International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 1 (2018) pp. 298-303

A Defect Tolerance Scheme for Nanoscale Crossbar Memory

Yoon-Hwa Choi

Department of Computer Engineering,
Hongik University, Seoul, Korea.

Abstract

Designing a nanoscale crossbar memory system in the presence of high defect rates poses a significant challenge. Multiple crossbar modules that share the same address space can be used as a simple memory architecture to tolerate the defects in the crossbars. In this paper, we presents a defect tolerance scheme for a nanoscale crossbar memory. Redundancies at both the module level and row/column level in each module are used to enhance defect tolerance. Defects affecting an entire row or column are also taken into account to cover more realistic defect patterns. Computer simulation results demonstrate that the proposed scheme can realize a functioning memory with reduced overhead compared to an existing simple redundancy scheme.

Keywords: Crossbar memory; defect tolerance; molecular electronics

INTRODUCTION

Although CMOS technology is dominating the digital electronic circuits, a number of new challenges have been made by using bottom-up chemical assembly [1][2]. Among different proposed technologies, nanoscale devices based on crossbars of nanowires are widely discussed in literature [3-6]. The crossbars can be used to implement memories and logic functions.

Self-assembled nanoscale memories, however, are expected to suffer from high defect rate compared to the traditional memories. Several defect tolerance techniques have been proposed to deal with defects in crossbar-based memories and logic arrays [7-15]. For a crossbar-based memory spare rows and columns alone might be insufficient to achieve defect tolerance due to the under-utilization of spares in the case of high defect rates [7]. Spare modules alone might also show unacceptable performance unless sufficient redundancy is present.

A hierarchical sparing is used to provide runtime protection against device failures in crossbar memory architecture in [9]. A combination of error correcting code(ECC) and reconfiguration implemented in a hierarchical fashion is used. A multijunction fault-tolerance architecture for nanoscale crossbar memories was proposed in [12]. Redundancies in the rows and columns of a molecular switch crossbar array are utilized, where redundancy is defined as the number of nanowires connected to the same row (column) electrode representing a single wordline (bitline) in the crossbar memory.

A defect tolerance technique exploiting hybrid redundancy, a combination of hard redundancy and soft redundancy generated by runtime utilization of spatial/temporal access locality in memory access was proposed in [13]. A memory consists of banks, each of which is divided into blocks/sub-blocks. A lookup table is used to monitor the sub-block status for hybrid redundancy allocation.

Logical merging of two defective rows (or two columns) that emulate a defect-free row (or column) has been employed to achieve defect tolerance in [14]. Two heuristics are proposed to identify the largest defect-free crossbar in a nanoscale crossbar architecture with certain defects.

Defect tolerance in molecular memory with defect maps has been introduced in [8]. Space efficiency in defect maps is achieved by the use of Bloom filters, a randomized data structure for membership queries with a tunable false positive probability [15]. A nanoscale memory consisting of multiple memory-modules sharing an address space has also been proposed to achieve improved memory configurability over a triple modular redundancy. However, it might incur relatively high hardware overhead, when the defect rate is high, due to the lack of flexibility in address mapping.

In this paper, we present a two-level redundancy scheme for tolerating defects in crossbar-based molecular memories. Redundancies at both module-level and row/column-level are employed to configure a functioning memory even with relatively high defect rate. A memory here consists of multiple modules that share the same address space, where each of the modules is constructed as an augmented crossbar array. Spare rows and columns in each crossbar module are used to replace the most defective rows and columns in the module. Spare modules with address remapping are then used to further enhance the performance. All the supporting logics are provided by the reliable CMOS technology as shown in hybrid technology [16] to cope with the difficulties in implementing molecular crossbar memories.

NANOSCALE MEMORY MODULES

Multiple memory modules that share the same address space have been used to achieve defect tolerance in nanoscale crossbar architecture [8]. Bloom filters are employed to realize defect maps with considerable reduction in hardware overhead. The redundant memory architecture (RM for convenience) consists of k memory modules of 2^m cells, k Bloom filters BFs, and a selection logic as shown in Fig. 1. Each memory module has its own defect map, implemented using a Bloom filter with CMOS technology. In the figure,

BF_i stores the defect map of the nano-module *i*. Each cell can be addressed using a tuple (*x*, *y*), representing the row and column addresses, respectively. A defect-free cell is then selected by the selection logic with the aid of the Bloom filters. In the figure, *k* modules of 2^m cells each are used to construct a memory of 2^m cells.

