Run-Time Data-Dependent Defect Tolerance for Hybrid CMOS/Nanodevice Digital Memories

Fei Sun, Lu Feng, and Tong Zhang
ECSE Department, Rensselaer Polytechnic Institute, Troy, NY

***Abstract*— This paper presents a data-dependent defect tolerance design approach to improve the storage capacity of defect-prone hybrid CMOS/nanodevice digital memories. The basic idea is to reduce the memory redundancy overhead by exploiting the run-time matching between the data to be stored and the memory defects. A conditional bit-flipping technique is used to enable the practical realization of this design approach in presence of the conflict between the dynamic nature of run-time data-defect matching and static nature of memory system design. Computer simulation results demonstrate that the proposed method can achieve much higher storage capacity compared with conventional data-independent defect tolerance at small memory operation overhead.**

I. Introduction

As CMOS technology scales toward the end of roadmap, hybrid CMOS/nanodevice design paradigm [1]–[7] emerges as one of the most viable paths to continue the life of Moore's law. It is almost evident that, compared with current CMOS technology, any emerging nanodevices will have worse reliability characteristics (such as the probabilities of permanent defects and transient faults) [8]–[10]. Because the uniform functionality and regular structure of digital memories make it relatively easy to tolerate a high degree of device unreliability, nanoelectronic digital memories have attracted much attention and experienced significant recent advances, e.g., see [11]–[17]. This work concerns the fault tolerance system design for hybrid nanoelectronic digital memories.

Fault tolerance for hybrid nanoelectronic digital memories has been recently addressed in [18]–[21]. In spite of different specific techniques, all the prior work share two features, including (1) defect tolerance involves error correcting codes (ECC) together with certain redundancy repair/reconfiguration scheme, and (2) defects are treated in a *data-independent* manner, i.e., all the defective memory cells are always considered as errors that must be corrected, irrelevant to the data being stored into these cells during runtime. As assumed in prior work, nanodevice memory cell defects mainly include open defect (i.e., absence of the nano-memory cell at one crosspoint) and short defect (i.e., a short at one crosspoint). Since a memory cell short defect can be considered as nanowire short defect and the associated two orthogonal nanowires will be eliminated from the memory address space, ECC is used to handle only open defects. Because an open defect always delivers the same output, say logic 1, during the read operation, an error occurs *only if* the bit written to this cell is a logic 0. This suggests that the data-independent defect tolerance assumed in all the prior work essentially results in run-time ECC over-protection, leading to more-than-necessary ECC coding redundancy overhead during runtime.

The above intuition motivates us to investigate the potential of using run-time data-dependent defect tolerance to reduce the ECC coding redundancy and hence improve the effective memory storage capacity. As elaborated later, although the basic idea is very intuitive, its practical realization may not be trivial due to the conflict between the dynamic nature of defect-data matching during runtime and static nature of memory design. In this work, we propose a conditional bit-flipping data storage concept inspired by a well-known technique for low power data bus design [22] in order to accommodate such design conflict. As discussed in Section III, its practical realization incurs a design trade-off between storage capacity and memory operation overhead. This paper presents a specific approach to realize such data-dependent defect tolerance through run-time conditional bit-flipping, which allows flexible configurations of the storage capacity vs. operation overhead trade-off. Using BCH (Bose-Chaudhuri-Hocquenghem) codes as the ECC and following the estimations of nanodevice memory power consumption and access time presented in [19], this paper demonstrates the effectiveness of the proposed design solution with computer simulations. The simulation results show that the proposed design approach may largely improve the storage capacity while keeping the power and access time overhead very small.

II. Background and Key Assumptions

Fig. 1 illustrates the principal structure of hybrid nanoelectronic memories considered in this work. The bulk of data storage is realized by an array of highly dense nanoscale crossbars in which each crosspoint holds one two-terminal nanoscale switching device. The CMOS circuits perform testing, fault tolerance, I/O, and some other critical functions. This work assumes the following fault model in nanodevice

