View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 70 (2015) 421 – 427
Procedia Computer Science

4th International Conference on Eco-friendly Computing and Communication Systems (ICECCS 2015)

On Finding a Defect-Free Component in Nanoscale Crossbar Circuits

Malay Kule[a*], Hafizur Rahaman[a*], Bhargab B. Bhattacharya[b]*

[a]Indian Institute of Engineeering Science and Technology, Shibpur, Howrah – 711103, India
[b] Nanotechnology Research Triangle, Indian Statistical Institute, Kolkata – 700108, India

Abstract

We propose a technique for the analysis of manufacturing yield of nano-crossbar architectures for different values of defect percentage and crossbar-size. We provide an estimate of the minimum-size crossbar to be fabricated wherein a defect-free crossbar of a given size can always be found with a guaranteed yield. Our technique is based on logical merging of two defective rows (or two columns) that emulate a defect-free row (or column). Experimental results show that the proposed method provides higher defect-tolerance compared to that of previous techniques.

© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of the Organizing Committee of ICECCS 2015

Keywords: Logic design; nanoscale devices; nano-crossbar; defect-tolerance.

1. Introduction

As the device dimension of the conventional lithography-based VLSI technology gradually shrinks from micron to submicron and further below, nanoscale fabrication is expected to dominate the future high-speed VLSI technology. These molecular-electronic devices can be built with very high densities (10^{12} devices per cm^2), and operated at very high frequencies of the order of *THz* [1]. Such nanoscale devices need new technology for fabrication and newer methodology for circuit design and analysis. Among the various successful nano-electronic devices, resonant-tunnelling diodes (RTD), single-electron transistors (SET), spin transistors, ballistic-electron device, carbon nanotube (CNT), and quantum dot cell [2,3,4] offer the most promising technologies. Recently, an amazing two-terminal device with nanometer dimensions, known as memristor, is shown to possess the properties of both memory and resistor [21, 22, 23]. Such devices can be used to construct high-density resistive memory arrays. Nanowire crossbar arrays that incorporate memristive devices can be integrated on a CMOS subsystem [22], and they have found applications in many promising areas. Fabrication technologies of many such emerging devices and

* Corresponding author. Tel.: [a] +91-9433444980, [a] +91-9836533802, [b] +91-9831425585.
E-mail address: [a]malay.kule@gmail.com, [a]rahaman_h@yahoo.co.in, [b]bhargab.bhatta@gmail.com

1877-0509 © 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of the Organizing Committee of ICECCS 2015
doi:10.1016/j.procs.2015.10.064

systems such as 2D nanowire crossbars, are based on low-cost self-assembly process or its variant [5], which overcomes the limitations of conventional nanoscale lithography [1]. In general, nanoscale crossbar-based architectures provide several advantages such as ease-of-programmability, fault-tolerance, low-cost fabrication, and high device-density. However, there are several potential drawbacks associated with such nano-crossbars that deploy two-terminal devices, especially, the lack of signal gain, and the absence of an inverting function.

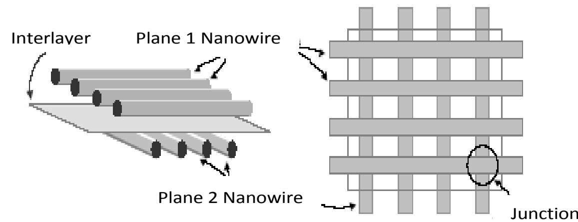

Fig. 1: Nanoscale crossbar

A nanoscale crossbar, as shown in Fig. 1, consists of two parallel planes of nanowire arrays separated by a thin chemical layer with certain electrochemical properties [5, 7-11]. Each plane consists of a number of parallel nanowires of the same type [12]. The wires in one plane cross the wires in the other plane at right angles. The region where two perpendicular wires cross each other is called a junction or a cross-point. Some of these cross-points may become defective at the time of manufacturing or at later stages because of various imperfections. Given such a nanowire crossbar matrix with some defective junctions, a sub-matrix consisting of defect-free cross-points may be used instead of discarding the chip as a whole [13].

