Testing of Combinational Majority and Minority Logic Networks

Faizal Karim, *Student Member, IEEE*, Konrad Walus, *Member, IEEE*, André Ivanov, *Fellow, IEEE*

***Abstract*— In this paper, we present an extension to the existing PODEM algorithm to include the ability to generate test patterns for majority and minority networks, specifically targeting quantum-dot cellular automata (QCA), but that is directly applicable to other emergent nanotechnologies such as single electron tunneling (SET) and tunneling phase logic (TPL). A dynamic probability-based controllability technique was developed and used as a guide to make more intelligent decisions on which lines to justify during the automatic test pattern generation (ATPG) process. Lastly, a genetic algorithm was 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 MCNC benchmark circuits when using both fixed polarized cells and external inputs to implement the AND and OR gates. Test set sizes were much smaller when implementing the AND/OR gates using fixed polarized cells, however, the computational times for the latter method were generally shorter.**

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

I. Introduction

THE availability of nano fabrication processes 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 cellular automata (QCA), single electron tunneling (SET), and tunnel phase logic (TPL) are capable of implementing majority and minority logic [1]–[7]. 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 will be difficult to realize without defects. Amongst the numerous potential faults likely to occur in such circuits, single stuck-at faults (SSF) can be used to represent a large portion (80-85%) of the general faults in these systems [8]. In the SSF model, a line in a circuit is assumed to be permanently stuck-at-1 (SA1) or stuck-at-0 (SA0). Several algorithms to automatically generate test patterns for SSFs in CMOS circuits have already been proposed [9]–[11], however, it remains to be seen whether or not the conventional CMOS test flow can accommodate the testing of nanotechnologies. To this effect, we present an extension to the PODEM algorithm such that majority and minority gates are included within its scope. Specifically, since the generation of test vectors does not differ for majority or minority logic, this work will simply focus on the testing of QCA circuits which use the 3-input majority gate as its basic logic element.

The authors acknowledge financial support from NSERC.

Since its initial proposal, testing and design for test of QCA circuits has generated considerable interest. Most notably, Gupta *et al*. have developed a comprehensive test generation methodology using Boolean satisfiability (SAT) to cover both SSFs and bridging faults in QCA circuits and have tested their automatic test pattern generator (ATPG) on a set of MCNC benchmark circuits [12]. Tahoori *et al*. have investigated test set generation for fanout-free AND/OR 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 [13]–[15]. The authors, however, do not discuss the implementation of an ATPG to detect any of the functional faults described in their work. More recently, a new strategy for exhaustive SSF testing of combinational QCA logic is presented in [16]. Here, 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 benchmark circuits considered in this work. Lastly, a pseudo-random pattern generator (PRPG) has been used to study the effects of N-detectability of QCA circuits in [17]. 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].

The rest of this paper is organized as follows. In Section II we present background material on the fundamental concepts of QCA. The implementation details of our PODEM algorithm is discussed in Section III with experimental results provided in Section IV. Section V concludes the paper.

II. Preliminaries

QCA is a computing paradigm which utilizes the electrostatic coupling between electronic configurations in neighboring cells to perform information processing. A QCA cell, shown in Fig. 1, contains four quantum dots and two mobile electrons which can tunnel from one quantum dot to another [18]. The different polarizations of the QCA cell are also shown in Fig. 1. Here, the bounding box around the cell is used only to distinguish one cell from another and has no physical analogue. The NULL state is a superposition of the two ACTIVE states and does not represent a binary value.

A. QCA Building Blocks

The fundamental logic primitive that can be realized in QCA is the majority gate as shown in Fig. 2. The truth table for

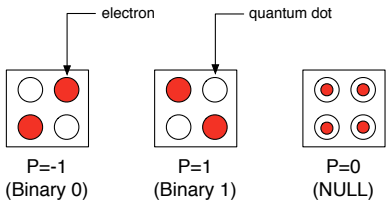


Fig. 1. Three QCA cells with different polarizations are shown. Each cell carries two extra electrons which tend to occupy the diagonals of the cell when in one of the two ACTIVE states.

the majority gate is shown in Table I. The output cell of the majority gate assumes the value of the majority of the input cells.

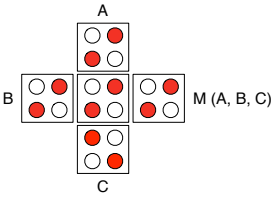
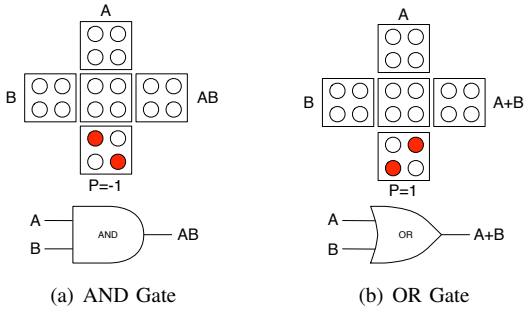


Fig. 2. QCA majority gate. The output of the majority gate reflects the majority of the inputs.