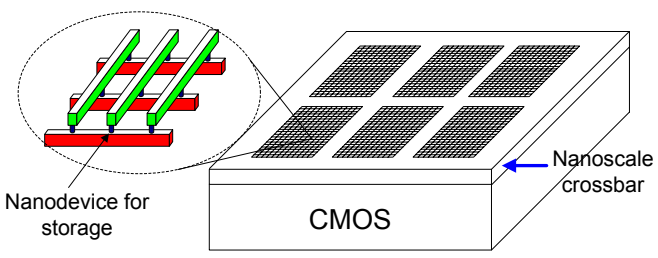

Fig. 1. Principal structure of hybrid nanoelectronic memories.

memory array: In terms of defects, we only consider static defects of nanowires and nanodevice memory cells. We assume a defective nanowire (irrelevant to defect type) will make all the connected nanodevice memory cells unfunctional. A memory cell may be subject to open or short defects. Since a memory cell short defect will short two orthogonal nanowires, it can be considered as nanowire defect. An open defect does not affect the operation of any other memory cells and any nanowires. We assume these static defects are random and statistically independent, which are characterized by two defect probabilities, including (1) bit defect probability p_{bit} that represents the probability of memory cell open defects, and (2) nanowire defect probability p_{wire} that represents the probability of nanowire defects. In a broad sense, transient faults refer to all the memory operational errors that are not induced by the above static defects. We also assume that transient faults are random and statistically independent, which is characterized by a transient fault rate p_{tf}.

As pointed out in [21], due to the high defect density and its possible spatial variations, different memory blocks may have (largely) different number of defective memory cells hence demand (largely) different error correcting capability. Therefore, other than using a single ECC code, we may use a group of ECC codes, denoted as $\mathcal{C}$, with different error correcting capability and hence different coding redundancy. All the codes in $\mathcal{C}$ should be able to share the same hardware ECC encoder and decoder. Given target memory block error rate E_{target}, transient fault rate p_{tf}, and an ECC code with codeword length of l and error correction capability of t, we can calculate the required transient fault correcting capability, i.e., find the minimum value of t_{tr} that satisfies

$$\sum_{i=t_{tr}+1}^{l} \binom{l}{i} p_{tf}^{i}(1-p_{tf})^{l-i} \leq E_{target}. \quad (1)$$

This means that, out of the total t-error-correcting capability of the ECC code, t_{tr} is reserved to compensate the possible transient errors and the remaining $t - t_{tr}$ is available for correcting the errors induced by defects.

III. Proposed Data-Dependent Fault Tolerance Design Approach

As pointed out in Section I, conventional data-independent defect tolerance tends to result in a significant ECC over-protection since open defects may not necessarily produce read errors. By leveraging the run-time data-defect matching, data-dependent defect tolerance intends to only compensate those open defects that do not match the stored data during runtime. Fig. 2 shows a motivating example: A 12-bit memory block contains four open defects marked by "X", and the data as shown in Fig. 2 has been written into this memory block. Without loss of generality, we assume an open defect always produce a logic 1 during read operation. As shown in Fig. 2, only two bit errors occur when this data block is being read even though the memory block has four open defects. Intuitively, it is desirable to leverage such run-time data-defect matching to reduce the ECC coding redundancy. However, its practical realization may not be trivial, which is explained as follows. Clearly, memory design (i.e., to determine the physical location and length of each memory block and the associated ECC code) is static and each physical memory block must be able to handle the worst-case scenario. Straightforwardly, the worst-case run-time data-defect matching will make all the open defects produce read errors and hence demand the same ECC coding redundancy as that of conventional data-independent defect tolerance. Therefore, the conflict between the dynamic nature of run-time data-defect matching and static nature of memory design makes it non-trivial to realize data-dependent defect tolerance in practice.

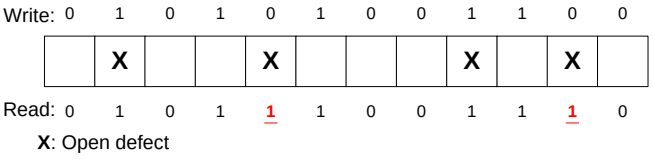

Fig. 2. An example shows run-time data-defect matching.

