determined to be redundant.

#### Function 1 PODEM-MAJ( $C, f$ ) [30]

```

1: initialize all gates to don't cares;
2:  $D$ -frontier =  $\emptyset$ ;
3:  $M$  = majority gate under test;
4: while  $M$  has unassigned inputs do
5:    $g$  = an unassigned input of  $M$ ;
6:    $v$  = binary input from selected test pattern (Table I);
7:    $(pi, u) = \text{backtrace}(g, v)$ ;
8:    $\text{logicSimulate\_and\_imply}(pi, u)$ ;
9: end while
10: result = PODEM-MAJ-Recursion( $C$ );
11: if result == success then
12:   print out values at the primary inputs;
13: else
14:   print fault  $f$  is untestable
15: end if

```

The most important functions in PODEM-MAJ-Recursion() are the `getObjective()`, `backtrace()` and `logicSimulate_and_imply()` functions. The `getObjective()` function returns the next objective that the ATPG should try and justify. As discussed earlier, the first objectives are always to set the input lines on a given majority gate to one of the PDCFs. Once the fault is excited, `getObjective()` chooses the

---

**Function 2** PODEM-MAJ-Recursion( $C$ ) [30]
 

---

```

1: if fault-effect is observed at a PO then
2:   return (success);
3: end if
4: ( $g, v$ ) = getObjectiv( $C$ );
5: ( $pi, u$ ) = backtrack( $g, v$ );
6: logicSimulate_and_imply( $pi, u$ );
7: result = PODEM-MAJ-Recursion( $C$ );
8: if result == success then
9:   return (success);
10: end if
11: /* backtrack */
12: logicSimulate_and_imply( $pi, \bar{u}$ );
13: result = PODEM-MAJ-Recursion( $C$ );
14: if result == success then
15:   return (success);
16: end if
17: logicSimulate_and_imply( $pi, x$ );
18: return (failure);

```

---

best fault effect in the  $D$ -frontier to propagate to a PO by placing gates along that path on threshold. The algorithm for getObjectiv() is shown in Function 3 [30]. For a majority gate,  $d$ , the non-controlling input of  $d$  is dependent on the other input. For instance, suppose  $g_1$  and  $g_2$  are needed to place a 3-input majority gate  $d$  on threshold. If either  $g_1$  or  $g_2$  are specified (*i.e.*, have been assigned a logic 0 or 1), then the other must be set such that it is logically opposite to it.

---

**Function 3** getObjectiv( $C$ ) [30]
 

---

```

1: if fault is not excited then
2:   return ( $g, \bar{v}$ );
3: end if
4:  $d$  = gate in  $D$ -frontier;
5:  $g$  = an input of  $d$  whose input is  $x$ ;
6:  $v$  = a non-controlling value of  $d$ ;
7: return ( $g, v$ );

```

---

The backtrack() function takes in as its inputs, the gate and logic value provided by the getObjectiv() function. It then backtraces through a path of unjustified gates until it reaches a PI and sets the value of that PI such that it will allow the circuit to meet the desired objective. This algorithm is shown in Function 4 [30].

---

**Function 4** backtrack( $g, v$ ) [30]
 

---

```

1:  $i = g$ ;
2: num_inversions = 0;
3: while  $i \neq$  primary input do
4:    $M$  = a gate for which  $i$  is the output;
5:    $i$  = an input of  $M$  whose value is  $x$ ;
6:   if  $M$  is an inverted gate type then
7:     num_inversions++;
8:   end if
9: end while
10: if num_inversions == odd then
11:    $v = \bar{v}$ ;
12: end if
13: return ( $i, v$ );

```

---

Lastly, the logicSimulate\_and\_imply() is a simple logic simulation routine which also aims to derive any additional implications that may arise in order to enhance the getObjectiv() function.