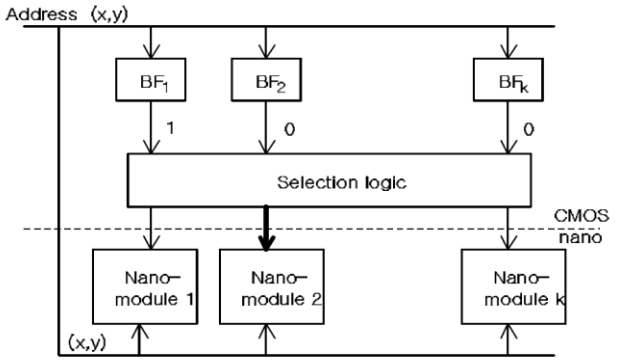

Figure 1. A redundant nanoscale memory

For a read or write operation on address (*x*, *y*) the *k* Bloom filters are first queried to see if the corresponding cells are defective. Based on the query results the selection logic will select the module(s) for the read/write operation. The memory might have some potential to tolerate runtime faults. If more than one nanoscale module is defect-free at address (x,y), the corresponding locations can be treated as a cell with spare(s) during normal operation.

Let *p* be the probability that each crosspoint of 2^nx2^n crossbar array is defective. Then the expected number of defective crosspoints for a 2^nx2^n crossbar array is $2^{2n}p$. If *k* modules are employed to achieve defect tolerance, an access with address (*x*,*y*) may fail with probability p^k. That is, the defect rate *p* in that case is effectively reduced to p^k with the added redundancy.

Suppose that a broken nanowire is modeled as a defect at its closest crosspoint. Let *q* represent the conditional probability that a crosspoint has a broken nanowire when it is defective. Then the expected number of faulty crosspoints is approximately $2^{2n}\ p(1\text{-}q)+2^{2n}\ pq2^n$ for a relatively small *p*. A broken wire causes its corresponding row or column unusable, roughly equivalent to 2^n faulty crosspoints. This can be treated as an increase in defect rate from *p* to $p' \approx p(1\text{-}q) + pq2^n$. The increase in defect rate becomes significant as *q* and *n* increase.

We have estimated the configurability of a 2^nx2^n memory using *k* modules in the redundant nanoscale memory (RM) for two different cases: i) for given *p* and *q*, ii) for random and uniform distribution with defect rate *p'*. The results for *n*=5 are shown in Table 1, where the number before slash represents the configurability for the former. Although some differences can be noted, the number of modules required for a near perfect configurability is almost the same for the given values of *q*.

Table 1. Configurability of a 2^nx2^n memory for various *p*,*q*, and *k* for RM(*n*=5).(a) *q*=0.01, (b) *q*=0.03, (c) *q*=0.05

p	*k*=2	3	4	5
0.01	0.86/0.85	1.00/1.00	1.00/1.00	1.0/1.0
0.03	0.26/0.20	0.95/0.95	1.00/1.00	1.0/1.0
0.05	0.03/0.01	0.76/0.76	0.98/0.98	1.0/1.0

p	*k*=2	3	4	5
0.01	0.76/0.68	0.99/0.99	1.00/1.00	1.0/1.0
0.03	0.14/0.03	0.85/0.86	0.99/0.99	1.0/1.0
0.05	0.01/0.00	0.55/0.44	0.94/0.93	0.99/0.99

p	*k*=2	3	4	5
0.01	0.66/0.49	0.98/0.98	1.00/1.00	1.0/1.0
0.03	0.06/0.00	0.74/0.64	0.97/0.97	1.0/0.99
0.05	0.00/0.00	0.30/0.17	0.83/0.80	0.98/0.98