To tackle this challenge, we propose a method based on conditional bit-flipping. The key is to modify the memory write/read procedure so that the worst-case run-time data-defect matching demands much less ECC coding redundancy than that of conventional data-independent defect tolerance. Fig. 3 illustrates the corresponding run-time memory write and read procedure, which is further explained as follows. Consider a memory block that uses a t-error-correcting ECC code. Since the ECC code is used to compensate both defect and transient faults, let t_{def} and t_{tr} (note that $t = t_{def} + t_{tr}$) denote the error correcting capability for compensating read errors induced by defects and transient faults, respectively. Let d denote the number of open defects in this memory block, we set $t_{def} \geq \lfloor \frac{d}{2} \rfloor$, i.e., the storage reliability can be ensured up to when half of the defects produce read errors. Therefore, given any data block to be stored, either itself or its bit-wise flip will ensure enough data-defect matching so that less than t_{def} open defects produce errors. As shown in Fig. 3, in order to write one input data block into the memory, we first write the ECC encoded data to the memory block, then immediately read it out and count the number of errors which is denoted as N_{err}. If $N_{err} > t_{def}$, i.e., the current data-defect matching is not sufficient, we simply bit-wise flip the entire data block and write it to the memory block. A 1-bit flip decision associated with each memory block is stored in the CMOS memory, which will be used in the read operation as shown in Fig. 3.

As the cost of reduced ECC coding redundancy, certain memory operation overhead is incurred, particularly during the write operation. To store each data block, we need to perform either write-read or write-read-write operation sequence. Intuitively, if we set $t_{def} = \lfloor \frac{d}{2} \rfloor$ for each memory block, the occurrence of write-read and write-read-write operations may be equally possible. Meanwhile, based on the memory operation metric estimation results presented in [19], memory

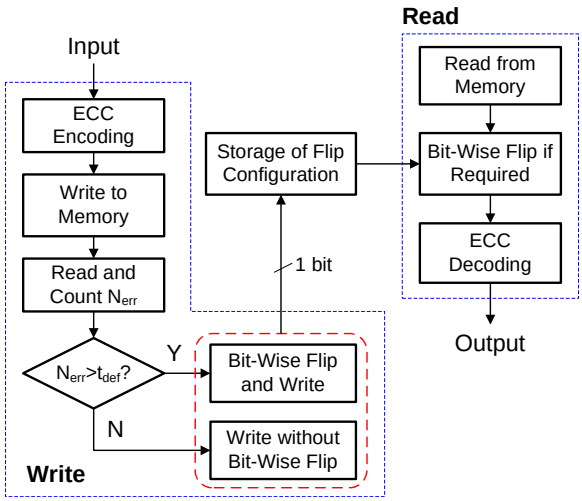

Fig. 3. The memory write and read procedure using the proposed bit-flipping method.

read can be much more energy-efficient and faster compared with memory write[1]. Hence, the overall energy and speed overhead may largely reduce if we reduce the possibility of occurrence of write-read-write operations. This can be realized by increasing the value of t_{def} for each memory block, i.e., using a stronger ECC code to reduce the possibility that the data block has to be flipped. Clearly, this will increase the ECC coding redundancy and hence reduce the effective storage capacity. Therefore, this results in a design trade-off between memory operation overhead and achievable storage capacity. Such trade-off can be easily adjusted by introducing a parameter Δ such that $t_{def} = \lfloor \frac{d}{2} \rfloor + \Delta$. In next section, we will demonstrate such design trade-offs based on computer simulations.

To realize the above proposed data-dependent defect tolerance, one remaining issue is the static memory system design, i.e., how to determine the physical location of each memory block and its associated ECC code. Using the same principle of the two-level hierarchical memory system design approach in [21], we develop the following memory design procedure. The design objective is to partition the nanodevice memory array into a certain number of consecutive memory cell segments so that each segment can store one ECC codeword with just enough coding redundancy to handle the worse-case data-defect matching. Hence each physical memory segment corresponds to one unique logical memory address. The information of each segment location and the associated ECC code configuration (i.e., which code code out of the code group is being used for present segment) are stored in CMOS memory. Let $\mathcal{C}$ denote the ECC code group, and for each code C^i, t^i_{def} denote its error correcting capability used for defect tolerance and l^i_c denote its codeword length. All the codes are ordered so that $t^i_{def} < t^j_{def}$ for $i < j$. Given this code group $\mathcal{C}$ and the design parameter Δ that is used to adjust the trade-off between storage capacity and memory operation overhead, we have the following design procedure.