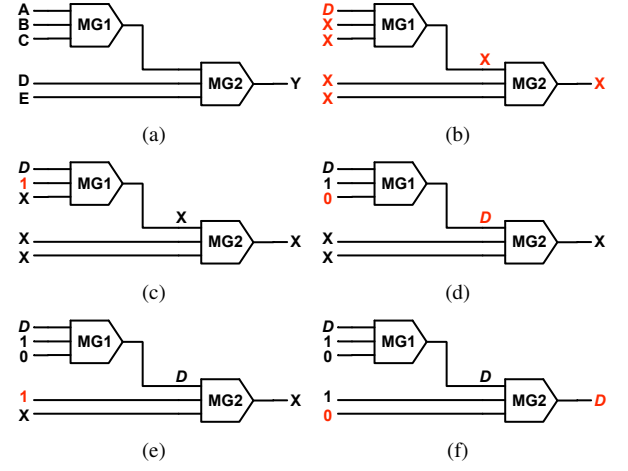


Fig. 3. Example of the PODEM algorithm developing a test pattern to test for node A SA0. (a) Circuit under test (CUT). (b) All nodes are initialized to don't cares, and node A is set to  $D$  (1/0). The  $D$ -Frontier consists of majority gate MG1. (c) In an attempt to propagate the  $D$ -Frontier to the PO, node B is set to a logic 1. (d)  $D$ -Frontier is advanced by setting node C to a logic 0. The  $D$ -Frontier now consists of majority gate MG2. MG1 is no longer in the  $D$ -Frontier. (e) In an attempt to propagate the  $D$ -Frontier to the PO, node D is set to a logic 1. (f)  $D$ -Frontier is propagated to the PO by setting node E to a logic 0.

### III. SIMULATION RESULTS

In this section, we present our simulation results. Our algorithm was implemented by extending the PODEM algorithm such that it considers majority logic as a gate primitive. The algorithm amounted to roughly 700 lines of MATLAB code. In addition, we developed a genetic algorithm similar to the one described in [32] to statically fill-in the unspecified bits in our resulting test set. This compaction algorithm resulted in approximately 500 additional lines of C++ code and is available upon request to the authors. We ran our PODEM algorithm on 38 MCNC benchmark circuits [33] which have been synthesized into multi-level 3-input majority networks using the MAJORITY Logic Synthesizer (MALS) tool [34]. Table II shows the number of test vectors generated for each of the considered benchmark circuits. The second and third columns display the total number of PIs and majority gates in the circuit, while the final column shows the percentage of the entire set of input patterns used to detect all SSFs and SBFs. The results shown in Table II are the test set sizes post-compaction.

While there exists no other work against which we can compare our results, based on the results reported in [19] (where the PODEM algorithm was used to generate test sets to detect SSFs on 3-input majority networks, and compared to those using a SAT-based framework in [12], [13]), we believe that for most of the benchmark circuits considered in this work, the test sets produced using our PODEM algorithm are as small, if not smaller, than what any other test generation method can produce. An exhaustive test simulation would require  $2^{\text{PI}}$  test vectors, and thus for most of the considered benchmark circuits, our method has shown to provide significant reduction in the overall test set size. In fact, more than two-thirds of the test sets produced were less than 5% of the

TABLE II  
SIMULATION RESULTS

| Benchmark Circuit | Layout  |            | No. of Test Vectors | % of Input Patterns Used |
|-------------------|---------|------------|---------------------|--------------------------|
|                   | # of PI | # of Gates |                     |                          |
| 9symm1            | 9       | 214        | 145                 | 28.3%                    |
| alu2              | 10      | 256        | 182                 | 17.7%                    |
| b1                | 5       | 10         | 5                   | 15.63%                   |
| c8                | 23      | 49         | 80                  | < 0.001%                 |
| cc                | 30      | 116        | 98                  | < 0.001%                 |
| cht               | 47      | 120        | 81                  | < 0.001%                 |
| cm138a            | 6       | 20         | 13                  | 20.3%                    |
| cm150a            | 21      | 46         | 42                  | < 0.001%                 |
| cm151a            | 12      | 42         | 25                  | 0.61%                    |
| cm152a            | 11      | 21         | 20                  | 0.98%                    |
| cm162a            | 14      | 46         | 37                  | 0.23%                    |
| cm163a            | 16      | 42         | 44                  | 0.07%                    |
| cm42a             | 4       | 21         | 12                  | 75%                      |
| cm82a             | 5       | 16         | 12                  | 37.5%                    |
| cm85a             | 11      | 34         | 35                  | 1.71%                    |
| cmb               | 16      | 44         | 36                  | 0.05%                    |
| count             | 35      | 144        | 83                  | < 0.001%                 |
| cu                | 14      | 46         | 38                  | 0.23%                    |
| decod             | 5       | 28         | 17                  | 53.13%                   |
| example2          | 85      | 259        | 199                 | < 0.001%                 |
| frg1              | 28      | 111        | 116                 | < 0.001%                 |
| i1                | 27      | 45         | 52                  | < 0.001%                 |
| i2                | 201     | 209        | 402                 | < 0.001%                 |
| lal               | 28      | 96         | 88                  | < 0.001%                 |
| ldd               | 9       | 75         | 24                  | 4.69%                    |
| majority          | 5       | 6          | 11                  | 34.38%                   |
| mux               | 21      | 46         | 41                  | 0.002%                   |
| pcle              | 19      | 67         | 68                  | 0.013%                   |
| pcle8             | 27      | 90         | 97                  | < 0.001%                 |
| pm1               | 16      | 45         | 43                  | 0.066%                   |
| sct               | 19      | 74         | 71                  | 0.014%                   |
| tcon              | 17      | 32         | 28                  | 0.021%                   |
| term1             | 36      | 174        | 167                 | < 0.001%                 |
| ttt2              | 24      | 154        | 139                 | < 0.001%                 |
| unreg             | 36      | 84         | 51                  | < 0.001%                 |
| x2                | 10      | 42         | 25                  | 2.44%                    |
| z4ml              | 7       | 27         | 19                  | 14.84%                   |