Various work on the study of defect-tolerance in nanoscale crossbars were reported in the past [6-15, 17-18]. A graph-based method for finding a defect-free sub-matrix of size ($k \times k$) in a crossbar of size ($n \times n$), where $k \leq n$, appears in [24]. An application-independent defect-tolerant scheme is described for reconfigurable nano-architectures [16]. Some variation- and defect-aware mapping techniques for logic synthesis with defective nanoscale crossbars have been proposed in [19, 20]. Test and repair techniques for defective (Memristor Look-Up Table) MLUT-based nanoscale asynchronous crossbar architectures are reported in [23].

Discarding the whole crossbar structure in the presence of one or more cross-point defects is not economical because of the resulting yield-loss. A much better option would be to identify the usable (defect-free) sub-structures in the presence of some defective cross-points, so that they can still be used for logic synthesis. During the fabrication of nano-crossbar architectures, it has been observed that the defects that arise due to shorts between horizontal and vertical nano-wires are much less than those caused by missing or misplaced switches. Hence, "open" defects are more likely to occur in the crossbar. In this work, we attempt to locate a defect-free sub-matrix of switches of size ($k \times k$) in a given ($n \times n$) defective 2D-crossbar in the presence of possible stuck open faults. In other words, we restrict ourselves to inoperative or imperfect switches only, rather than shorted-wire defects, or interconnect-opens. Given a defective crossbar, we also attempt to maximize the value of k so as to render a largest-size defect-free sub-matrix useful. We perform simulation experiments for different defect densities, and results indicate that the proposed technique provides higher defect-tolerance compared to prior work [24].

The rest of the paper is organized as follows. Section II describes the proposed methods. Section III presents the experimental set-up and analysis of results. Concluding remarks and future work are briefed in Section IV.

2. Proposed Method

2.1. Representation of a crossbar structure and the defects therein

As stated earlier, we consider only stuck-open faults in a crossbar structure, an example of which is shown in Fig. 2(a). The encircled-junctions are configurable, whereas, the star-marked junctions are assumed to be defective. Such a crossbar structure can be represented by a bipartite graph [16, 24] as shown in Fig. 2(b). Here, the set of nanowires in two planes are represented by two disjoint set of nodes; an edge between two nodes exists when the junction between the two corresponding wires is correctly configurable. The missing edges between two partitions indicate defective junctions. The bipartite-complement [24] of the representative graph of Fig. 2(b) is shown in Fig. 2(c),

where each edge represents a defective junction. Usually, more than 50% of junctions turn out to be defect-free, and hence, it is more convenient to handle the complementary graph rather than the original one.

2.2. Merging defective rows or columns

In order to make the best use of a defective crossbar, we first select two rows (or columns) of nanowires such that their defect-sets are disjoint, and then merge them together to construct a single defect-free row (or column). We iterate this process until all defects become invisible. The merged row (column) can be treated as a single logical input (output) during function mapping. We propose two algorithms (Algorithm 1 and Algorithm 2) to implement recursive merging. Algorithm 1 chooses the most-defective row (column) for merging with a compatible least-defective row (column), whereas, Algorithm 2 selects two most-defective rows (columns) that are compatible with each other, as potential candidates for merging.

The basic merging technique is illustrated in Fig. 3. Two wires with disjoint defect-positions can be merged logically. Equivalently, in the bipartite graph, two compatible nodes of a partition are merged into one to form a hyper-node such that they collectively produce an equivalent defect-free row or column.