Table 2. Configurability of a 2^nx2^n memory for various *p*,*q*, and *k* for RM (*n*=6) (a) *q*=0.01, (b) *q*=0.03, (c) *q*=0.05

p	*k*=2	3	4	5
0.01	0.43/0.31	0.98/0.98	1.00/1.00	1.0/1.0
0.03	0.00/0.00	0.69/0.64	0.98/0.98	1.0/1.0
0.05	0.00/0.00	0.20/0.15	0.86/0.84	0.99/0.98

p	*k*=2	3	4	5
0.01	0.12/0.04	0.92/0.91	1.00/1.00	1.0/1.0
0.03	0.00/0.00	0.26/0.08	0.84/0.82	0.98/0.98
0.05	0.01/0.00	0.01/0.00	0.40/0.26	0.87/0.85

p	*k*=2	3	4	5
0.01	0.08/0.00	0.84/0.75	1.00/0.99	1.0/1.0
0.03	0.00/0.00	0.07/0.00	0.62/0.47	0.92/0.90
0.05	0.00/0.00	0.00/0.00	0.07/0.01	0.50/0.36

The number of nanoscale modules to construct a functioning memory of a given size increases as the module size grows or the defect rate increases. In the case of *p*=*q*=0.03 and *n*=5, for example, at least four modules are needed to achieve a configurability over 0.99. If *n* is increased to 6, however, the same configurability cannot be achieved even with five modules as shown in Table 2. The overhead required for achieving high configurability might be problematic unless the defect rate is lowered. For a memory to be configured in molecular-based crossbars, it would be desirable to find a low-overhead alternative without lowering the configurability.

PROPOSED DEFECT TOLERANCE SCHEME

In the proposed defect tolerance scheme, a memory of 2^n x2^n consists of *k* nanoscale modules (nano-modules or modules, for short) of $(2^n+r)x(2^n+r)$, as shown in Fig. 2, where all the *k* modules share the same address space and each module can invert the address inputs for address remapping. Defect tolerance in this paper is said to be achieved if a functioning

memory of 2^n x2^n can be constructed in the given *k* crossbars of (2^n+r)x(2^n+r). The *r* extra rows and columns in each nano-module are used to eliminate the most defective rows and columns.

In configuring a crossbar memory, we first choose 2^n rows and 2^n columns by taking defect locations into account. This makes effective defect rate greatly reduced. Here effective defect rate is defined as the defect rate after removing the most defective *r* rows and *r* columns in a single crossbar module of (2^n+r)x(2^n+r).

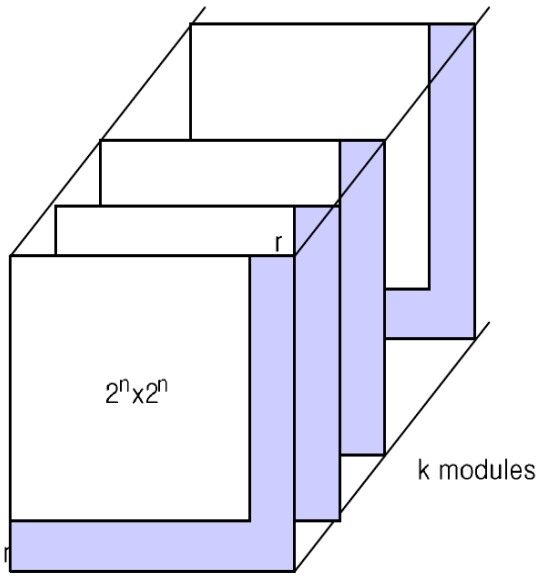

Figure 2. Multiple modules with spare rows/columns

A defect tolerance scheme with spare rows and columns alone would be insufficient to achieve high configurability unless defect rate is relatively low. Even with *k* nano-modules of augmented crossbars there is still some room for improvement in memory configurability.

To further reduce the number of modules required for memory construction we employ address remapping (by inverting address inputs) in the case of a failure. The remapping is illustrated in Fig. 3, where a 2^2x2^2 functioning memory is formed in two (2^2 +1) x (2^2 +1) augmented crossbars (in Fig, 3(b)). In Fig. 3, the most defective row and column removed are highlighted with thick lines. In the upper two crossbars (Fig. 3(a)) of the illustration, both modules, modules 0 and 1, after eliminating the most defective row and column, still have a defective crosspoint at address 0110. This results in a failure in memory configuration. In Fig. 3(b), however, we invert one of the address inputs (here, the second address bit) of the module 0 to cope with the defects. It remaps the defective crosspoint 0110 (in module 0) to 0010 in Fig. 3(b). Apparently, all other rows of the crossbar array (module 0) are affected by the remapping.

