Defect and Transient Fault-Tolerant System Design for Hybrid CMOS/Nanodevice Digital Memories

Fei Sun, *Student Member, IEEE*, and Tong Zhang, *Member, IEEE*

***Abstract*—Targeting on the future fault-prone hybrid CMOS/nanodevice digital memories, this paper presents two fault-tolerance design approaches that integrally address the tolerance for defects and transient faults. These two approaches share several key features, including the use of a group of Bose–Chaudhuri–Hocquenghem (BCH) codes for both defect tolerance and transient fault tolerance, and integration of BCH code selection and dynamic logical-to-physical address mapping. The first approach is straightforward and easy to implement but suffers from a rapid drop of achievable storage capacity as defect densities and/or transient fault rates increase, while the second approach can achieve much higher storage capacity under high defect densities and/or transient fault rates at the cost of higher implementation complexity and longer memory access latency. Based on extensive computer simulations and BCH decoder circuit design, we have demonstrated the effectiveness of the presented approaches under a wide range of defect densities and transient fault rates, while taking into account of the fault-tolerance storage overhead and BCH decoder implementation cost in CMOS domain.**

***Index Terms*—Bose–Chaudhuri–Hocquenghem (BCH) codes, complementary metal–oxide–semiconductor (CMOS), defect/fault tolerance, error correcting code (ECC), hybrid digital memory, nanodevice, very large scale integration (VLSI) implementation.**

I. Introduction

The past few years experienced spectacular advances in the fabrication and manipulation of molecular and other nanoscale devices [1]–[7]. Although these new devices show significant future promise to sustain Moore's Law beyond the CMOS scaling limit, there is a growing consensus [8], [9] that, at least in the short term, they cannot completely replace CMOS technology. As a result, there is a substantial demand to explore the opportunities for CMOS and molecular/nanotechnologies to enhance and complement each other. This naturally leads to a paradigm of hybrid CMOS/nanodevice nanoelectronics [10]–[16], where an array of nanowire crossbars, with wires connected by simple nanodevices at each crosspoint, sits on the top of a CMOS circuit. The crosspoint nanodevices are responsible for the bulk of information processing and/or storage, while the CMOS circuit may perform testing and fault tolerance, global interconnect, and some other critical functions. It is almost evident that, compared with the current CMOS technology, any emerging nanodevices will have (much) worse reliability characteristics (such as the probabilities of permanent defects and transient faults). Hence, fault tolerance[1] have been well recognized as one of the biggest challenges in the emerging hybrid nanoelectronic era [9].

This work concerns the fault-tolerant system design for hybrid nanoelectronic digital memories. Conventionally, defects and transient faults in CMOS digital memories are treated separately, i.e., defects are compensated by using spare rows, columns, and/or words to repair (i.e., replace) the defective ones, while transient faults are compensated by error correcting codes (ECC) such as Hamming and Bose–Chaudhuri–Hocquenghem (BCH) codes. In order to realize satisfactory defect tolerance efficiency, the repair-only approach requires very low defect densities that can be readily met by current CMOS technologies. Nevertheless, the much higher defect densities of nanodevices make the repair-only approach not sufficient, which naturally demands extending the use of ECC for both defect tolerance and transient fault tolerance. Because of the dual role of ECC, defect tolerance and transient fault tolerance should be addressed integrally. More importantly, realization of fault tolerance in hybrid nanoelectronic memory will incur area, energy, and operational latency overhead in CMOS domain, e.g., the overhead incurred by the implementation of ECC decoder and reliable storage of certain nanodevice memory configuration information in CMOS memory. Such overhead in CMOS domain must be taken into account when investigating and evaluating hybrid nanoelectronic digital memory fault-tolerant system design solutions.

Defect tolerance in hybrid nanoelectronic digital memory have been addressed in [17]–[19]. In [17], the authors analyzed the effectiveness of integrating Hamming code with spare row/column repair for defect tolerance. The ECC-only defect tolerance has been used to estimate the hybrid nanoelectronic memory storage capacity in [18]. In [19], the authors investigated the effectiveness of Hamming and BCH codes for hybrid nanoelectronic memory defect tolerance while taking into account of the overhead in CMOS domain. Nevertheless, integration of defect tolerance and transient fault tolerance has never been addressed in prior work.

This paper presents two hybrid nanoelectronic digital memory fault-tolerant system design approaches using strong BCH codes, and evaluates the BCH coding system implementation overhead in CMOS domain based on practical IC design. We understand that, at this early stage of nanoelectronics when relatively few preliminary experimental data

Manuscript received June 11, 2006; revised November 1, 2006. This work was supported by Macro IFC. The review of this paper was arranged by Associate Editor K. Likharev.

The authors are with the Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180 USA (e-mail: sunf@rpi.edu; tzhang@ecse.rpi.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNANO.2007.893572

[1]For the purpose of brevity, we will use the term *fault tolerance* for both permanent defect tolerance and transient fault tolerance.

under laboratory environments have been ever reported, there is a large uncertainty of the defect and transient fault statistical characteristics (such as their probabilities and temporal/spatial variations) in the future real-life hybrid CMOS/nanodevice digital memories. Therefore, instead of attempting to provide a definite and complete fault-tolerant system design solution, this work mainly concerns the feasibility and effectiveness of realizing memory fault tolerance under *as-worse-as-possible scenarios*. In particular, we are interested in the fault-tolerant strategies with two features: 1) they should handle as high as possible of the defect probabilities and transient fault rates and 2) they can automatically adapt to the variations of the defect statistics in digital memories (i.e., the on-chip fault-tolerant system can automatically provide just enough defect tolerance capability for a wide range of defect densities due to possible temporal/spatial variations of the defect probabilities).[2]

The presented two design approaches integrally consider defect tolerance and transient fault tolerance and share the following two features: 1) a group of BCH codes is used for both defect tolerance and transient fault tolerance and 2) for the storage of each user data block with an unique memory logical address, its BCH encoding and mapping to the physical nanodevice memory cells are integrally determined. The first approach, referred to as two-level hierarchical fault tolerance, is relatively straightforward and easy to implement; nevertheless the achievable storage capacity quickly drops as the defect density and/or transient fault rate increase. The second approach, referred to as three-level hierarchical fault tolerance, can realize a much slower drop on the achievable storage capacity as defect density and/or transient fault rate increase, while it suffers from higher implementation complexity and longer operational latency.