total number of possible input patterns. Furthermore, since our algorithm produces 6 test vectors per majority gate in the circuit (see: Table I), the compaction algorithm used in this work has succeeded in significantly reducing the original test sets produced by our ATPG.

While the proposed algorithm was only used to test majority networks made up of entirely 3-input majority gates, it can be easily extended to test circuits with larger majority gates by simply producing a new set of PDCFs to apply to each gate.

#### IV. CONCLUSIONS

In this paper, we presented a novel extension to the well-known PODEM algorithm for applications in testing the emerging nanotechnologies including QCA, RTD, SET, and TPL which implement majority/minority logic. In addition to testing all possible SSFs in a circuit, we have also modified the PODEM algorithm to test for all bridging faults between interconnects within a majority gate. We tested our PODEM algorithm on a set of MCNC benchmark circuits implemented using 3-input majority gates. For the majority of the benchmark circuits considered in this work, the test sets produced using our PODEM algorithm are as small, if not smaller, than what any other test generation method can produce. The resulting test sets from the proposed algorithm were also highly compact. In over two-thirds of the benchmark circuits considered in this work, the size of the test set was less than 5% of the total number of possible input patterns.

Since it is likely that interconnects will consume the majority of chip area in these emerging nanotechnologies, it would be interesting to try and expand this work to include the potential for detecting multiple bridging faults.

#### ACKNOWLEDGMENT

The authors acknowledge financial support from the Natural Science and Engineering Research Council (NSERC), Canadian Foundation for Innovation (CFI), as well as the British Columbia Knowledge Development Fund (BCKDF).

#### REFERENCES

- [1] M. B. Haider, J. L. Pitters, G. A. DiLabio, L. Livadaru, J. Y. Mutus, and R. A. Wolkow, "Controlled coupling and occupation of silicon atomic quantum dots at room temperature," *Phys. Rev. Lett.*, vol. 102, no. 4, p. 4, 2009.
- [2] L. Wang, V. Krapek, F. D. abd Felicity Horton, A. Schliwa, D. Bimberg, A. Rastelli, and O. G. Schmidt, "Self-assembled quantum dots with tunable thickness of the wetting layer: Role of vertical confinement on interlevel spacing," *Phys. Rev. B*, vol. 80, no. 8, p. 9, 2009.
- [3] U. F. Keyser, H. W. Schumacher, U. Zeitler, R. J. Haug, and K. Eberl, "Fabrication of quantum dots with scanning probe nanolithography," *Phys. Stat. Sol. B*, vol. 224, no. 3, pp. 681–684, 2001.
- [4] K. Hennessy and C. S. Lent, "Clocking of molecular quantum-dot cellular automata," *J. of Vacuum Science and Technology B: Microelectronics and Nanometer Structures*, vol. 19, pp. 1752–1755, 2001.
- [5] C. S. Lent, B. Isaksen, and M. Lieberman, "Molecular quantum-dot cellular automata," *J. Am. Chem. Soc.*, vol. 125, pp. 1056–1063, 2003.
- [6] Marya Lieberman et al., "Quantum-dot cellular automata at a molecular scale," *Ann. N.Y. Acad. Sci.*, vol. 960, pp. 225–239, 2002.
- [7] G. Toth, *Correlation and Coherence in Quantum-Dot Cellular Automata*, 2000, Ph.D Thesis.
- [8] H. Fahmy and R. Kiehl, "Complete logic family using tunneling-phase-logic devices," in *Proc. Int. Conf. Microelectronics*, Nov. 1999, pp. 22–24.
- [9] T. Oya, T. Asai, T. Fukui, , and Y. Amemiya, "A majority-logic device using an irreversible single electron box," *IEEE Trans. on Nanotechnol.*, vol. 2, no. 1, pp. 15–22, Mar. 2003.
- [10] —, "A majority-logic nanodevice using a balanced pair of single electron boxes," *J. Nanosci. Nanotech.*, vol. 2, no. 3/4, pp. 333–342, Aug. 2002.
- [11] N. Jha and S. Gupta, *Testing of Digital Systems*. Cambridge, United Kingdom: Cambridge University Press, 2003.
- [12] P. Gupta, N. K. Jha, and L. Lingappan, "Test generation for combination quantum cellular automata (QCA) circuits," in *Proc. Design, Automation and Test in Europe*, 2006, pp. 311–316.
- [13] —, "A test generation framework for quantum cellular automata circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) systems*, vol. 15, no. 1, pp. 24–36, 2007.
- [14] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Comput.*, pp. 215–222, 1981.
- [15] J. Roth, "Diagnosis of automata failures: a calculus and a method," *IBM J. R&D*, vol. 10, no. 4, pp. 278–291, 1966.
- [16] M. Schulz, E. Trischler, and T. Sarfert, "SOCRATES: a highly efficient automatic test pattern generation system," *IEEE Trans. Comput.-Aided Des.*, vol. 8, no. 7, pp. 811–816, 1988.
- [17] S. Blanton and J. Hayes, "On the properties of the input pattern fault model," *ACM Transactions on Design Automation of Electronic Systems*, vol. 8, no. 1, pp. 108–124, 2003.
- [18] N. A. Touba and E. J. McCluskey, "Pseudo-random pattern testing of bridging faults," in *Proceedings of the 1997 International Conference on Computer Design (ICCD '97)*, 1997.
- [19] F. Karim, K. Walus, and A. Ivanov, "Testing of combinational majority and minority logic networks," in *IEEE International Mixed-Signals, Sensors and Systems Test Workshop 2008 (IMS3TW'08)*, 2008.
- [20] L. Fang and M. Hsiao, "Bilateral testing of nano-scale fault-tolerant circuits," in *Proceedings of the 21st International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'06)*, 2006.
- [21] —, "Bilateral testing of nano-scale fault-tolerant circuits," *J. of Electron. Test.*, vol. 24, pp. 285–296, 2008.