[1]Notice that, if a special write mode is used other than typical random write, the speed of write may be higher than read. Readers are referred to [19] for a detailed discussion.

Proposed Procedure for Memory System Design

After excluding all the defective nanowires from the nanodevice memory physical address space, we initialize two memory cell pointers, *Ptr_Head* and *Ptr_Tail*, that point to the first memory cell, and start the following iterative process to locate each memory segment and determine the associated ECC code. This iterative process will terminate when either pointer reaches the end of the memory cell array.

Step 1: Initialize the ECC code index counter i as 0;

Step 2: Move *Ptr_Tail* forward over the next $(l^i_c - l^{i-1}_c)$ memory cells, where we set $l^{-1}_c = 0$.

Step 3: Count the number of defective memory cells, denoted as N_{def}, between *Ptr_Head* and *Ptr_Tail*.

Step 4: If $t^i_{def} \geq \lfloor \frac{N_{def}}{2} \rfloor + \Delta$, i.e., the currently selected ECC code can support the realization data-dependent defect tolerance and ensure the target block error rate, then one segment has been successfully located. We store the physical address of *Ptr_Head* and the designation of the currently selected BCH code into CMOS memory, set *Ptr_Head* = *Ptr_Tail* + 1, and go to Step 1.

Step 5: If $t^i_{def} < \lfloor \frac{N_{def}}{2} \rfloor + \Delta$, then increment the ECC code index counter by 1, i.e., $i = i + 1$. If $i < |\mathcal{C}|$, go to Step 2, otherwise go to Step 6.

Step 6 Change the location of current segment by moving *Ptr_Head* forward over the first next defective memory cell, and go to Step 1.

The implementation overhead of this proposed design technique mainly includes ECC code encoder/decoder realization and configuration data storage in CMOS domain. Furthermore, to improve the effective storage capacity, this proposed design technique sacrifices memory access speed and energy. Therefore, trade-offs among the implementation overhead, memory access penalty, and effective storage capacity must be carefully evaluated in order to obtain an appropriate overall memory system design. We will present a design example in the next section to demonstrate the effectiveness of this proposed design technique and the involved design trade-offs,

IV. Design Example

Let S_{nano} denote the average number of user bits that could be stored in each nanodevice memory cell array, and S_{CMOS} denote the average number of configuration bits stored in CMOS memory. To take into account of the storage overhead in CMOS domain, we define *net storage capacity* as $S_{net} = S_{nano} - d \cdot S_{CMOS}$, where the factor d represents the ratio between the effective cell area of a CMOS memory cell and a nanodevice memory cell. Simulations are carried out with the following configurations: each nanodevice memory cell

array is 512×512; nanowire defect probability $p_{wire} = 0.3$; target block error rate $E_{target} = 1 \times 10^{-15}$; and the factor $d = 25$. We considered three different numbers of user bits per block, including 512, 1024 and 2048. We use BCH codes as ECC and construct three BCH code groups as listed in Table I for these three scenarios, respectively, where n_{max} represents the maximum code length and t_{max} represents the maximum number of correctable errors.

TABLE I
BCH CODE GROUP CONFIGURATIONS.

	Group I	Group II	Group III
Galois Field	GF(2^{10})	GF(2^{11})	GF(2^{12})
n_{max}	1023	2047	4095
t_{max}	57	106	198

Fig. 4 shows the simulation results on the average storage capacity per 512×512 nanodevice memory cell array, including the user bits stored in nanodevice memory cells, configuration bits stored in CMOS memory, and net storage capacity. We set $\Delta = 6$. In each figure, the dotted, solid, and dashed curves correspond to the transient fault rates of 0, 1‰, and 5‰, respectively. Given the nanowire defect probability of 0.3, on average each nanodevice memory cell array contains $(1-0.3)^2 \cdot 512 \cdot 512 \approx 1.3 \times 10^5$ memory cells after excluding the defective nanowires.

To further demonstrate the storage capacity improvement by using the proposed data-dependent defect tolerance, Fig. 5 shows the net storage capacity comparison between data-dependent defect tolerance and conventional data-independent defect tolerance. We set $\Delta = 6$ for all the cases when using data-dependent defect tolerance. The transient fault rate is assumed to be 1‰. It clearly shows that the proposed data-dependent defect tolerance can significantly improve the effective data storage capacity as the defect density increases.