To further evaluate the overhead in CMOS domain of the proposed fault-tolerant design approaches, we designed the corresponding BCH decoders using 0.13 μm CMOS standard cell libraries. The Synopsys electronic design automation (EDA) tools are used throughout the entire design hierarchy down to place and route. Based on the postlayout results at 0.13 μm CMOS technology, we projected the BCH decoder implementation metrics, including silicon area, decoding latency, and decoding energy consumption, at future 32 nm CMOS technology based on a simple scaling rule. The results show that the BCH implementation overhead in CMOS domain will not be significant even though for very strong BCH codes.

II. Binary BCH Codes and Decoder Implementation

A. Background

Because of their strong random error correction capability, binary BCH codes [20] are among the best ECC candidates for realizing fault tolerance in hybrid nanoelectronic digital memories where the faults (both defects and transient faults) are most likely random and statistically independent. Binary BCH code construction and encoding/decoding are based on binary Galois fields. A binary Galois field with degree of m is represented as GF(2^m). For any $m \geq 3$ and $t < 2^{m-1}$, there exists a primitive binary BCH code over GF(2^m), denoted as $C^m(t)$, that has the code length $n = 2^m - 1$ and information bit length $k \geq 2^m - m \cdot t$ and can correct up to (or slightly more than) t errors. For most values of t, $C^m(t+1)$ requires m more redundant bits than $C^m(t)$. A primitive t-error-correcting (n, k, t) BCH code can be shortened (i.e., eliminate a certain number, say s, of information bits) to construct a t-error-correcting $(n - s, k - s, t)$ BCH code with less information bits and code length but the same redundancy.

²Notice that we do not consider the effect of the variation of transient fault rates since it would be very difficult, if not impossible, to trace and predict the transient fault statistics on-the-fly and the worst case transient fault rates must be always used in practice.

TABLE I
BCH Code Group Configurations

	Group I	Group II	Group III	Group IV
Galois Field	GF(2^{10})	GF(2^{11})	GF(2^{12})	GF(2^{13})
n_{max}	1023	2047	4095	8191
t_{max}	57	106	198	366
r_{max}	510	1023	2038	4095

Although BCH code encoding is very simple and only involves a Galois field polynomial multiplication, BCH code decoding is much more complex and computation intensive. While different BCH code decoding algorithms may lead to (slightly) different decoding computational complexity and hardware implementation results, for a (n, k, t) binary BCH code under GF(2^m), the product of the decoder silicon area and decoding latency is approximately proportional to $n \cdot t \cdot m^2$. Moreover, a group of binary BCH codes under the same GF(2^m) can share the same hardware encoder and decoder that are designed to accommodate the maximum code length, maximum information bit length, and maximum number of correctable errors among all the codes within the group. For a detailed discussion on BCH codes and their encoding/decoding, readers are referred to [20] and [21].

B. Code Construction and Decoder Implementation

In nanodevice memory, due to the high defect probabilities and their possibly large temporal/spatial variations, different physical memory portions may have (largely) different number of defective memory cells hence demand (largely) different error correcting capability. Therefore, other than using a single BCH code, we propose to use a group of BCH codes with different error correcting capability (i.e., different coding redundancy). In order to share the same hardware encoder and decoder, all the BCH codes in the group should be constructed under the same binary Galois field.

In this work, to demonstrate and evaluate the proposed fault-tolerance design approaches, we constructed four BCH code groups as listed in Table I, where $n_{\max}$ represents the maximum code length, $t_{\max}$ represents the maximum number of correctable errors, and $r_{\max}$ represents the number of redundant bits required for correcting $t_{\max}$ errors. Each code group contains 8 BCH codes whose t_is roughly uniformly distribute between 0 and $t_{\max}$.

To evaluate the BCH decoding implementation overhead in CMOS domain, we designed one ASIC (application-specific integrated circuit) BCH decoder for each BCH code group listed

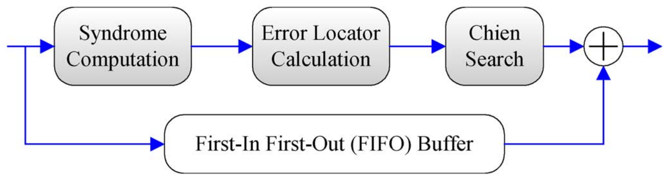

Fig. 1. Binary BCH code decoder structure.

TABLE II
BCH DECODER ASIC DESIGN POSTLAYOUT RESULTS (0.13 μM CMOS)

(n, k, t) BCH Codes	Area (mm^2)	Latency (μs)	Energy (μJ)
(1023, 513, 57)	0.85	4.1	0.4
(2047, 1024, 106)	1.83	8.2	1.7
(4095, 2057, 198)	4.34	16.4	7.5
(8191, 4096, 366)	9.15	32.8	29.9

TABLE III
ESTIMATED DECODER IMPLEMENTATION METRICS AT 32 NM TECHNOLOGY NODE

(n, k, t) BCH Codes	Area (mm^2)	Latency (μs)	Energy (μJ)
(1023, 513, 57)	0.06	0.4	0.05
(2047, 1024, 106)	0.12	0.8	0.24
(4095, 2057, 198)	0.28	1.6	1.07
(8191, 4096, 366)	0.58	3.3	4.27

above. A binary BCH code decoder consists of three computational blocks and one first-in first-out (FIFO) buffer, as shown in Fig. 1. While the implementations of syndrome computation and Chien search blocks are straightforward, the realization of error locator calculation is nontrivial and several algorithms [21] have been proposed in this regard. In this work, we use the inversion-free Berlekamp–Massey algorithm [22] to realize the error locator calculation. To minimize the decoder silicon area, the BCH decoders are fully serial, i.e., it receives 1-bit input and generates 1-bit output per clock cycle.