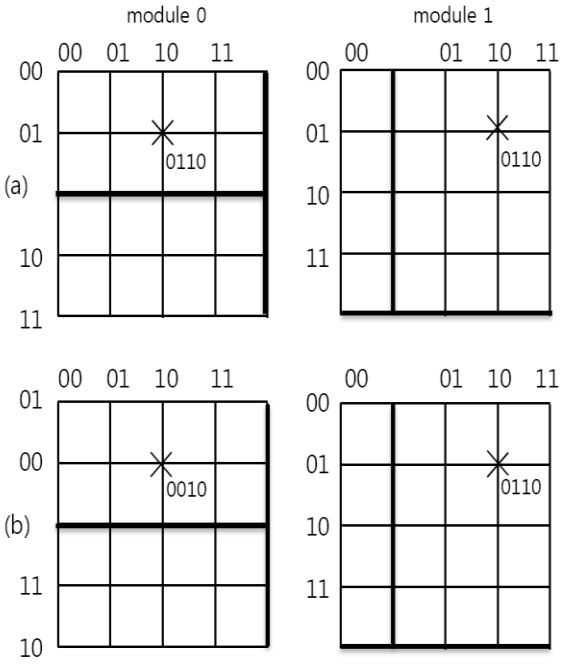

Figure 3. Address remapping for defect tolerance

The proposed defect tolerance scheme consists of two phases as follows. Phase 2 is only needed when phase 1 fails in constructing a functioning memory.

Phase 1 (Select 2^n rows and 2^n columns of each module to form a 2^nx2^n memory)

a) Repeat until no spare rows and columns.
   (1) For each module, count the number of defective cross points in each row and column, excluding the rows and columns removed.
   (2) Remove the most defective row or column in the module.

b) (b) Check to see if *k* modules of the reduced crossbar arrays form a functioning memory.

Phase 2 (Invert address inputs to enhance configurability if phase1 fails)

a) Try all possible row assignments for a single module, by selecting one at a time, each of the *k* modules.

b) If Phase 2(a) fails, try all possible column assignments for a single module, by selecting one at a time, each of the *k* modules.

In phase 2(a)(b), address remapping is applied only to a single module, one at a time, while the remaining *k*-1 modules assume their initial assignments. The search process stops when a satisfactory assignment is found. The best performance can be achieved if an exhaustive search is applied to the entire *k* modules and to both rows and columns altogether. A simplified version, however, is chosen here

since a notable improvement can be observed even with a greatly reduced search space.

Computer simulation is conducted to evaluate the performance of the proposed scheme in terms of configurability of a functioning memory of 2^nx2^n. Defects are generated randomly and independently based on the given defect rates. In the simulation, 1,000 defect patterns are used to obtain the statistical data for relatively small values of *n*. The reason for not choosing a large crossbar is that defect tolerance schemes based on spare rows and columns alone in molecular crossbar memory are less cost-effective as *n* increases as shown in [7]. Hence in constructing a large memory, it is desirable to use relatively small crossbars, as far as the crossbar overhead is concerned. For a fair comparison, computer simulation is also performed for the redundancy scheme RM.

The number of modules required for achieving a near perfect configurability for various *p*, *q*, *k*, and *r* when *n*=5 are shown in Table 3, where the three numbers separated by slashes represent the configurability for *r*=0/2/4, respectively. For $0.01 \leqq p, q \leqq 0.05$, the configurability is almost perfect even with one or two spare modules if proper remapping is done when phase 1 fails.

