On Design and Testing of Combinational Circuits based on Quantum Cellular Automata

SURESH RAI
Louisiana state University, Baton Rouge, USA

(Received on November 21, 2010, revised on April 12, 2011)

Abstract: Quantum Cellular Automata (QCA) present one of the many emerging computational nanotechnology paradigms. It uses the arrangements of individual electrons, instead of currents and voltages, to encode binary information. Note a QCA cell consists of four quantum dots located at the corners of a square. This tutorial paper provides an introductory material on the design and testing of combinational logic using a QCA based majority gate. The objective is to create an interest among readers of IJPE in the area of QCA and testing of combinational circuits.

Keywords: CMOS circuit, logic design and testing, majority gate, nanotechnology, quantum cellular automata

1. Introduction

To implement high-density, high-speed, and low-power very large-scale integrated systems, complementary metal-oxide semiconductor (CMOS) technology was, so far, very useful. It provided the required dimension scaling to support the integration. However, the growing technology cost and limitations in CMOS design make it imperative to research in nanoscale technologies for a viable alternative. This paper considers the alternative paradigm of quantum-dot cellular automata (QCA), which use the arrangements of individual electrons, instead of currents and voltages, to encode binary information [1, 2].

QCA uses arrays of quantum-dot cells to implement digital logic performing all complex functions required for general-purpose computations. Reference [2] highlights its advantage as follows: “It is believed that a QCA cell of a few nanometers can be fabricated through molecular implementation by a self-assembly process. If this does hold true, then it is anticipated that QCA can achieve densities of $10^{12}$ devices/cm$^2$ and operate at THz frequencies.”

A QCA cell, shown in Figure 1 (a), consists of four quantum dots located at the corners of a square. When the cell is charged with two excel electrons, they occupy diagonal sites because of mutual electrostatic repulsion. Due to Coulombic interactions, only two configurations of the electron pair exist and they are stable. Assign a logic 0 and logic 1 to represent the stable (ground) states of the cells. The QCA logic device is a three-input majority logic gate, M, which is logic high if two or more of its inputs are logic high. That is

$$M(A, B, C) = AB + AC + BC.$$  

Fig. 1 (b) shows a majority gate created with a cross pattern of five QCA cells. In Fig. 1 (c), we provide a representation of an M gate. The reason why the device cell always
assumes a majority polarization is that it is in this polarization state that the Coulombic interaction between electrons in the input cells is minimized [3].

![Diagram of QCA cell and majority gate](image)

Figure 1: A QCA (a) Cell, (b) Majority Gate M, and (c) Representation of M

In order to design complex combinational circuits with QCA, we require a method for representing Boolean functions as optimally arranged majority gate based circuits [4]. For majority logic synthesis, refer to [5, 6, 7]. Techniques in [5, 6, 7] employ reduced-utilized-table, Karnaugh map (K-map), and Shannon’s decomposition based principles to synthesize the combinational circuit using M gates. All methods fail to perform majority synthesis efficiently. Recently, reference [3] has used a factorization and simplification approach to solve this problem.

Sections 2 and 3 reconsider synthesis problem with majority logic and discuss an efficient approach for this for applications with QCA. Sections 4 and 5 are devoted to testing issues for combinational CMOS and QCA circuits designed around M gates. The types of defects that are likely to occur in the manufacturing of QCA devices have been investigated in the literature [8]. Two examples, discussed in these sections, are motivated by the fact that the stuck-at fault test set of a circuit is not guaranteed to detect all defects that occur in its QCA implementation. Note that in CMOS circuits stuck-at and stuck-on fault models are studied as they model the physical defects. These examples help generate the interest of the reader on the topic by contrasting both design and testing in old (CMOS) and new (QCA) technologies. Finally, Section 6 concludes the paper.

2. Motivational Example

Before describing the design methodology in Section 3, consider a Boolean circuit (Fig. 2) that implements four non-adjacent minterms (0-cubes) in a 3-cube structure. Thus,

\[ F(A, B, C) = \overline{A}BC + \overline{A}BC + \overline{A}BC + \overline{A}BC = \sum (0, 3, 5, 6) \]  

(1)

A technique, given in [4], works on the following three principles:

(a) Determine if a Boolean function is a majority function or not. The Boolean function denotes a majority function if it’s four minterms (0-cubes) forming a T or inverted - T shape structure is present in the K-map representation of the function (see \( F_1 \) in Fig. 3). Note that \( F(A, B, C) \) in (1) is not a majority function.