These four BCH decoders are designed using 0.13 μm CMOS standard cell library with 4 metal layers and a power supply of 1.2 V. Synopsys tools are used throughout the design hierarchy down to place and route. Table II shows the postlayout design results, where the decoding latency and energy consumption per codeword are obtained by assuming the codes with $n_{\max}$ and $t_{\max}$ are being used. If a BCH code with less code length and/or correctable errors is being used, the decoding latency and/or energy consumption will accordingly reduce. Furthermore, since the hybrid digital memory may become a viable option at the end-of-CMOS-roadmap, we estimate the decoder implementation metrics at the future 32 nm CMOS node, as listed in Table III, based on the projected data presented in the International Technology Roadmap for Semiconductors (ITRS) [23]: the silicon area will be scaled down by approximately 16, the logic datapath propagation delay will scale down by approximately 10, and the decoding energy consumption will scale down by approximately 7.

III. PROPOSED FAULT-TOLERANT DESIGN APPROACHES

In this work, we assume the following fault model for nanodevice memory. 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 short memory cell defect will short two orthogonal nanowires, we consider such short memory cell defects as nanowire defects. An open memory cell 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 the open memory cell defect and 2) nanowire defect probability p_{wire} that represents the probability of nanowire defect. In a broad sense, transient faults refer to all the memory operational errors that are not induced by the above static defects (e.g., the pattern-sensitive defects are considered as transient faults). We also assume that transient faults are random and statistically independent, which is characterized by a transient fault rate p_{tf}.

Let l_u represent the number of user bits per block in the memory. Given the BCH code group $\mathcal{C}$, each BCH code $C_i \in \mathcal{C}$ is shortened (if necessary) so that the codewords contain exactly l_u information bits. Let t_i represent the maximum number of errors that can be corrected by each BCH code C_i, we have $t_{\max} = \max\{t_i\}$.

Given the BCH code group and memory defect map, a fault-tolerant system should determine: 1) which BCH code should be used for protecting each l_u-bit user data block and 2) how to physically map each BCH coded data block onto the nanodevice memory cells. Intuitively, these two issues should be addressed jointly in order to obtain the best fault-tolerance efficiency. This section presents two different design approaches that address these two issues jointly, where the first approach is simple and works well under relatively low and modest bit defect probabilities and/or transient fault rates, while the second one is more complex but provide much stronger fault tolerance as bit defect probabilities and/or transient fault rates become very high.

A. Approach I: Two-Level Hierarchical Fault Tolerance

The basic idea of this design approach can be described as follows: we partition each nanodevice memory cell array into a certain number of memory cell segments; each segment contains consecutive memory cells and can store one BCH codeword that provide just enough coding redundancy to compensate all the defects in present segment and ensure a target block error rate under a given transient fault rate. Hence, each physical memory segment corresponds to one unique logical memory address. Notice that the tail of one segment is not necessarily adjacent to the head of the next segment (i.e., there might be some unused memory cells in between). The information of each segment location and the associated BCH code configuration (i.e., which BCH code out of the code group is being used for present segment) are stored in CMOS memory. Whenever we access one logical memory address in the nanodevice memory, we need first read from the CMOS memory to get the physical location and BCH coding information, then perform the corresponding operations. Therefore, we call this approach a two-level hierarchical fault-tolerance design and, in the following, we present a procedure to implement this design approach.

Two-Level Hierarchical Design Procedure

Input: the number of user bits per block l_u, BCH code group $\mathcal{C}$, nanodevice memory cell array defect map, transient fault rate $p_{\rm tf}$, and target block error rate $E_{\rm target}$.

Procedure: We first exclude all the defective nanowires from the nanodevice memory physical address space.[3] Then 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 cell segment and determine the associated BCH code. This iterative process will terminate when either pointer reaches the end of the memory cell array.

Step 1) Move Ptr_Tail forward over the next l_u memory cells. Initialize two variables $t_c = 0$ and $l = l_u$, where t_c represents the maximum number of errors that can be corrected by currently selected BCH code and l represents the length of current segment.

Step 2) Count the number of defective memory cells, denoted as $t_{\rm def}$, between Ptr_Head and Ptr_Tail. Calculate the transient fault correcting capability required to meet the target block error rate, i.e., find the minimum value of $t_{\rm trans}$ that satisfies

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

Step 3) If $t_c \geq t_{\rm def} + t_{\rm trans}$, i.e., the currently selected BCH code can provide enough coding redundancy to compensate all the defects within the present segment and achieve 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 4) If $t_c < t_{\rm def} + t_{\rm trans} \leq t_{\rm max}$ (recall that $t_{\rm max}$ is the maximum number of errors that can be corrected by any BCH codes in the code group $\mathcal{C}$), then select a BCH code from $\mathcal{C}$ that can correct $t_{\rm def} + t_{\rm trans}$ errors with the least coding redundancy. Let r represent the number of redundant bits of the selected BCH code, move Ptr_Tail forward to make $l = l_u + r$, set t_c as the maximum number of correctable errors of the currently selected BCH code, and go to Step 2.

Step 5) If $t_{\rm def} + t_{\rm trans} > t_{\rm max}$ (i.e., none of the BCH codes in $\mathcal{C}$ can correct all the defects within the present segment and ensure the target block error rate), then change the location of current segment by moving Ptr_Head forward over the first next defective memory cell, and go to Step 1.

[3]We note that how to exclude the defective nanowires from the physical address space heavily depends on the design of the interface between nanodevice memory cell array and CMOS circuits. In this work, we assume it is readily feasible and do not consider its overhead.

Suppose each nanodevice memory cell array contains N memory cells and the code group $\mathcal{C}$ contains h different BCH codes. For each segment, we need to store up to $(\lceil \log_2 N \rceil + \lceil \log_2 h \rceil)$ bits in CMOS memory, where $(\lceil \log_2 N \rceil$ bits represent the physical address of the segment head and $\lceil \log_2 h \rceil)$ bits designates which BCH code is being used for present segment. If the value of N is big (e.g., for a 512×512 nanodevice memory array, we have $N = 256$ K, hence $(\lceil \log_2 N \rceil = 18)$-bit location data have to be stored in CMOS memory for each segment), it may lead to a large storage overhead in CMOS domain. In this regard, we can modify the above procedure by setting an alignment constraint on the physical address of Ptr_Head, i.e., we require its physical address be a multiple of a constant value k (e.g., 64), which will reduce the CMOS storage overhead by $\lfloor \log_2 k \rfloor$ bits per segment.