Table 3. Configurability for various *p*,*q*, *k* and *r* in constructing a 2^nx2^n memory for the proposed scheme (*n*=5) *q*=0.01, (b) *q*=0.03, (c) *q*=0.05

p	k=2	k=3
0.01	0.993/1.000/1.000	1.000/1.000/1.000
0.03	0.947/1.000/1.000	1.000/1.000/1.000
0.05	0.768/1.000/1.000	0.998/1.000/1.000

p	k=2	k=3
0.01	0.952/1.000/1.000	0.997/1.000/1.000
0.03	0.697/0.999/1.000	0.996/1.000/1.000
0.05	0.305/0.970/1.000	0.982/1.000/1.000

p	k=2	k=3
0.01	0.903/1.000/1.000	0.998/1.000/1.000
0.03	0.448/0.988/1.000	0.982/1.000/1.000
0.05	0.090/0.838/0.990	0.925/1.000/1.000

Similar trends are observed for *n*=6 as shown in Table 4. We have, however, noted that *k* strongly depends on *n*. When p=0.05, q=0.05, and r=4, for example, only two modules are needed to construct a functioning memory with probability over 0.99 for *n*=5. If *n* becomes 6, even with three modules the result is unacceptably low (0.854).

Table 4. Configurability for various *p*,*q*, *k* and *r* in constructing a 2^nx2^n memory for the proposed scheme (*n*=6) *q*=0.01, (b) *q*=0.03, (c) *q*=0.05

p	k=2	k=3
0.01	0.929/1.000/1.000	1.000/1.000/1.000
0.03	0.417/0.991/1.000	0.995/1.000/1.000
0.05	0.001/0.122/0.431	0.972/1.000/1.000

p	k=2	3
0.01	0.598/1.000/1.000	0.995/1.000/1.000
0.03	0.015/0.578/0.958	0.909/0.993/1.000
0.05	0.000/0.004/0.099	0.596/0.929/0.997

p	k=2	3
0.01	0.351/0.983/1.000	0.988/1.000/1.000
0.03	0.000/0.110/0.602	0.737/0.948/0.997
0.05	0.000/0.000/0.001	0.061/0.459/0.854

Overhead Estimation

The simulation results show that a functioning crossbar memory can be implemented with much fewer nano-modules compared to the simple redundancy scheme RM. We now determine an adequate number of modules for given values of *n*, *r*, *p*, and *q*. We have considered two different cases (a) *p*=0.03 and *q*=0.03, (b) *p*=0.05, *q*=0.03, when *n*=5 and 6. The resulting configurabilities for the above two schemes under comparison in case (a) are shown in Fig. 4, where S_0, S_4, and RM represent the proposed scheme with *r*=0, the proposed scheme with *r*=4, and the simple redundancy scheme, respectively. We can notice a significant improvement in configurability with four extra rows and columns.

Similar results are also observed for *p*=0.05 and *q*=0.03 (case (b)) as shown in Fig. 5, where the scheme RM requires at least 5 modules when *n*=5. If *n* is increased to 6, 7 modules are required to achieve near perfect configurability. The proposed scheme shows better performance even with two modules for *n*=5. It needs only one additional module when *n*=6, a great reduction as far as the number of nano-modules required is concerned.