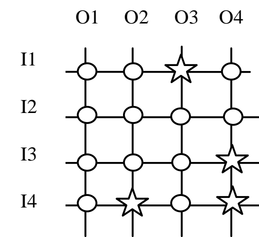

Fig. 2(a): 4 × 4 nanoscale crossbar with 4 defects

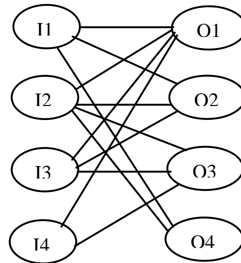

Fig. 2(b): Representative Graph

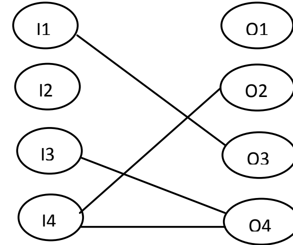

Fig. 2(c): Bipartite complement of Fig. 2(b)

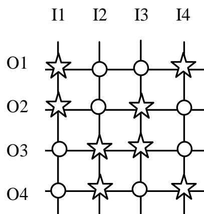

Fig. 3(a): Example crossbar with defects

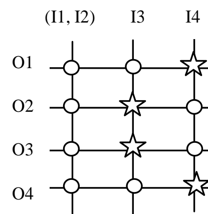

Fig. 3(b): Example crossbar after merging I1 and I2

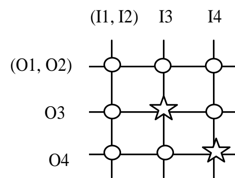

Fig. 3(c): Example crossbar after merging O1 and O2

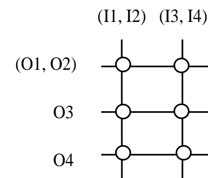

Fig. 3(d): Example crossbar after merging I3 and I4

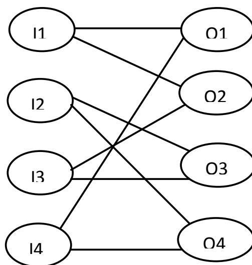

Fig. 4(a): Bipartite complement of the crossbar of figure 3(a)

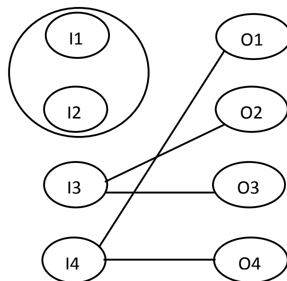

Fig. 4(b): Effect of merging I1 and I2 on the bipartite complement

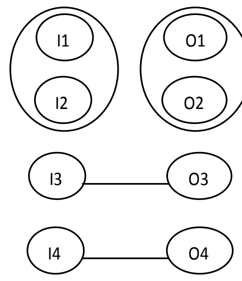

Fig. 4(c): Effect of merging O1 and O2 on the bipartite complement

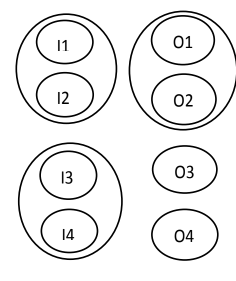

Fig. 4(d): Effect of merging I3 and I4 on the bipartite complement

First, column I1 is logically merged with column I2 to create a resulting defect-free-group (I1, I2), as shown in Fig. 3(b), where the defects of I1 can be compensated by I2 and vice versa. Next, row O1is merged with row O2 as shown in Fig. 3(c). Finally I3 and I4 are merged into one to form a defect-free crossbar as shown in Fig. 3(d). The above merging process, as reflected in the bipartite-complement of the crossbar graph, is illustrated in Fig. 4.

2.3. Merging with Algorithm 1

Algorithm 1 selects the most-defective row (column) and a compatible least-defective row (column) for possible merging. The algorithm basically attempts to find a maximally-balanced bipartite sub-graph of the original crossbar graph. Since in the complementary graph, the edges represent defective junctions, our objective can be envisaged as that of making the complementary graph devoid of any edge. The algorithm first searches for the nodes that have degree zero. They are directly taken into the final solution. Instead of deleting the nodes to remove edges, two nodes in the same partition with disjoint defect-positions are merged to form a hyper-node such that these two nodes effectively produce a defect-free wire as illustrated in Fig. 4. In this algorithm, we merge two columns, and next, two rows such that we obtain a nearly-balanced bipartite graph. This technique is likely to reduce the probability of false rejection. The merging process of columns and rows is repeated alternately until no further merging is possible. Next, the nodes which still have some incident edges are examined and the one with the maximum degree from each partition is deleted alternately, until the graph is completely void of any edges. The algorithm can be formally described as follows.