(b) If a function is not a majority function, decompose the function into as few majority functions as possible. To do this, find logically adjacent four 0-cubes (of minterms and maxterm) forming a T or inverted-T shape structure in the K-map. If tabular
form is used then we need to find a ternary tree rooted at a minterm or maxterm and three adjacent 0-cubes. In either T or tree structure, at most one maxterm is present.

(c) To reduce the network gate count, decompose a large network into as few three-input subnetworks as possible.

![Diagram of majority network](image)

**Figure 2:** Majority Network for $F(A, B, C) = \sum(0, 3, 5, 6)$

Based on concepts in (b) and (c), a solution for $F(A, B, C)$ requires four $M$ gates. (Refer to Figure 12 in [3]). As shown in Fig. 2, $F_1 = \overline{A}B + BC + \overline{A}C$, $F_2 = AB + BC + AC$, and $F_3 = AB + BC + AC$. Note that concept in section (b) obtains majority function $F_1$ and $F_2$, while concept in (c) gives $F_3$. Fig. 3 uses a K-map to illustrate the steps discussed in [3]. The function $F_1(F_2)$ needs a make-up minterm 010 (111), which is shown in a rectangle in the K-map cell.

Reference [4] considers this problem as standard function #13, which is $\sum(0, 3, 5, 6)$. The function in (1) and #13 contain four non-adjacent minterms. Authors [4] have used the logical proximity of minterms to propose thirteen standards (out of 256) three-variable Boolean functions and have given their efficient majority gate equivalences using K-map representation and Boolean simplification. It is, however, not clear how to extend the principle for more than three variables.

Both references consider two-level realization with an aim to minimize the delay. Our proposed method [9] follows an algorithm-like approach of [3] but gives efficient realization like [4]. The method in [3] is implemented in a tool called Majority Logic Synthesizer, which is integrated within SIS.

![K-map for $F_1$, $F_2$, and $F_3$](image)

**Figure 3:** Steps in (Zhang et al., 2005) gives $F_1$, $F_2$, and $F_3$
3. Design Technique

The design approach in [9] uses the principles given in [3] and [10, 11]. For F₁ and F₂, we do not make any distinction vis-à-vis the method presented in [3]. As shown in Figures 2 and 4, the problem (between a not-so-efficient and efficient realization) lies in how we work or manipulate the terms to get the function F₃. In reliability literature or the processor allocation problem in the hypercube or handling prefixes in the IP routing problem, a common theme is to appropriately apply the concept of “Boolean disjointness” to achieve efficiency. Refer to [12, 17] for details.

To help understand the basic idea, consider again the running example. Maxterms 010 and 111 give \((A+B+C)(\overline{A}+\overline{B}+\overline{C})=\overline{B}+AC+AC\). Use * operator (Table 1(a)) to verify the covering (subsuming) property of the terms so obtained. For example, \((000)*(-0\ -)=(-0\ -)\). Here the 2-cube \((-0\ -)\) denotes \(\overline{B}\). Select \(\overline{B}\). Now, by making remaining terms \(\overline{A}\overline{C}+\overline{A}C\) disjoint with \(\overline{B}\), we get \(A\overline{B}C+A\overline{B}C\). Both 110 and 011 are disjoint with 000 and 101. Drop them or do not select them. We, thus, get only \(\overline{B}\) as \(F₃\).

In another example, consider \(F₃\) having minterms as 100, 101, and 011, and maxterm as 010. Expand the maxterm as \((A+B+C)\). The covering operation gives \(100*1--\) as \(1--\). Select \(A\) and make remaining terms disjoint with it. Thus, we have \(\overline{A}B+\overline{A}C\). Drop 00- because

\[
100*00- = \varphi \text{ (or disjoint)}, \\
101*00- = \varphi, \text{ and } 011*00- = \varphi
\]

<table>
<thead>
<tr>
<th>(a) cover + operator</th>
<th>(b) combine + operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

On the other hand, we find \(100+0-1=\varphi\), \(101+0-1=\varphi\), and \(011+0-1=0-1\). Select 0-1. When more than one cube is selected, apply combine operator (Table 1(b)) to grow the size of the cube, if possible. Thus, \(1--\), 0-1 = --1, which is \(F₃ = C\).