Denote the average number of user bits stored in each nanodevice memory cell array and the average number of associated configuration bits stored in CMOS memory as $S_{\rm nano}$ and $S_{\rm CMOS}$, respectively. To take into account of the storage overhead in CMOS domain, we define the *net storage capacity* as $S_{\rm net} = S_{\rm nano} - d \cdot S_{\rm CMOS}$, where the factor d represents the ratio between the effective cell area of a CMOS memory cell and a nanodevice memory cell. To demonstrate the effectiveness of this design approach, we carried out simulations under the following configurations: each nanodevice memory cell array is 512×512; the physical address of each segment is aligned to be a multiple of 64; nanowire defect probability $p_{\rm wire} = 0.3$; target block error rate $E_{\rm target} = 1 \times 10^{-15}$; and the factor $d = 25$. We considered three different numbers of user bits per block l_u, including 512, 1024, and 2048.

Fig. 2 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 assuming $d = 25$. In each figure the solid and dashed curves correspond to the transient fault rates of $1‰$ and $5‰$, respectively. For the purpose of comparison, each figure also includes a set of dotted curves corresponding to zero transient fault rates. Given the nanowire defect probability of 0.3, on average each nanodevice memory cell array provide $(1-0.3)^2 \cdot 512 \cdot 512 \approx 1.3 \times 10^5$ memory cells after excluding the defective nanowires. Furthermore, we use Fig. 3 to highlight the performance difference when using BCH codes under different Galois fields. In each figure, the dashed curves correspond to the results of BCH codes on GF(2^{13}). Clearly, using BCH code group under larger Galois fields can tolerate a wider defect rate range due to the stronger error correcting capability, which comes with the cost of higher BCH decoder implementation complexity. Although a system designed based on this approach works well over the range of relatively low and modest bit defect probabilities and/or transient fault rates, the fault-tolerance efficiency rapidly drops as we further increase the bit defect probability and/or transient fault rate.

Besides the above comparison on fault-tolerance effectiveness, we further carried out the comparison in terms of BCH decoding latency per codeword and energy consumption per user bit. This is based on the estimated BCH decoder implementation metrics at the 32 nm CMOS technology node presented in Section II-B. Since different BCH codes within the same code group

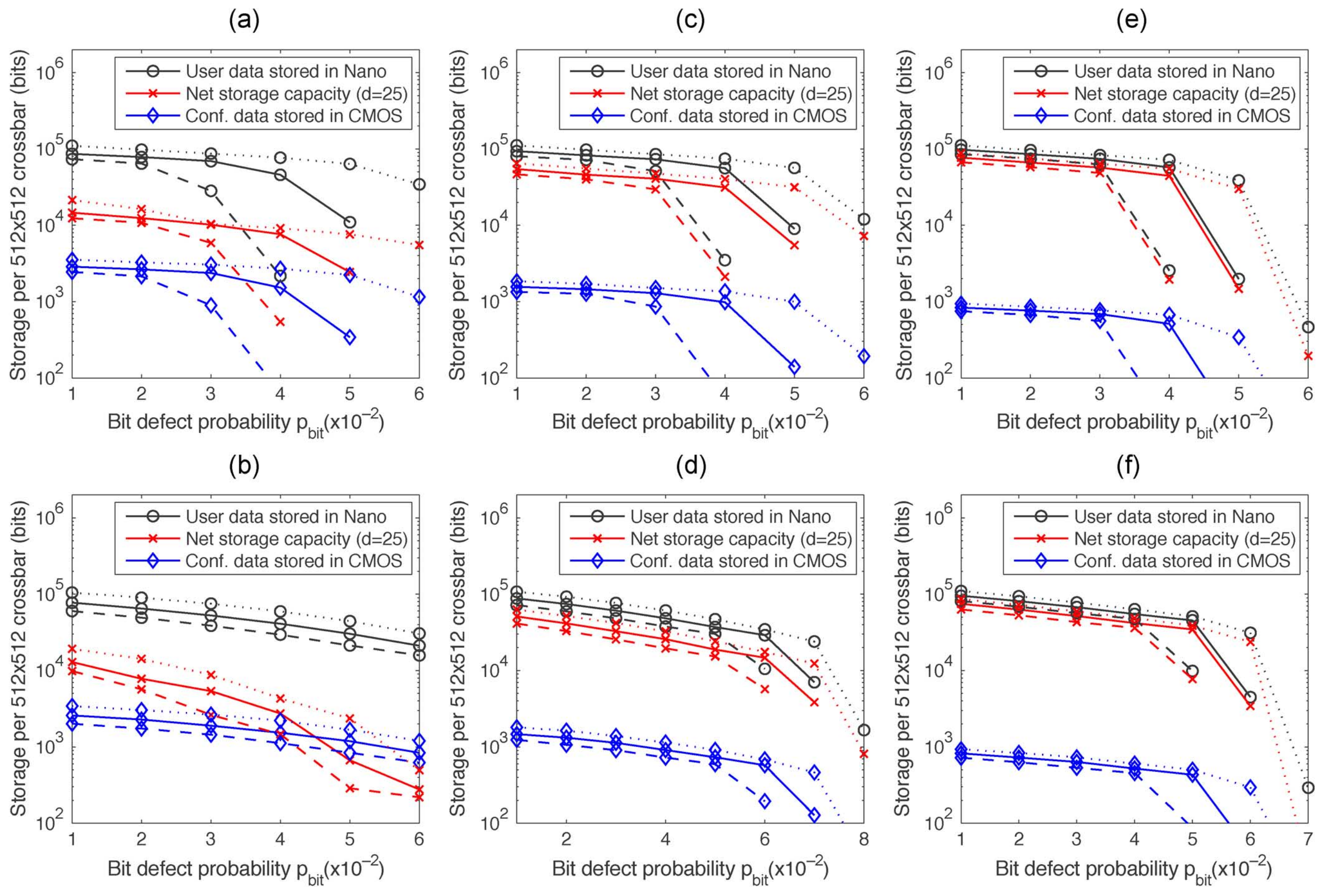

Fig. 2. Simulation results on the average storage capacity per 512 × 512 nanodevice memory cell array using the two-level hierarchical fault-tolerance approach. The dotted, solid, and dashed curves correspond to the transient fault rates of 0, 1‱, and 5‱, respectively. Under the nanowire defect probability of $p_{\mathrm{wire}} = 0.3$, on average each 512 × 512 nanodevice memory cell array contains 1.3×10^5 cells after excluding the defective nanowires. (a) 512-b (BCH on GF(2^{10})). (b) 512-b (BCH on GF(2^{13})). (c) 1024-b (BCH on GF(2^{11})). (d) 1024-b (BCH on GF(2^{13})). (e) 2048-b (BCH on GF(2^{12})). (f) 2048-b (BCH on GF(2^{13})).