Algorithm 1

1. Read the size of the crossbar matrix (n) and the defect percentage (z)
2. Generate the random crossbar matrix with the required no. of 0's (indicating defects)
3. Set no. of columns merged (NCM) ← 0, no. of rows merged (NRM) ← 0, possibility of merging (PM) ← 'yes'
4. **While** (PM = = 'yes')
   (a) Mark the columns having no defect or having defects at a junction of already merged rows as **'free'**
   (b) **While** not a single column merging is done and there are some unmarked columns
      i. ***pos*** ← the unmarked column with the maximum number of defects
      ii. **If** ***pos*** can be merged at all
         A. ***mer*** ← the unmarked column with the minimum defect that can be merged to column ***pos***
         B. Mark ***pos*** and ***mer*** as **'free'**
         C. NCM ← NCM + 1
      iii. **Else** mark ***pos*** as **'np'**
         End if
      End While
   (c) Mark the rows having no defect or having defects at a junction of already merged columns as **'free'**
   (d) **While** not a single row merging is done and there is some unmarked row
   i. ***pos*** ← The unmarked row with maximum number of defects
   ii. **If** ***pos*** can be merged at all
      A. ***mer*** ← the row with minimum defect that can be merged to row ***pos***
      B. Mark ***pos*** and ***mer*** as **'free'**
      C. NRM ← NRM + 1
   iii. **Else** mark ***pos*** as **'np'**
   End While
   (e) unmark columns and rows marked as **'np'**
   (f) If neither merging of row nor columns could be done, set **PM** ← **'no'**
End While
5. **TFC** ← total number of columns marked as **'free'**
6. **TFR** ← total number of rows marked as **'free'**
5. No. of effective defect-free rows (u) ← (TFR – NRM)
6. No. of effective defect-free columns (t) ← (TFC – NCM)
7. **While** there is some unmarked column or row

If ($t > u$)
While ($t > u$ and there is some unmarked column)
i. Find defective column with maximum no. of defects
ii. Delete the column, i.e., mark it as **'deleted'**
iii. Set resulting defect free rows as **'free'**
iv. $u \leftarrow u$ + number of rows made **'free'** in Step iii
End While
Else if ($u > t$)
While ($u > t$ and there is some unmarked row)
i. Find defective row with maximum no. of defects
ii. Delete the row i.e. mark it as **'deleted'**
iii. Set resulting defect free columns as **'free'**
iv. $t \leftarrow t$ + number of columns made **'free'** in Step iii
End While
Else
i. Find some unmarked defective row or column with maximum no. of defects
ii. Delete the selected row or column i.e. mark it as **'deleted'**
iii. **If** column is deleted
A. Set resulting defect free rows as **'free'**
B. $u \leftarrow u$ + number of rows made **'free'** in Step A
Else
A. Set resulting defect free columns as **'free'**
B. $t \leftarrow t$ + number of columns made **'free'** in Step A
End If
End If
End While
9. $x \leftarrow$ minimum of t and u
10. If there is even 1 defect-free junction in the entire crossbar, k is larger of 1 and x
11. Print k
12. Stop

2.4. Algorithm 2

Algorithm 2 selects two compatible columns or rows that are most-defective at every step. Merging two columns (or two rows) with a large number of defective junctions may also produce good results. Note that in this case, both of the defective columns (rows) can be tied together in one step. However, this approach suffers from too many false rejections at initial stages. At subsequent stages it is likely to perform better because the leftover wires will have a fewer number of defects.

3. Experimental Results

The algorithms have been implemented using Turbo C++ on Intel Core 2 duo 3.1 *GHz* processor with 2 *GB* of RAM under Windows operating system.