Our approach for finding \(F₃\), has the following steps:

(a) Express maxterms in sum-of-products form by using DeMorgan’s theorem.

Arrange the terms according to their increasing order of cardinality.

**Figure 4**: A Three Majority Gate Realization of \(F(\overline{A}, \overline{B}, \overline{C}) = \sum (0,3,5,6)\)
(b) Apply the cover operator to check for coverage or disjointness.
(c) If one selected, made it disjoint with the remaining terms in (a). Repeat (b) until all
    terms in (a) are considered.

Apply the combine operator to grow, if possible, the selected cubes in (c).

4. Fault Model and Logic Circuit Testing - Basics

In a digital system, when a physical failure happens it creates a fault where the logic
value on one or many lines gets affected or some parameters such as current drawn by
the circuit or delay offered by the circuit change from their normal values. Former fault is
termed as logical fault, while the later is of parametric type. A fault also falls into the
category of (i) transient, (ii) intermittent, and (iii) permanent, depending on the durations
for which it lasts. For a detailed description on these types, refer to [13].

To test a digital system to verify whether it contains some physical failure or not,
we use an appropriate fault model to cover most physical failures depending on the logic
family. For example, the fault models commonly used in a VLSI chip (or CMOS circuits)
include stuck-at, stuck-open, stuck-on, bridging, and delay faults. Refer to any text book
on the topic of fault model and testing for CMOS circuits.

Testing involves applying a sequence of input stimuli, known as test vectors, and
observing the response at primary outputs to check for possible defects (based on the fault
model). The fault is said to be detected if the response is different from an expected
response. A normal requirement of these tests is that they detect a very high fraction of
the modeled faults. There exist a number of techniques to generate test vectors in VLSI
circuits [14]. A Boolean difference (BD) approach defines a complete set of tests for a
given fault and, thus, offers a unique advantage over path sensitization techniques.
However, it was not favored because of the difficulty in the manipulation of algebraic
forms using computers. Currently, reformulating Boolean difference between the un-
faulted and faulted circuits using pseudo-Boolean programming, Boolean satisfiability and
implication graph, and neural network has renewed interest in it as these methods make
network (HNN) approach by extending the concept of neural modeling technique of logic
gates to solve the BD problem. A neural model offers the use of parallel processing for
compute-intensive design automation tasks. However, due to hidden layer, a large portion
of the search space belongs to inconsistent states leading to the problem of scalability and
finding a valid test set becomes increasingly hard. The HNN model overcomes these
problems while retaining the advantages of NN model of digital circuit representation. An
extension of NN concepts for QCA is discussed by Neto et al in [15].

Before attempting to generate tests in a circuit, a common practice is to use “fault-
collapsing” approach based on 1- way or 2-way fault equivalence relationships [13]. This
reduces the number of faults and, hence, the test vectors by a considerable size
(sometimes more than 50%). As an example consider a 2-level NAND/NAND realization
of logic function F = AB + CD. This will have at most 14 single stuck-at faults (SSF), as
there are seven signal lines (nets). By applying fault collapsing, we need to test stuck-at 1
and stuck-at 0 faults on input lines only, which gives eight faults and test vectors 1101,
0111, 0101, and 1010 will test them. In a general combinational circuit, primary inputs
and fan-out branches of the circuit define checkpoints. A test set that detects all single
(multiple) stuck-at faults on all checkpoints of a combinational circuit, also detects all
single (multiple) stuck-at faults in that circuit.

**Example 1:** Figure 5 shows an implementation of the Boolean function F = ab + c using
(a) AND-OR logic and (b) majority gate based on QCA. Note that there is no built in $V_{DD}$
or ground lines in quantum based design; thus e = 0 and f = 1 in majority gate implementation are extra inputs, referred to as control lines. Inputs a, b, and c are non-control lines. Figure 5(a) has 5 signal lines, and at most ten single stuck-at faults. An exhaustive testing will require all input patterns, which are $2^3$ because of three inputs a, b, and c. To achieve 100% single stuck-at fault coverage, the concept of fault collapsing requires checkpoints, which are three primary inputs and six stuck-at faults on them. Table 2 shows the test vectors abc = (110, 101, 100, 010) and $x_i/j$ denotes line $x_i$ stuck-at j. Here $x_i = (a, b, c)$ and j = (0, 1). In case of majority gate implementation, we require only two vectors as abcef = (11100, 00011) for 100% fault coverage. These test vectors are different from those in Table 2. A reduced test set is achievable due to control inputs. Note, the first vector connects all control inputs to 0 and detects a/0, b/0, c/0, d/0, and F/0; while second vector that has control inputs as 1 detects a/1, b/1, c/1, d/1, and F/1. To test all stuck at faults on control lines (namely, e/0, e/1, f/0, f/1), include following two additional test vectors: abcef = (10011, 01100).