As pointed out in Section III, the storage capacity improvement is gained at the cost of memory operation overhead. Furthermore, the trade-off between the storage capacity and memory operation overhead is directly determined by the design parameter Δ. Fig. 6 demonstrates such a design trade-off and quantifies the memory operation overhead compared with the conventional data-independent defect tolerance. In this context, we use the estimated memory operation metrics given in [19], where the ratio of power consumption between write and read is about 2500 and the ratio of access time between write and read is about 20. By normalizing the metrics of conventional data-independent defect tolerance as 1, Fig. 6(b) and (c) show the normalized average write cycle time and write power consumption when using data-dependent defect tolerance. We note that the worst-case write cycle time is more than twice as large compared to data-independent case. These results clearly justify that the proposed data-dependent defect tolerance may achieve a significant storage capacity improvement at small memory operation overhead. For example, when $\Delta = 6$, the average write cycle time increases by less than 10% and write power consumption increases by less than 5%. Finally, it should be pointed out that the effectiveness of this technique heavily depends on the underlying fault model (this work assumes the defects and transient faults are random and statistically independent) and does not depend on the specific ECC codes being used.

V. CONCLUSIONS

A data-dependent defect tolerance design approach is presented to tackle the fault tolerance challenge in the emerging defect-prone hybrid nanoelectronic digital memories. The basic idea is to exploit the run-time data-defect matching to reduce the required ECC coding redundancy and hence improve the effective memory storage capacity. We proposed a conditional bit-flipping technique and the corresponding memory system design procedure to practically implement data-dependent defect tolerance. Computer simulations demonstrate that such data-dependent defect tolerance may greatly improve storage capacity at small memory operation overhead in terms of average write cycle time and write power consumption.

REFERENCES

[1] S. Goldstein and M. Budiu, "NanoFabrics: spatial computing using molecular electronics," in *Proc. of International Symposium on Computer Architecture*, July 2001, pp. 178–189.

[2] M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach, and M. M. Ziegler, "Molecular electronics: from devices and interconnect to circuits and architecture," *Proceedings of the IEEE*, vol. 91, pp. 1940–1957, Nov. 2003.

[3] A. DeHon, "Array-based architecture for FET-based, nanoscale electronics," *IEEE Transactions on Nanotechnology*, vol. 2, pp. 23–32, 2003.

[4] K. K. Likharev and D. B. Strukov, *CMOL: Devices, Circuits, and Architectures (in Introducing Molecular Electronics edited by G. Cuniberti et al.)* , Springer, Berlin, 2005. available at http://129.49.56.136/likharev/personal/.

[5] P. J. Kuekes, D. R. Stewart, and R. S. Williams, "The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits," *Journal of Applied Physics*, vol. 97(3):034301, 2005.

[6] A. DeHon and K. K. Likharev, "Hybrid cmos/nanoelectronic digital circuits: devices, architectures, and design automation," in *Proc. of International Conference on Computer-Aided Design (ICCAD)*, 2005, pp. 375–382.

[7] M.R. Stan, "Hybrid CMOS/molecular Electronic Circuits," in *Proc. of IEEE Conference on Emerging Technologies - Nanoelectronics*, Jan. 2006, pp. 183–188.

[8] Semiconductor Industry Association, *The International Technology Roadmap for Semiconductors (ITRS)*, http://www.itrs.net/reports.html.

[9] *Silicon Nanoelectronics and Beyond: Challenges and Research Directions, version 1.1*, Aug. 2004.

[10] K. K. Likharev, *Electronics Below 10 nm (in Nano and Giga Challenges in Microelectronics edited by J. Greer et al.)*, Elsevier, 2003.

[11] X. Duan, Y. Huang, and C.M. Lieber, "Nonvolatile memory and programmable logic from molecule-gated nanowires," *Nano Lett.*, vol. 2, no. 5, pp. 487, May 2002.

[12] M. S. Fuhrer, B. M. Kim, T. Durkop, and Brintlinger, "High-mobility nanotube transistor memory," *Nano Letters*, vol. 2, pp. 755–759, 2002.