- [22] M. B. Tahoori, J. Huang, M. Momenzadeh, , and F. Lombardi, "Testing of quantum cellular automata," *IEEE Transactions on Nanotechnology*, vol. 3, pp. 432–442, 2004.
- [23] M. B. Tahoori, M. Momenzadeh, J. Huang, and F. Lombardi, "Defects and faults in quantum cellular automata at nano scale," in *Proc. of the 22nd IEEE VTS*, 2004, pp. 291–296.
- [24] M. Momenzadeh, M. Ottavi, , and F. Lombardi, "Modeling qca defects at molecular-level in combinational circuits," in *Proc. of the 20th IEEE DFT*, 2005, pp. 208–216.
- [25] X. Ma, J. Huang, C. Metra, and F. Lombardi, "Testing reversible one-dimensional qca arrays for multiple faults," in *Proceedings of the 22nd International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'07)*, 2007.
- [26] —, "Reversible gates and testability of one dimensional arrays of molecular qca," *J. of Electron. Test.*, vol. 24, pp. 297–311, 2008.
- [27] —, "Detecting multiple faults in one-dimensional arrays of reversible qca gates," *J. of Electron. Test.*, vol. 25.
- [28] S. Sultana, S. A. Imam, and K. Radecka, "Testing QCA modular logic," in *13th International Conference on Electronics, Circuits and Systems, 2006. ICECS '06.*, 2006, pp. 700–703.
- [29] B. Sikdar, "Study of N-Detectability in QCA designs," in *15th Asian Test Symposium, ATS'06.*, 2006.
- [30] L.T. Wang and C.W. Wu and X. Wen, *VLSI Test Principles and Architectures*. San Francisco, CA: Morgan Kaufmann Publishers, 2006.
- [31] J. P. Hayes, "A nand model for fault diagnosis in combinational logic networks," *IEEE Trans. on Computers*, vol. c-20, no. 12, pp. 1496–1506, 1971.
- [32] E. M. Rudnick and J. H. Patel, "Efficient techniques for dynamic test sequence compaction," *IEEE Trans. Comput.-Aided Des.*, vol. 48, no. 3, pp. 323–330, 1999.
- [33] R. Lisanke, "Logic synthesis and optimization benchmarks," in *Microelectronics Center of North Carolina, Tech. Rep.*, 1988.
- [34] R. Zhang, P. Gupta, and N. K. Jha, "Synthesis of majority and minority networks and its applications to QCA, TPL, and SET based nanotechnologies," in *Proc. In. Conf. VLSI Design*, Jan. 2005, pp. 229–234.