The above example does not say explicitly but there exists no fault collapsing relationships between the inputs and outputs of a majority gate [2]. However, an output stuck-at 0 (stuck-at 1) dominates an input stuck-at 0 (stuck-at 1) fault. Thus, any vector that detects an input stuck-at 0 (stuck-at 1) fault will also detect an output stuck-at 0 (stuck-at 1) fault. Chapter 5 in [8] provides a detailed description of some interesting “stuck-at properties” of majority gate–based circuits.

Table 2: Detecting Faults in AND-OR Circuit. Here “×” shows the Test Detects the Fault

<table>
<thead>
<tr>
<th>SSF</th>
<th>Test vector abc</th>
<th>110</th>
<th>101</th>
<th>100</th>
<th>010</th>
</tr>
</thead>
<tbody>
<tr>
<td>a/0</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a/1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>b/0</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b/1</td>
<td></td>
<td></td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c/0</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c/1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>×</td>
</tr>
</tbody>
</table>

(a) AND  (b) OR

Figure 5: An Implementation using (a) AND-OR and (b) Majority Gate for $F = AB + C$

5. QCA Defects and Testing

References [8, 16] provide a detailed description on the types of defects that are likely to occur in QCA devices. In the following, we describe them in brief:

(a) A displacement defect is said to occur if the defective cell is displaced from its original position.

(b) In a cell misalignment defect, the direction of the defective cell is misaligned.

(c) If a defective cell is missing as compared to the original or defect-free case, a cell omission defect occurs.
Figure 6 shows the defects, discussed in above (a) through (c), for a QCA majority gate QCA. Here, we consider a typical case for each defect type using vertical input cell A. However, defects may occur with all possible combinations of displacement of cells with respect to the central cell under different distances and misalignments in different directions. For example, we may have displacement defects for displaced cell(s) (i) A, (ii) B, (iii) C, (iv) A and B, (v) A, B, and C and etc. Similarly, various possibilities of cell misalignments with half a cell or sometimes even smaller in the vertical direction may cause M to malfunction. Extra cell defects and also a cell omission defect on the horizontal input (B in Figure 6) do not affect the functionality of the majority gate; however, a missing cell on any of the vertical inputs causes the output to be dominated only by the horizontal input, i.e., the output is shorted to the horizontal input.

We have mentioned earlier that the basic functionality of QCA is based on the Coulombic interaction among neighboring cells; this depends on the distance as well as the angle between cells. The testing literature has reported that the QCA defects when simulated manifest themselves in different logic values in the fault free and faulty cases. Further, a particular SSF test set could detect all of the simulated defects. For a QCA majority gate, one can apply nine possible SSF test vectors, but the unanswered question is to which of these test sets can detect all of simulated defects.

![Figure 6: (a) Fault Free Majority Gate, (b) Displacement Defect, (c) Misalignment Defect, and (d) Omission Defect. All defects consider vertical input cell A.](image)

![Figure 7: An Example Majority QCA](image)
Table 3: Detecting Faults in the Circuit shown in Figure 7

<table>
<thead>
<tr>
<th>SSF</th>
<th>Test vector ABCDEFG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1000110</td>
</tr>
<tr>
<td>A/0</td>
<td>X</td>
</tr>
<tr>
<td>A/1</td>
<td></td>
</tr>
<tr>
<td>B/0</td>
<td></td>
</tr>
<tr>
<td>B/1</td>
<td>X</td>
</tr>
<tr>
<td>C/0</td>
<td></td>
</tr>
<tr>
<td>C/1</td>
<td></td>
</tr>
<tr>
<td>D/0</td>
<td></td>
</tr>
<tr>
<td>D/1</td>
<td></td>
</tr>
<tr>
<td>E/0</td>
<td></td>
</tr>
<tr>
<td>E/1</td>
<td></td>
</tr>
<tr>
<td>F/0</td>
<td></td>
</tr>
<tr>
<td>F/1</td>
<td></td>
</tr>
<tr>
<td>G/0</td>
<td></td>
</tr>
<tr>
<td>G/1</td>
<td></td>
</tr>
</tbody>
</table>

Example 2: Figure 7 shows an example majority circuit. There are ten lines in this circuit and 20 SSFs. Note, there exists no fault equivalence relationship between input and output of a majority gate. But, in an irredunt, fan-out free combinational majority circuit, it is sufficient to test all SSFs on primary inputs. It is, thus, important to test 14 SSFs on seven primary inputs A through G. Table 3 presents a set of six tests that detect 14 SSFs.