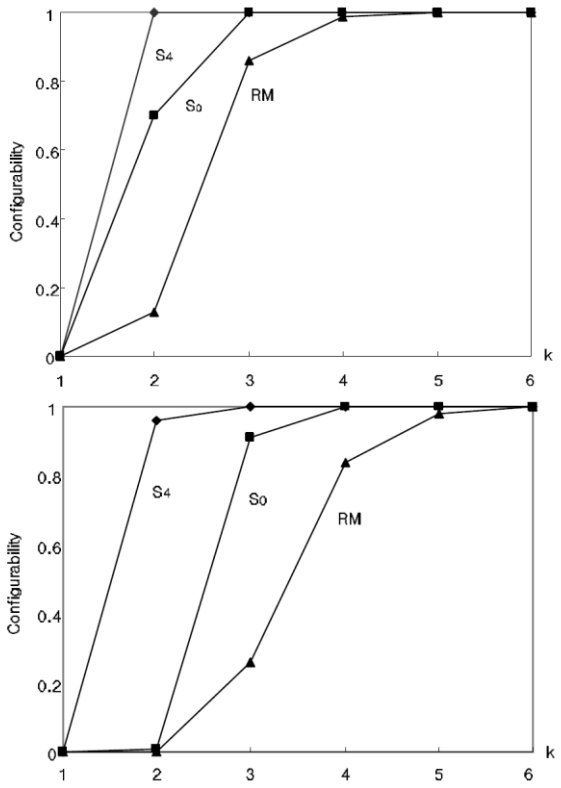

Figure 4. Configurability for p=q=0.03. (a)n=5,(b)n=6

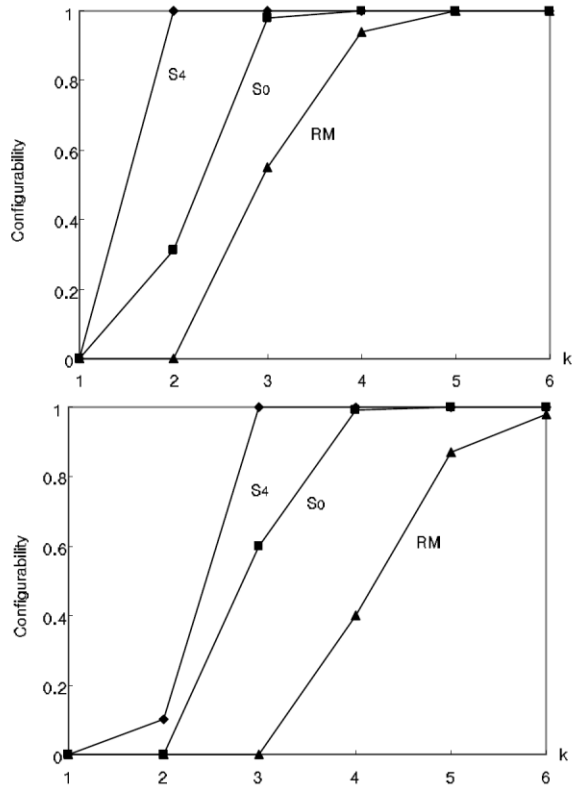

Figure 5. Configurability for p=0.05 and q=0.03 (a)n=5, (b)n=6

The overhead for configuring a memory of 2^n x2^n is estimated by the product of the number of modules and the size of each crossbar module. The overhead for the proposed scheme, A_r, can be expressed as $A_r = k_r(2^n + r)^2$, where k_r represents the number of modules (with r spare rows and r spare columns) required to achieve a near perfect configurability. The overhead for RM, A_{RM}, can be written as $A_{RM} = k_{RM}$ x2^{2n}, where k_{RM} denotes the number of nano-modules required for RM. Table 5 shows k_r (before slash) and k_{RM} for various cases under consideration. Up to 67 percent reduction in the required number of modules is made as compared to RM.

Table 5. The required k_r and k_{RM} (k_r /k_{RM}) for achieving a near perfect configurability (>0.99) for various p,q in constructing a 2^nx2^n memory (a) n=5, (b) n=6.

p	q=0.01	0.03	0.05
0.01	2(r=0)/3	2(r=1)/3	2(r=1)/4
0.03	2(r=1)/4	2(r=2)/5	2(r=3)/5
0.05	2(r=1)/5	2(r=3)/6	2(r=4)/6

p	q=0.01	0.03	0.05
0.01	2(r=1)/4	2(r=2)/4	2(r=3)/5
0.03	2(r=2)/5	3(r=2)/6	3(r=4)/6
0.05	3(r=1)/6	3(r=4)/7	4(r=4)/8

The ratio R_r of the overheads can be written as

$$R_r = \frac{A_r}{A_{RM}} = \frac{k_r(2^n + r)^2}{k_{RM}(2^n)^2}$$

Based on Table 5, we can obtain the ratio R_r for various values of p and q. The results are shown in Table 6, where reductions in the range of 29% to 60% are observed for the chosen memory sizes.