A	B	C	M(A, B, C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

TABLE I
TRUTH TABLE FOR MAJORITY GATE.

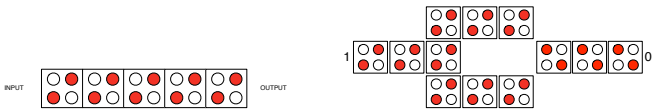
In QCA, a 2-input AND or OR gate can be created by simply fixing the polarization of one of the three majority gate inputs to a logic "0" or a logic "1", respectively, as shown in Fig. 3.



(a) AND Gate (b) OR Gate

Fig. 3. Fixing one of the inputs of a majority gate to P=-1 creates an AND gate. Similarly, fixing one of the inputs to P=1 creates an OR gate.

Lastly, an interconnect and an inverter can also be implemented in QCA as shown in Fig. 4.



(a) Binary Wire (b) Inverter

Fig. 4. A QCA (a) binary wire and (b) inverter.

Collectively, the majority gate along with the inverter provide a universally complete computing logic set (i.e., a set of Boolean logic gates that can perform the AND, OR, and NOT operations).

B. 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) [19]. Since both the fault-free and faulty logic values are needed, composite logic values are used. 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 [10], values $0, 1, X, D, \bar{D}$ are considered where $0, 1$ and X are conventional values used to represent true, false 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 III. In general, an ATPG 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 will have to make decisions. If any of these decisions causes a conflict on a given signal line, the ATPG must reverse a previous decision. The reversal of a decision is called a backtrack.

Since all QCA circuits are implemented using a network of majority gates and inverters, it is useful to consider the unique testing features provided by these networks that cannot be achieved in standard CMOS circuits.

Consider a majority gate with input lines A, B, and C and an output Y such that $Y = AB + AC + BC$.

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 is detected by input pattern abc, then the stuck-at-v' fault on that line is detectable by the input pattern $a'b'c'$ (where x' is the complement of input x) [13].

This property is particularly useful because for any circuit that is composed exclusively of majority gates and inverters, Property 1 suggests that a maximum of 2^{n-1} test vectors are required, where n represents the number of primary inputs. Indeed we can assume all our circuits are entirely composed of majority gates and inverters. As mentioned in Section II, if a circuit contains AND/OR gates, these are themselves generated with majority gates for which one of the inputs is fixed. It is worth noting that these fixed cells can be implemented by either fixing the polarization of a QCA cell such it never changes its configuration, or as suggested by

Tahoori *et al*. in [13], by using control lines which can be seen as extra input lines during testing time. Considering only the size of the final test set, it may be more desirable to implement the fixed cells using the former method as it would not be necessary to test the fixed cells for stuck-at-v faults, where v is the logic value associated with the fixed cell. However, computationally, the latter method is more desirable since, 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. This will be investigated in more detail in Section IV.

III. PODEM for QCA Circuits

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

A. 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 [20].

To test any input line in a non-redundant network, we need to sensitize the input by placing its 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 sensitive if toggling the input toggles the output.

Definition 3 A gate G is considered to be on threshold with respect to one of its inputs i, if the other two gate inputs are made logically opposite to one another.

Fig. 5 shows an example of a 3-input majority gate that has been placed on threshold with respect to input line A. Notice that the order of the inputs does not matter when placing a gate on threshold, *i.e.*, M(A,1,0) = M(A,0,1) = A.

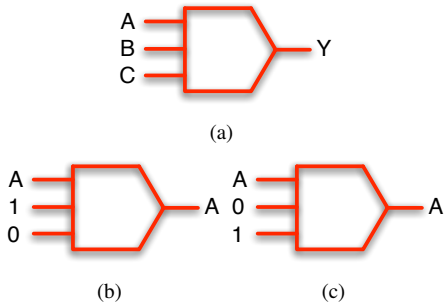


Fig. 5. An example of a majority gate being placed on threshold with respect to input A. The order of the inputs does not matter when placing a gate on threshold.

While the majority gate in Fig. 5 remains fault-free and in 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 also be stuck-at-1(0), and the stuck-at fault will be detected. Therefore, if every line in a majority network can be sensitized and toggled, this would be sufficient enough to test for all SSFs in the network.

B. Fault Collapsing

The reduction of an entire test set by removing equivalent faults is referred to as fault collapsing. Fault collapsing helps to reduce test generation time and provides more compact test set sizes. For majority circuits, we can make use of the fault equivalencies that exist between the outputs and the inputs of majority gates. For instance, in order to advance the fault-effect (D or $\bar{D}$) from the location of the target fault to a PO, it is necessary to sensitize every gate in the given path.