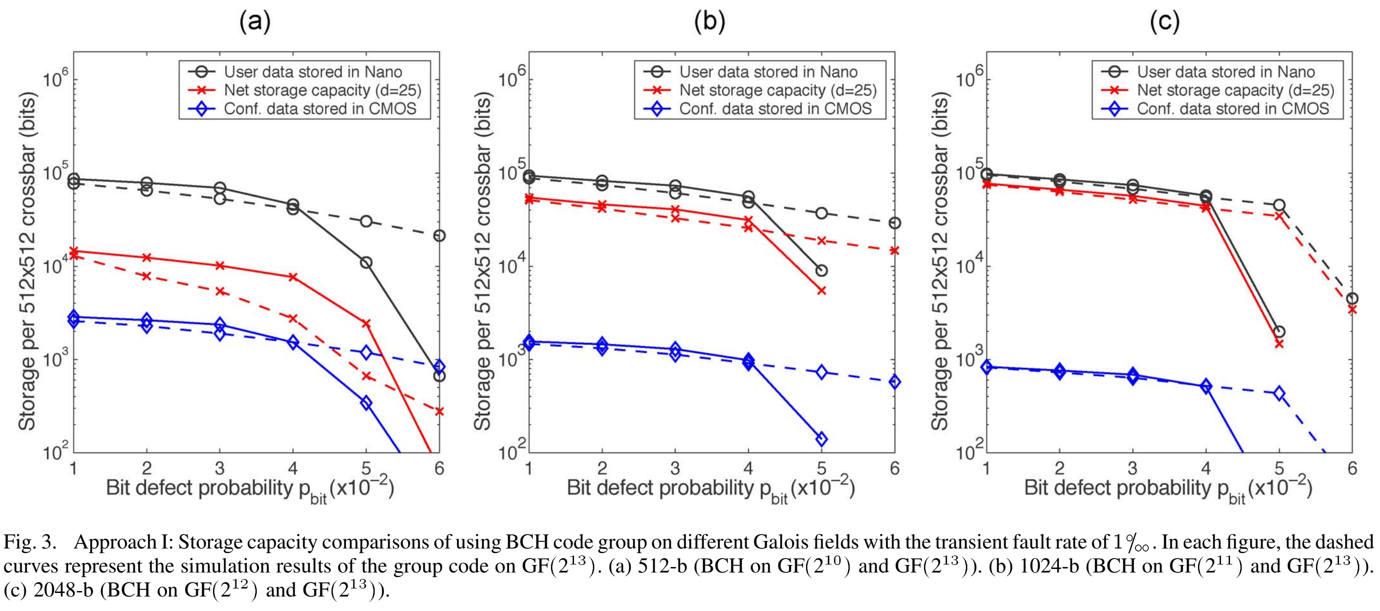

Fig. 3. Approach I: Storage capacity comparisons of using BCH code group on different Galois fields with the transient fault rate of 1‱. In each figure, the dashed curves represent the simulation results of the group code on GF(2^{13}). (a) 512-b (BCH on GF(2^{10}) and GF(2^{13})). (b) 1024-b (BCH on GF(2^{11}) and GF(2^{13})). (c) 2048-b (BCH on GF(2^{12}) and GF(2^{13})).

have different decoding energy consumption and decoding latency, we obtained the statistics on the use of different BCH codes for each scenario considered above. Since we use fully serial BCH decoders, the decoding latency is proportional to the BCH code length and the decoding energy consumption is proportional to the product of code length and the number of correctable errors. Figs. 4 and 5 show the comparison among various scenarios on the decoding energy per user bit and decoding

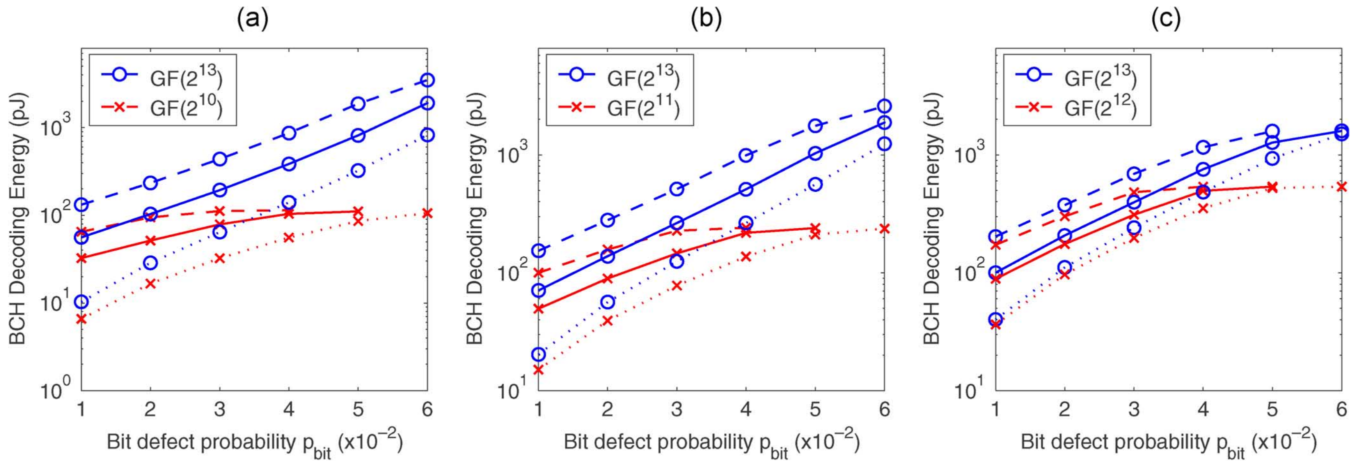

Fig. 4. Approach I: Decoding energy per user bit for the scenarios of 512-bit, 1024-bit, and 2048-bit user data per codeword. The dotted, solid, and dashed curves correspond to the transient fault rates of 0, $1‰$, and $5‰$, respectively. (a) 512-b. (b) 1024-b. (c) 2048-b.