Table 6. The ratio R_r for various values of p and q in configuring a 2^nx2^n memory (a) n=5, (b) n=6.

p	q=0.01	0.03	0.05
0.01	0.67	0.71	0.53
0.03	0.53	0.45	0.48
0.05	0.43	0.40	0.42

p	q=0.01	0.03	0.05
0.01	0.52	0.53	0.44
0.03	0.44	0.53	0.56
0.05	0.52	0.48	0.56

Although constructing a molecular-based memory for particular ranges of crossbar sizes and defect rates is considered in this paper, we have obtained some useful guidelines for overcoming defects in crossbar memories with

defect maps. An even higher level of redundancy can also be used to increase the total memory size.

CONCLUSION

In this paper, we have presented a two-level redundancy scheme for coping with defects in molecular-based crossbar memories. Extra rows and columns are used to reduce the effective defect rate. Spare modules with address remapping are employed to further enhance the memory configurability while reducing the crossbar overhead. Defects affecting an entire row or column are also taken into account to obtain the performance under a more realistic defect model. The proposed crossbar memory also has some potential to tolerate faults occurring during normal operation.

ACKNOWLEDGMENTS

The author would like to thank Myeong-Hyeon Lee for his support in performing simulation. This work was supported by 2014 Hongik University Research Fund.

REFERENCES

[1] J.R. Heath and M.A. Ratner, "Molecular electronics," Physics Today, May 2003, pp.43-49

[2] C.P. Collier et al., "Electronically configurable molecular-based logic gates," Science, 285, 1999, pp. 391-394.

[3] P.J. Kuekes and R.S. Williams, "Defect-tolerant molecular electronics," IEEE Int. Symp. Circuits and Systems, Phoenix-Scotdale, AZ, May 2002, pp. II-42-44, vol 2.

[4] Y. Chen et al., "Nanoscale molecular-switch crossbar circuits," Nanotechnology, 14, 2003, pp. 462-468.

[5] I. Vourkas, D. Georgios, C. Sirakoulis, and S. Hamdioui, "Alternative architectures toward reliable memristive crossbar memories," IEEE Trans. VLSI Systems, Vol. 24, No. 1, Jan 2016, pp. 206-217.

[6] C. Huang, "Robust computing with nano-scale devices: progresses and challenges, Springer, Lecture notes in Electrical Engineering, 58, 2010.

[7] M-H. Lee, Y.K. Kim and Y.-H. Choi, "A defect-tolerant memory architecture for molecular electronics," IEEE Trans. Nanotechnology, 2004, pp. 152-157.

[8] G. Wang, W. Gong and R. Kastner, "On the use of Bloom filters for defect maps in nanocomputing," IEEE ICCAD, Nov. 2006, pp. 743-746.

[9] C.M. Jeffery and J.O. Figueiredo, "Hierarchical fault tolerance for nanoscale memories," IEEE Trans. Nanotechnology, Vol. 5, No. 4, July 2006, pp. 407-414.

[10] F. Sun and T. Zhang, "Defect and transient fault-tolerant design for hybrid CMOS/nanodevice digital memories," IEEE Trans. Nanotechnology, Vol.6, No. 3, May 2007, pp. 341-351.

[11] M. Tehranipoor and R.M.P. Rad, "Defect tolerance for nanoscale crossbar-based devices," IEEE Design & Test of Computers, 2008, pp. 549-559.

[12] A. Coker, V. Taylor, D. Bhaduri, S. Shukla, A. Raychowdhury and K. Roy, " Multijunction fault-tolerance architecture for nanoscale crossbar memories," IEEE Trans. Nanotechnology, Vol. 7, No. 2, March 2008, pp. 202-208.

[13] S. Wang, J. Dai, and L. Wang, "Hybrid redundancy for defect tolerance in molecular crossbar memory," ACM J. Emerging Tech. in Computing Systems, Vol. 9, No. 1, Article 7, 2013.

[14] M. Kule, H. Rahaman, B.B. Bhattacharya, "On finding a defect-free component in nanoscale crossbar circuits," Procedia Computer Science 70, Elsevier, 2015, pp. 421-427.

[15] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors," Communications of the ACM 13:7, 1970, pp. 422-426.

[16] M.M. Ziegler and M.R. Stan, "CMOS/Nano codesign for crossbar-based molecular electronic systems," IEEE Trans. Nanotechnology, Vol 2, No. 4, Dec. 2003, pp. 217-230.