Definition 4 A path P in a network W is an ordered set of lines $(x_1, x_2, \ldots, x_p)$ such that if x_i is an input to gate G_i, then x_i+1 is the output of gate G_i for $i = 1, 2, \ldots, p-1$ [20].

To sensitize a path, every gate input along the path needs to be placed on threshold as shown in Fig. 6.

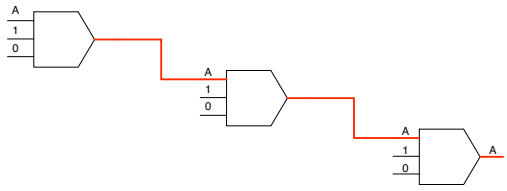


Fig. 6. An example of a sensitized path.

It is clear from Fig. 6 that when a path is sensitized, every node along that path is also being sensitized. Thus, for a combinational majority circuit, C, any test set, V, that detects all SSFs on the primary inputs and fanout branches detects all SSFs in C [12]. This particular property is useful for fault collapsing of majority circuits.

C. Testability Analysis

During the test pattern generation process, it is required that the ATPG make decisions. 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 technique is considered. For a random input pattern, the following three measures for each signal s in a combinational circuit can be calculated:

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. All probabilities range from 0 to 1. 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. 7(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. 7(b). Note, that for each signal, s, in the circuit, C0(s) + C1(s) = 1.

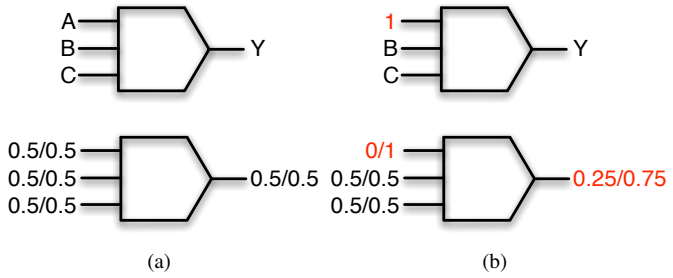


Fig. 7. Calculating probability-based controllability measures. The 2-value tuple v_1/v_2 of each line represents the signal's 0-controllability (v_1) and 1-controllability (v_2).

D. High Level Overview

In [10], 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 (don't care) 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 empty, the algorithm must backtrack and try a different path. The basic flow of the modified PODEM algorithm is illustrated in Algorithms 1 and 2 [19]. According to the algorithm, the search begins by picking an objective and backtracing from the objective to a PI via the best path. The discussed controllability measures are used here to determine which path is regarded as the best. As the algorithm progresses, more PIs get assigned logic values. If at any point the D-frontier becomes empty, 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 determined to be undetectable.

Algorithm 1 PODEM-QCA(C, f) [19]
1: initialize all gates to don't cares;
2: D-frontier = $\emptyset$;
3: result = PODEM-QCA-Recursion(C);
4: **if** result == success **then**
5: print out values at the primary inputs;
6: **else**
7: print fault f is untestable
8: **end if**

The most important functions in PODEM-QCA-Recursion() are the getObjective(), backtrace() and logicSimulate_and_imply() functions. The getObjective() function returns the next objective that the ATPG should try and justify. The first objective is always to set the line on which the target fault resides to the opposite value of the stuck value. Once the fault is excited, getObjective() chooses the best fault effect in the D-frontier to propagate to a PO by placing gates along that path on threshold. The algorithm for getObjective() is shown in Algorithm 3 [19]. 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 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.

Algorithm 2 PODEM-QCA-Recursion(C) [19]
1: **if** fault-effect is observed at a PO **then**
2: return (success);
3: **end if**
4: (g, v) = getObjective(C);
5: (pi, u) = backtrace(g, v);
6: logicSimulate_and_imply(pi, u);
7: result = PODEM-QCA-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-QCA-Recursion(C);
14: **if** result == success **then**
15: return (success);
16: **end if**
17: logicSimulate_and_imply(pi, x);
18: return (failure);

Algorithm 3 getObjective(C) [19]
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 backtrace() function takes in as its inputs, the gate and logic value provided by the getObjective() 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 Algorithm 4 [19].

Algorithm 4 backtrace(g, v) [19]
1: i = g;
2: num_inversions = 0;
3: **while** $i \neq$ primary input **do**
4: i = an input of i whose value is x;
5: **if** i is an inverted gate type **then**
6: num_inversions++;
7: **end if**
8: **end while**
9: **if** num_inversions == odd **then**
10: $v = \bar{v}$;
11: **end if**
12: 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 getObjective() function.