3.1. Construction of the largest defect-free sub-crossbar of size ($k\times k$) for a given defect-percentage

In our simulation experiments, we consider nanowire crossbars with sizes ($n\times n$) varying from (40×40) to (120×120) incremented in steps of 40. The defects are injected at random positions for a given defect-percentage, which is varied from 0% to 30% incremented in step of 5. We observed that for defects in the range of 15% to 25%, our algorithms produce better results compared to previous approaches [24]. The experiment is repeated 1000 times for each value of defect-percentage, and the floor of the average is reported in Table 1.

Table 1: Variation of k with defect-percentage

Defect %	Crossbar-size (n) = 40				Crossbar-size (n) = 80				Crossbar-size (n) = 120			
	Algo1 [24]	Algo2 [24]	Algo1 (Prop.)	Algo2 (Prop.)	Algo1 [24]	Algo2 [24]	Algo1 (Prop.)	Algo2 (Prop.)	Algo1 [24]	Algo2 [24]	Algo1 (Prop.)	Algo2 (Prop.)
0	40	40	40	40	80	80	80	80	120	120	120	120
5	22	23	25	26	34	35	38	38	43	46	48	49
10	17	17	21	21	22	25	29	29	28	29	38	39
15	14	15	18	20	18	20	27	28	22	24	38	39
20	11	12	17	17	14	16	28	30	17	20	39	40
25	9	10	16	17	13	14	28	30	14	16	34	35
30	7	9	16	16	10	11	25	28	13	14	30	31

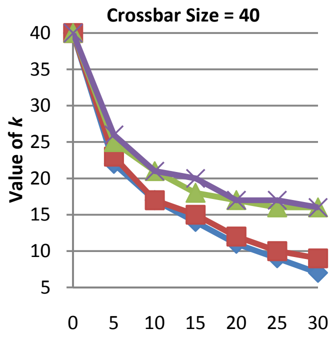

Fig. 5(a): k *vs*. defect % graph for (40 × 40) crossbar

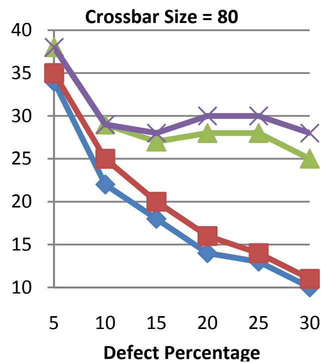

Fig. 5(b): k *vs*. defect % graph for (80 × 80) crossbar

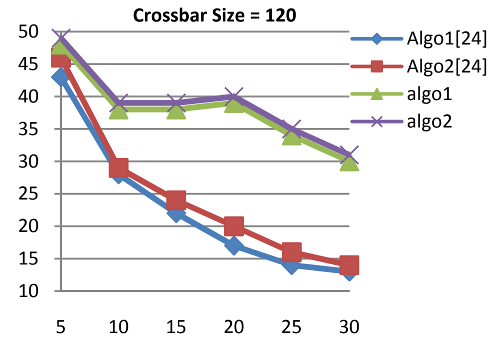

Fig. 5(c): k *vs*. defect % graph for (120 × 120) crossbar

Table 2: Variation of k with crossbar-size

Crossbar Size	Defect % = 15				Defect % = 20				Defect % = 25			
	Algo1 [24]	Algo2 [24]	Algo1 (Prop.)	Algo2 (Prop.)	Algo1 [24]	Algo2 [24]	Algo1 (Prop.)	Algo2 (Prop.)	Algo1 [24]	Algo2 [24]	Algo1 (Prop.)	Algo2 (Prop.)
20	9	9	12	12	8	8	11	11	7	8	10	11
40	14	15	18	20	11	12	17	17	9	10	16	17
60	16	17	22	24	12	14	22	24	11	13	22	24
80	18	20	27	28	14	16	28	30	13	14	28	30
100	20	22	34	36	16	18	34	37	14	15	31	33
120	22	24	38	39	17	20	39	44	14	16	34	35