[13] Y. Chen et al., "Nanoscale molecular-switch devices fabricated by imprint lithography," *Applied Physics Letter*, vol. 82, no. 10, pp. 1610–1612, 2003.

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

[15] D. R. Stewart et al., "Molecule-independent electrical switching in pt/organic monolayer/ti devices," *Nano Letters*, vol. 4, pp. 133–136, 2004.

[16] R. J. Tseng et al., "Polyaniline nanofiber/gold nanoparticle nonvolatile memory," *Nano Letters*, vol. 5, pp. 1077–1080, 2005.

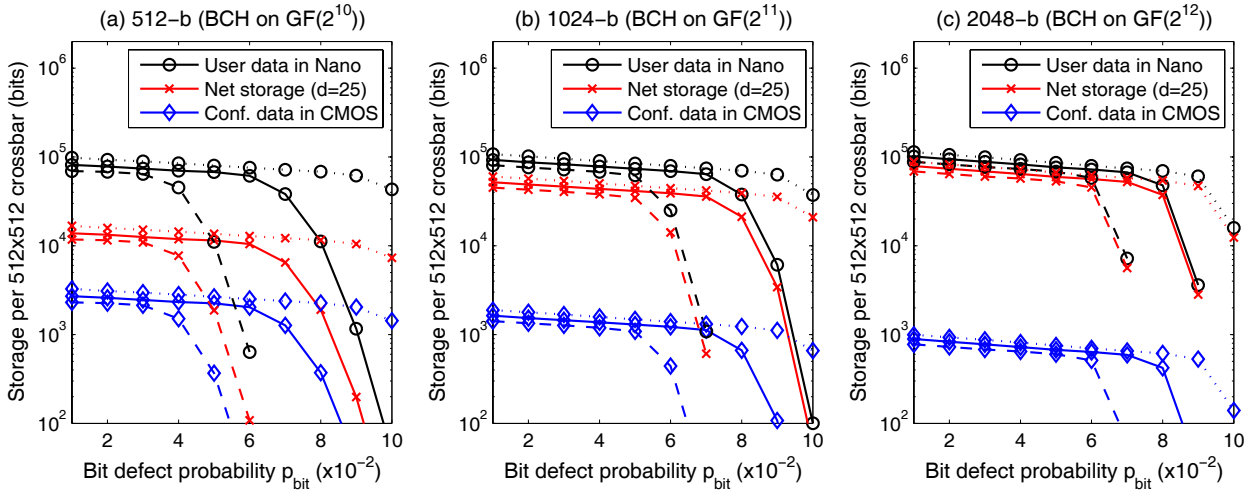

Fig. 4. Simulation results on the average storage capacity per 512×512 nanodevice memory cell array using the data-dependent fault tolerance approach ($\Delta = 6$). The dotted, solid, and dashed curves correspond to the transient fault rates of 0, 1‰, and 5‰, respectively. Under the nanowire defect probability of $p_{wire} = 0.3$, on average each 512×512 nanodevice memory cell array contains 1.3×10^5 cells after excluding the defective nanowires.