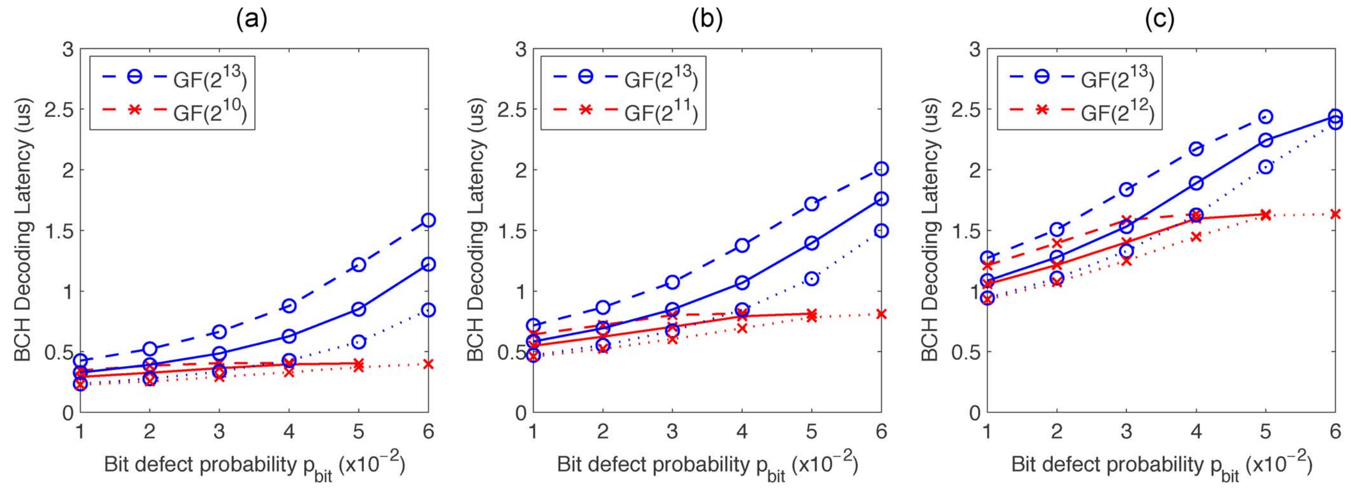

Fig. 5. Approach I: Decoding latency per codeword for the scenarios of 512-bit, 1024-bit, and 2048-bit user data per codeword. The dotted, solid, and dashed curves correspond to the transient fault rates of 0, $1‰$, and $5‰$, respectively. (a) 512-b. (b) 1024-b. (c) 2048-b.

latency per codeword, respectively. In each figure, the dotted, solid, and dashed curves correspond to the transient fault rates of 0, $1‰$ and $5‰$, respectively.

As mentioned in the above, multiple BCH codes (eight BCH codes per group in this work), which share the same encoding and decoding circuit, have been used for error correction. Although the use of multiple BCH codes may potentially improve the effective storage capacity in the nano domain, it will incur storage overhead in CMOS domain, leading to a design tradeoff. To demonstrate such tradeoff with the assumption of $d = 25$, Fig. 6 shows the comparison of using multiple BCH codes against using a single BCH code with $t_{\max}$. For l_u of 1024 and 2048, using multiple BCH codes can improve the net storage capacity at relatively small defect rates, however the advantage diminishes as the defect rate increases. This is mainly because the use of BCH code with $t_{\max}$ will tend to dominate at high defect rates, which makes the savings in the nano domain by using multiple codes reduces relatively to the storage overhead incurred in CMOS domain. For l_u of 512, using multiple codes turns out not to be a good choice due to small block length that will result in relatively higher storage overhead in CMOS domain.

B. Approach II: Three-Level Hierarchical Fault Tolerance

In the above two-level hierarchical design approach, we always attempt to locate a continuous memory cell segment to store each coded data block. Hence, with high bit defect probabilities, the total number of defective memory cells within a segment may accumulate very quickly and exceed the maximum error correcting capability. This will become more serious as the transient fault rate increases. Therefore, as shown in Fig. 2, the effectiveness of this design approach rapidly degrades as the bit defect probability and/or transient fault rate increases. In order to achieve a better storage capacity at high defect probabilities and/or transient fault rates, this section presents another approach called three-level hierarchial fault-tolerance design. The basic idea is that, other than using a continuous memory cell segment to store each coded data block, we selectively skip (or exclude) some small sectors that contain too many defective memory cells within each segment. For example, suppose we use a BCH code group on GF(2^{11}). As pointed out in Section II, for most values of t, increasing t by 1 (i.e., to compensate one more error) requires 11 more redundant bits. Hence, for a sector of 64 memory cells in which there are 6 defective memory