To obtain a test set, (a) consider an opposite value on the fault site and (b) set the remaining inputs to their enabling values such that the fault effect propagates through the gate(s) in the circuit and appears on the primary output where it is observed.

Table 4 illustrates the input vectors that majority gates $M_1$, $M_2$, and $M_3$ receive. For example, $M_1$ receives 000, 001, 011, 100, and 101. Even though these vectors test all SSF faults in $M_1$ and form a complete test set, they fail to detect all simulated defects in $M_1$. In fact, three misalignments and two displacements on the QCA cells in $M_1$ remain undetected. If we include 0100110, $M_1$ will receive 010, which forms a complete set for all simulated defects. Therefore additional test generation is required to cover the defects not covered by the SSF test set. In addition test generation will also be needed for bridging faults on QCA interconnects.

Table 4: Input Vectors for each Majority Gate. Here “×” shows $M_i$ receive the Input.

<table>
<thead>
<tr>
<th>Input vectors</th>
<th>00</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>10</th>
<th>10</th>
<th>10</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>$M_2$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>$M_3$</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

6. Conclusion

This tutorial paper has discussed two important issues with QCA based majority gate. First, it considered the design of a general combinational circuit using $M$. The method in Section 3 is applicable to all the 13 functions [4]. The design of a full adder circuit is presented as running example in this paper. Second, we have discussed CMOS
and QCA faults/defects. Examples show that QCA testing for single stuck-at faults might be easy with the requirement of less number of test sets; but, the stuck-at fault test set of a circuit is not guaranteed to detect all defects that occur in its QCA implementation. Additional test generation is required to cover the defects not covered by the SSF test set.

Note that nanotechnology is an emerging area. Besides QCA, other post-CMOS nano-computing directions include grapheme, carbon nano-tube, FinFETs etc. Testing and design on all types of combinational and/or sequential circuits, or other nano-scale architecture-related issues are being explored. Interested readers may refer [18] through [21] for various research issues/directions on the topic.

References


Suresh Rai joined the Department of Electrical and Computer Engineering at Louisiana State University, Baton Rouge, Louisiana as Assistant Professor in August 1988. He was promoted to the rank of Associate Professor in 1991 and Professor in August 2001.

Dr. Rai has taught and researched in the area of network traffic engineering, ATM, reliability engineering, fault diagnosis, neural net-based logic testing, network security, and steganography. He is a co-author of the book *Wave Shaping and Digital Circuits*, and tutorial texts *Distributed Computing Network Reliability* and *Advances in Distributed System Reliability*; last two published from IEEE Computer Society Press. Dr. Rai has given invited lectures on Internet routing at different universities in Australia and India. He has guest edited a special issue of *IEEE Transactions on Reliability* on the topic *Reliability of Parallel and Distributed Computing Networks*. He was an Associate Editor for *IEEE Transactions on Reliability* from 1990 to 2004. Currently, he is on the editorial board of *International Journal of Performability Engineering*. Dr. Rai has worked as program committee member for several international conferences.

Dr. Rai has published about 128 technical papers in the refereed journals and conference proceedings. He received the best paper award at the 1998 *IEEE International Performance, Computing, & Communication Conference* (Feb. 16-18, Tempe, Arizona; paper title: S. Rai and Y. C. Oh, Analyzing packetized voice and video traffic in an ATM multiplexer). Dr. Rai’s research has been funded by AFOSR, NSF, and ARO.

Dr. Rai has worked/working as referee for papers from various international journals such as *IEEE Transactions on Computers*, *IEEE Computer*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions Reliability*, *IEEE/ACM Transactions Networking*, *IEEE Transactions Communication*, *Int. Journal on Computers and Electrical Engineering*), and several conferences.

Dr. Rai is a Senior Member of the IEEE.