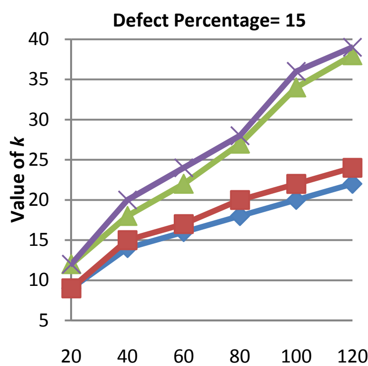

Fig. 6(a): k vs. crossbar-size graph for defect percentage =15

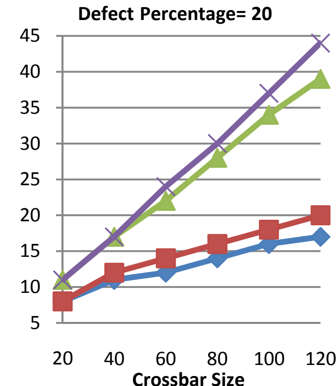

Fig. 6(b): k vs. crossbar-size graph for defect percentage =20

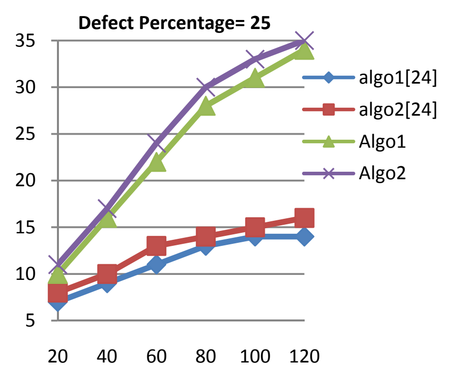

Fig. 6(c): k vs. crossbar-size graph for defect percentage =25

From Fig. 5, it is evident that our algorithms produce improved results compared to previous techniques [24], particularly when the defect-percentage exceeds 10%. Algorithm 1 and Algorithm 2 yield similar results with some improvements in the case of Algorithm 2.

3.2. Variation of defect-free sub-crossbar size (k) with crossbar-size

We also observe how the value of k changes with the increase of crossbar-size for some given defect-percentage. In Table 2, we have considered defect-percentage = 15%, 20%, and 25%. The crossbar-size is varied from (20×20) to (120×120) in steps of 20. Results shown in Fig. 6 demonstrate that the proposed algorithms work better with the increase in crossbar-size.

4. Conclusion

We have proposed two heuristics to identify the largest defect-free crossbar ($k \times k$) in a nanoscale crossbar architecture with certain defects. Conversely, we can estimate, for a given value k and defect-percentage, the smallest value of n such that a ($k \times k$) defect-free crossbar can effectively be constructed from an ($n \times n$) defective crossbar. The proposed column- and row-merging technique yields an improved value for k (maximum defect-free crossbar-size) compared to earlier approaches. However, our algorithms can only handle stuck-open defects at crossbar-junctions. The study may be extended further to take care of stuck-closed defects in a future work.