IV. Experimental Results

In this section, we present our experimental results. Our algorithm was implemented by extending the PODEM algorithm such that it considers majority logic within its scope. The algorithm amounted to roughly 700 lines of MATLAB code. In addition, we developed a genetic algorithm similar to the one described in [21] to statically fill-in the unspecified bits in our resulting test set. This compaction algorithm resulted in approximately an additional 500 lines of C++ code. We ran our PODEM algorithm on 28 MCNC benchmark circuits [22] which have been synthesized into multi-level majority networks using the logic synthesis tool MALS [23]. Table II shows the number of test vectors generated for each of the considered benchmark circuits when using both fixed polarized cells and external inputs to implement the AND and OR gates. The third column shows the results of the SSF test set size generated using SAT in [12] which also assumed fixed polarized cells to implement AND/OR gates. The bolded entries in the table are those which either equal or better the results reported in [12]. While the test set size is considerably smaller for circuits implementing the AND/OR gates using fixed polarized cells, the computational time for the latter method was generally shorter in spite of the larger fault list.

TABLE II
Simulation Results

Benchmark	External Inputs	Fixed Cells	SAT [12]
alu2	484	222	165
cht	104	98	24
cm138a	**33**	**12**	**–**
cm150a	74	**28**	**35**
cm151a	59	**18**	**18**
cm152a	31	**10**	**18**
cm162a	88	**22**	**22**
cm163a	69	**22**	**24**
cm42a	**28**	**14**	**–**
cm82a	24	**11**	**12**
cm85a	66	**26**	**26**
cmb	63	**22**	**39**
count	78	**30**	**46**
cu	86	**20**	**31**
decod	**41**	**19**	**–**
frg1	167	**84**	**90**
ldd	**95**	**22**	**–**
majority	**12**	**7**	**–**
mux	60	**24**	**38**
pcle	101	49	23
pcler8	234	60	41
pm1	**89**	**37**	**–**
sct	**141**	**55**	**–**
tcon	**52**	**23**	**–**
ttt2	237	112	43
unreg	85	37	24
x2	73	**18**	**23**
z4ml	45	26	15

V. Conclusions

In this paper, we presented an extension to the existing PODEM algorithm to make it applicable to some of the emerging nanotechnologies including QCA, SET, and TPL. We developed a probability-based controllability measure to guide the decision making process and modified an existing genetic algorithm designed for filling in the unspecified values in the resulting test set generated by the ATPG. We tested our PODEM algorithm on a set of MCNC benchmark circuits when using both fixed polarized cells and external inputs to implement the AND and OR gates. While test set sizes were smaller when implementing the AND/OR gates using fixed polarized cells, the computational time for the latter method were generally shorter.

Interconnects are likely to consume the bulk of the chip area in nanotechnologies such as QCA and as such, bridging faults are also likely to occur. It is necessary to extend this work to include those faults. Future work should also consider the possibility of multiple stuck-at and 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] 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.

[2] C. S. Lent, B. Isaksen, and M. Lieberman, "Molecular quantum-dot cellular automata," *J. Am. Chem. Soc.*, vol. 125, pp. 1056–1063, 2003.

[3] M. L. et. al., "Quantum-dot cellular automata at a molecular scale," *Ann. N.Y. Acad. Sci.*, vol. 960, pp. 225–239, 2002.

[4] G. Toth, *Correlation and Coherence in Quantum-Dot Cellular Automata*, 2000, Ph.D Thesis.

[5] H. Fahmy and R. Kiehl, "Complete logic family using tunneling-phase-logic devices," in *Proc. Int. Conf. Microelectronics*, Nov. 1999, pp. 22–24.

[6] 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.

[7] ——, "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.

[8] N. Jha and S. Gupta, *Testing of Digital Systems*. Cambridge, United Kingdom: Cambridge University Press, 2003.

[9] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Comput.*, pp. 215–222, 1981.

[10] J. Roth, "Diagnosis of automata failures: a calculus and a method," *IBM J. R&D*, vol. 10, no. 4, pp. 278–291, 1966.

[11] 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.

[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] 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.

[14] 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.

[15] 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.

[16] 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.

[17] B. Sikdar, "Study of N-Detectability in QCA designs," in *15th Asian Test Symposium, ATS'06.*, 2006.

[18] C. S. Lent, "Quantum cellular automata," *Nanotechnology*, vol. 4, pp. 49–57, 1993.
[19] L.T. Wang and C.W. Wu and X. Wen, *VLSI Test Principles and Architectures*. San Francisco, CA: Morgan Kaufmann Publishers, 2006.
[20] 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.
[21] 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.
[22] R. Lisanke, "Logic synthesis and optimization benchmarks," in *Microelectronics Center of North Carolina, Tech. Rep.*, 1988.
[23] 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.