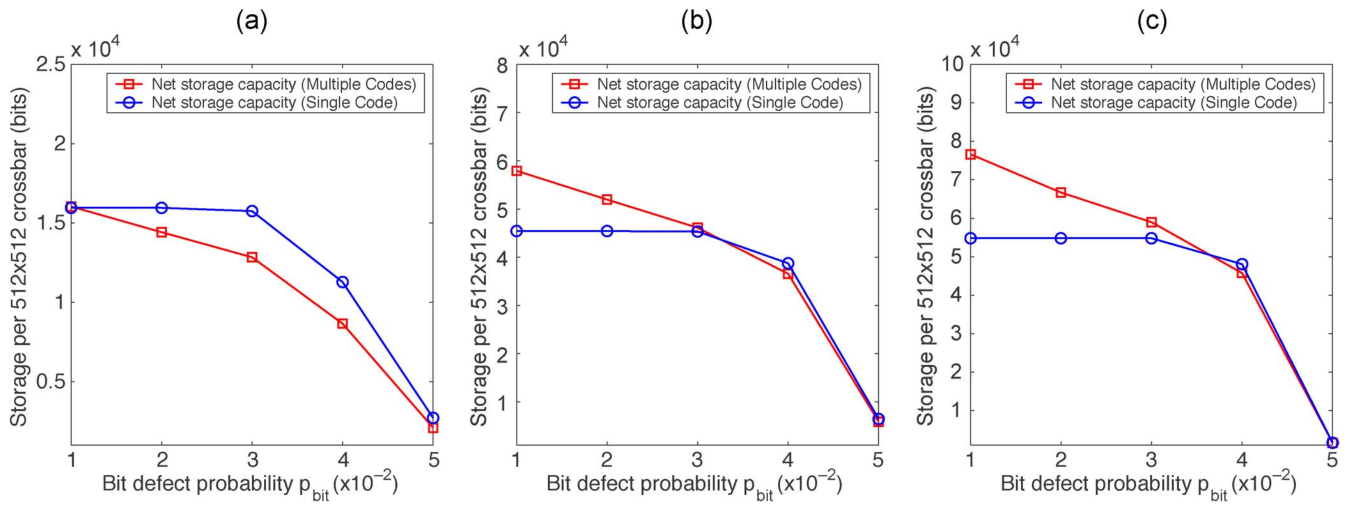

Fig. 6. Approach I: Net storage capacity comparisons between multiple-codes and single-code schemes with the transient fault rate of $1‱$. (a) 512-b (BCH on GF(2^{10})). (b) 1024-b (BCH on GF(2^{11})). (c) 2048-b (BCH on GF(2^{12})).

cells, it would be better to exclude this sector from the memory segment.

Therefore, we propose to partition the available nanodevice memory cells into a certain number of equal-sized sectors, each one is called indivisible memory unit. When we dynamically determine the BCH code selection and logical-to-physical address mapping, we have the flexibility to determine whether or not to use each indivisible memory unit for data storage. Therefore, each memory segment that stores one BCH coded data block no longer contains a consecutive region of memory cells. It is intuitively justifiable that, by selectively excluding those indivisible memory units that contain too many defective cells, we may improve the fault-tolerance efficiency. However, in support of this approach, we have to store certain configuration information, including: 1) the location and length of each memory segment; 2) the designation of the selected BCH code; and 3) whether or not each indivisible memory unit that falls into the region covered by the segment is used for data storage. If we directly store these information in CMOS memory, it will incur a significant CMOS storage overhead. For example, if the number of user bits per block is 2048 and each indivisible memory unit contains 64 consecutive memory cells, we have to store more than $32 (= 2048/64)$ bits per block for representing whether each indivisible memory unit is excluded or not.

To tackle such storage overhead issue, we propose to store these configuration information in nanodevice memory, and since the length of these configuration information will be much less than the coded user data block, we may use the above two-level hierarchical fault-tolerance approach to protect these configuration information. This leads to a so-called three-level hierarchical fault-tolerance as illustrated in Fig. 7.

In this way, we can largely reduce the storage overhead in CMOS domain. Nevertheless, as the cost, this three-level hierarchical approach requires extra operations that result in memory access energy and latency overhead: to read/write one user data block, we have to first read and decode the first level configuration data from the nanodevice memory to recover the memory segment configuration information, based on which we may read/write the intended user data block. Furthermore, this approach may require nonvolatile storage of the first level configuration data in nanodevice memory. This should not be a serious issue since most proposed/demonstrated nanodevice memory storage elements are nonvolatile in nature. In the following, we present a procedure to implement such three-level hierarchical fault-tolerance design approach.

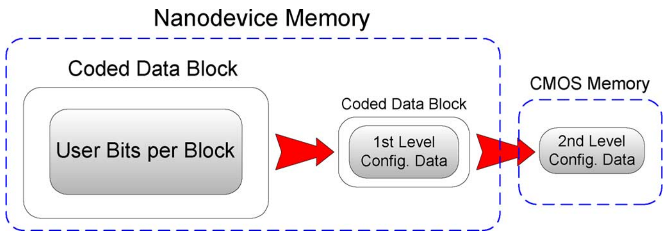

Fig. 7. Storage hierarchy in the three-level hierarchical fault-tolerance system.

Three-Level Hierarchical Design Procedure

Input: the number of user bits per block l_u, indivisible memory unit length l_c, BCH code group $\mathcal{C}$ and the degree m of the underlying Galois field GF(2^m), nanodevice memory cell array defect map, transient fault rate $p_{\rm tf}$, and target block error rate $E_{\rm target}$.

Procedure: We first exclude all the defective nanowires from the nanodevice memory physical address space. Then we partition the available nanodevice memory space into arrays of l_c-cell indivisible memory units. We mark all the indivisible memory units that contain more than $\lfloor l_c/m \rfloor$ defective memory cells as *unusable* memory units and all the others as *usable* units. The memory cells falling into usable indivisible memory units are called usable memory cells. We initialize two memory cell pointers, Ptr_Head and Ptr_Tail, that point to the first memory cell, and start the following iterative process until either pointer reaches the end of the memory cell array.