References

1. M. Butts, A. DeHon, and S. C. Glodstein, “Molecular Electronics: Devices, Systems and Tools for Gigagate, Gigabit Chips”, *ICCAD* 2002, pp. 443-440.
2. A. Bachtold, P. Harley, T. Nakanishi, and C. Dekker, “Logic Circuits with Carbon Nanotube Transistors”, *Science*, vol. 294, pp. 1317-1320, 2001.
3. Y. Cui and C. M. Lieber, “Functional Nanoscale Electronics Devices Assembled using Silicon Nanowire Building Blocks”, *Science*, vol 291, pp. 851-853, 2001.
4. Y. Huang, X. Duan, Y.Cui, L. J. Lauhon, K. H. Kim, and C. M. Leiber, “Logic Gates and Computation from Assembled Nanowire Building Blocks”, *Science*, vol. 294, pp. 1313-1317, 2001.
5. M. M Ziegler and M. R. Stan, “Design and Analysis of Crossbar Circuit for Molecular Nanoelectronics”, *IEEE Conference on Nanotechnology* (2002), pp. 323-327.
6. Y. Chen, G.-Y. Jung, D. A. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. S. Williams, “Nanoscale Molecular Switch Crossbar Circuits”, *Nanotechnology*, vol. 14, pp. 462-468, 2003.
7. T. Hogg and G. Snider, “Defect-Tolerant Adder Circuits with Nanoscale Crossbars”, *IEEE Trans. Nanotechnology*, vol. 5(2) 2006.
8. H. Naeimi and A. DeHon, “A Greedy Algorithm for Tolerating Defective Crosspoints in NanoPLA Design”, *IEEE Int’l Conf. Field-Programmable Technology* (FPT04), 2004, pp. 49-56.
9. S. G¨oren, H.F. Ugurdag, and O. Palaz, “Defect-Aware Nanocrossbar Logic Mapping through Matrix Canonization Using Two-Dimensional Radix Sort,” *ACM JETC*, vol. 7, no. 3(12), 2011.
10. A. DeHon and M. J.Wilson, “Nanowire-Based Sublithographic Programmable Logic Arrays”, *FPGA* 2004, California, USA.
11. A. DeHon, “Array-Based Architecture for FET-Based, Nanoscale Electronics”, *IEEE Trans. Nanotechnolog*y, vol. 2(1), 2003.
12. P. J. Kuekes, J. R. Heath, and R. S. Williams, “Molecular Wire Crossbar Memory,” Patent no. 6128214 (Hewlett-Packard), 2000 (9).
13. M. B. Tahoori, “Defect-Tolerance in Crossbar Array Nano-Architectures”, *Emerging Nanotechnologies: Test, Defect Tolerance, and Reliability*, Springer, pp. 121-151, 2007.
14. Y. Su, W. Rao, “Defect-Tolerant Logic Mapping on Nanoscale Crossbar Architectures and Yield Analysis”, *24th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pp. 322-330, 2009.
15. J.-S. Yang, R. Datta, “Efficient Function Mapping in Nanoscale Crossbar Architecture”, *IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pp. 190-196, 2011.
16. M. B. Tahoori, “Application-Independent Defect-Tolerance of Reconfigurable Nanoarchitectures,” *ACM JETC*, vol. 2(3), pp.197–218, 2006. (21).
17. M. B. Tahoori, “Low-Overhead Defect-Tolerance in Crossbar Nano-architectures,” *ACM JETC*, vol. 5(2), Article 11, 2009.
18. Y. Yellambalase and M. Choi, “Cost-Driven Repair Optimization of Reconfigurable Nanowire Crossbar Systems with Clustered Defects,” *Journal of Systems Architecture*, 54(8):729-741, 2008.
19. M. Zamani and M. B. Tahoori, “Variation-aware Logic Mapping for Crossbar Nano-architectures,” *IEEE 16th Asia and South Pacific Design Automation Conference* (ASP-DAC), 2011.
20. M. Zamani and M. B. Tahoori, “Reliable Logic Mapping on Nano-PLA Architectures”, *GLSVLSI*, USA, May 3-4, 2012.
21. L. O. Chua, “Memristor – The Missing Circuit Element”, *IEEE Transactions on Circuit Theory*, vol. CT-18, No. 5, September 1971.
22. H. Owlia, P. Keshavarzi, and A. Rezai, “A Novel Digital Logic Implementation Approach on Nanocrossbar Array using Memristor-based Multiplexers”, *Microelectronics Journal*, 45 (2014) 597-603.
23. V. Hongal, R. Kotikalapudi, and M. Choi, “Design, Test, and Repair of MLUT (Memristor Look-Up Table) Based Asynchronous Nanowire Reconfigurable Crossbar Architecture”, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2014.
24. A. A. Al-Yamani, S. Ramsundar, and D. K. Pradhan, “*A Defect-Tolerance Scheme for Nanotechnology Circuits*”, *IEEE TCAS-I,* vol. 54 (11), pp. 2402-2409, 2007 (25).