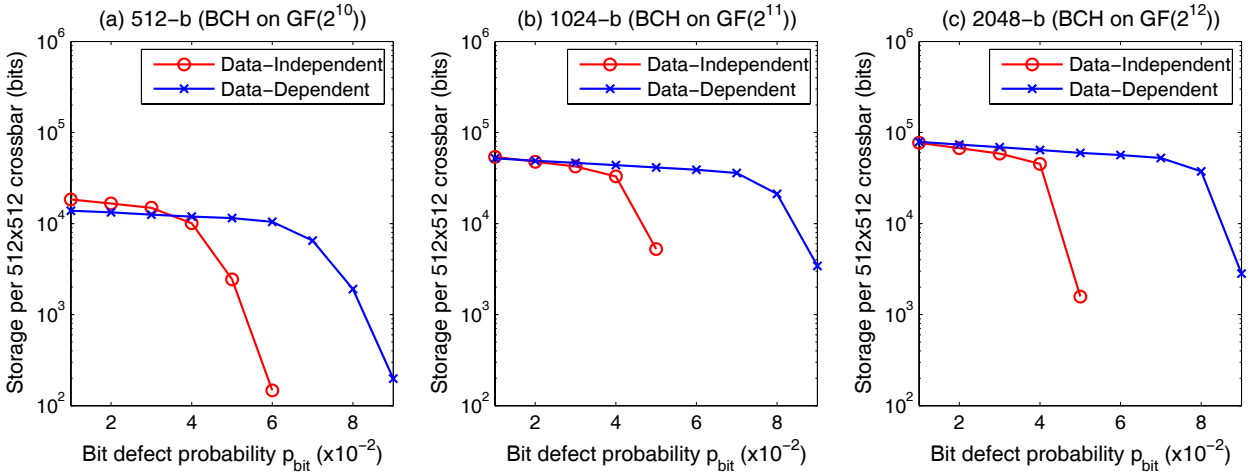

Fig. 5. The net storage capacity comparisons between data-dependent ($\Delta = 6$) and data-independent defect tolerance schemes at a transient fault rate of 1‰.

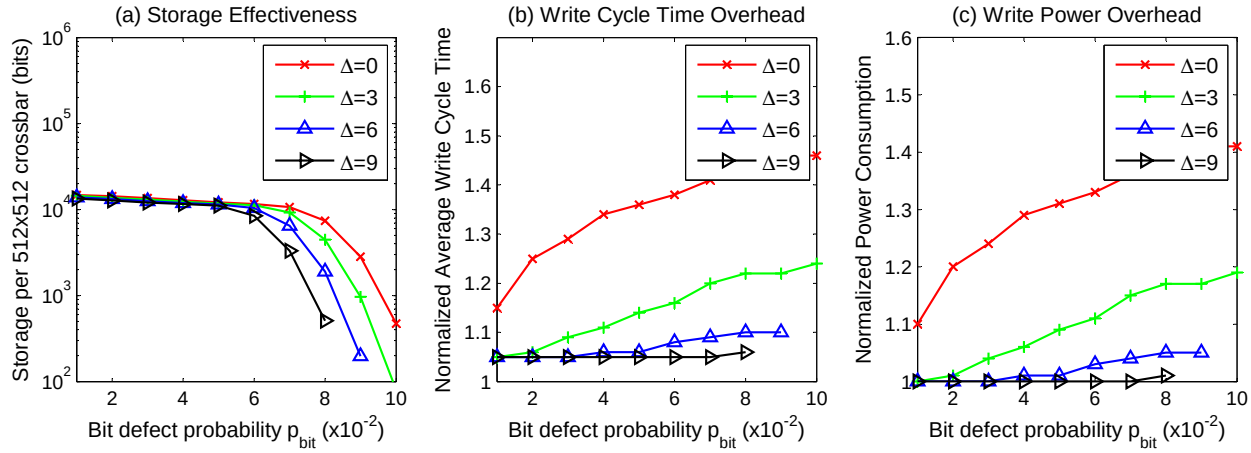

Fig. 6. Trade-offs between storage capacity and memory operation overhead with different Δ values. The user block length is 512 bits (BCH on (2^{10})) and the transient fault rate is 1‰.

[17] G. Rose et al., "Design approaches for hybrid cmos/molecular memory based on experimental device data," in *Proc. of Great Lakes Symposium on VLSI*, 2007, pp. 2–7.
[18] D. B. Strukov and K. K. Likharev, "Prospects for terabit-scale nanoelectronic memories," *Nanotechnology*, vol. 16, pp. 137–148, Jan. 2005.
[19] A. DeHon, S. C. Goldstein, P. J. Kuekes, and P. Lincoln, "Nonphotolithographic nanoscale memory density prospects," *IEEE Transactions on Nanotechnology*, vol. 4, pp. 215–228, March 2005.
[20] D. B. Strukov and K. K. Likharev, "Defect-tolerant architectures for nanoelectronic crossbar memories," *Journal of Nanoscience and Nanotechnology*, vol. 7, pp. 151–167, Jan. 2007.
[21] F. Sun and T. Zhang, "Two fault tolerance design approaches for hybrid CMOS/nanodevice digital memories," in *IEEE International Workshop on Defect and Fault Tolerant Nanoscale Architectures (Nanoarch)*, June 2006.
[22] M.R. Stan and W.P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, pp. 49–58, March 1995.