Step 1) Move Ptr_Tail forward so that there are l_u usable memory cells between Ptr_Head and Ptr_Tail. Initialize two variables $t_c = 0$ and $l = l_u$, where

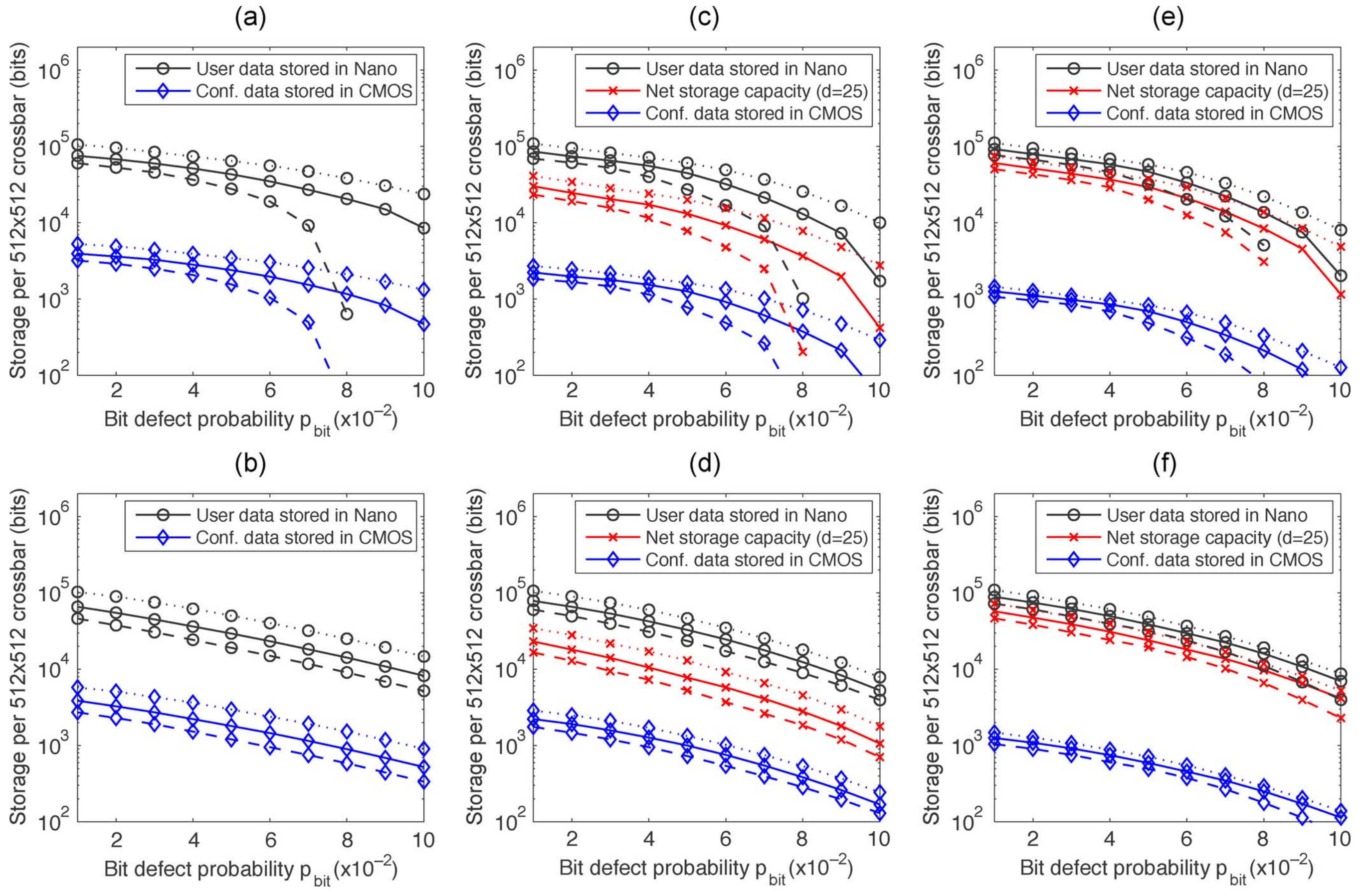

Fig. 8. Simulation results on the average storage capacity per 512 × 512 nanodevice memory cell array using the three-level hierarchical fault-tolerance approach. The dotted, solid, and dashed curves correspond to the transient fault rates of 0, $1‱$, and $5‱$, respectively. Notice that the net storage capacity is negative for $l_u = 512$ while we assume $d = 25$. Again, on average each 512 × 512 nanodevice memory cell array contains 1.3×10^5 cells after excluding the defective nanowires. (a) 512-b (BCH on GF(2^{10})). (b) 512-b (BCH on GF(2^{13})). (c) 1024-b (BCH on GF(2^{11})). (d) 1024-b (BCH on GF(2^{13})). (e) 2048-b (BCH on GF(2^{12})). (f) 2048-b (BCH on GF(2^{13})).

t_c represents the maximum number of errors that can be corrected by currently selected BCH code and l represents the number of usable memory cells within current segment.

Step 2) Count the number of defective memory cells, denoted as t_{def}, between Ptr_Head and Ptr_Tail. Calculate the transient fault correcting capability required to meet the target block error rate, i.e., find the minimum value of t_{trans} that satisfies the inequality (1) in Section III-A.

Step 3) If $t_c \geq t_{\text{def}} + t_{\text{trans}}$ (i.e., one segment has been successfully located), then go to Step 6 to process the storage of the first level configuration data in nanodevice memory.

Step 4) If $t_c < t_{\text{def}} + t_{\text{trans}} \leq t_{\max}$, then select a BCH code from $\mathcal{C}$ that can correct $t_{\text{def}} + t_{\text{trans}}$ errors with the least coding redundancy. Let r represent the number of redundant bits of the selected BCH code, move Ptr_Tail forward so that there are $l = l_u + r$ usable cells between Ptr_Head and Ptr_Tail, set t_c as the maximum number of correctable errors of the currently selected BCH code, and go to Step 2.

Step 5) If $t_{\text{def}} + t_{\text{trans}} > t_{\max}$, then move Ptr_Head forward to the next usable unit, and go to Step 1.

Step 6) Let s represent the number of indivisible memory units (both usable and unusable units) within Ptr_Head and Ptr_Tail, we need an s-bit vector to represent whether each unit is usable (i.e., included in current segment) or unusable (i.e., excluded from current segment). Hence, the first level configuration data to be stored in nanodevice memory includes an s-bit vector, the physical location and length of current segment, and the designation of the selected BCH code. Then we apply the two-level fault-tolerance approach (as described in Section III-A) to store these first level configuration data, where we can use the same BCH code group. Nevertheless, since the first level configuration data do not have a constant length, unlike the user data, we have to on-the-fly shorten those BCH codes in the code group. Hence, we need to store the information of how the selected BCH code is shortened in CMOS memory. After we encode and store the first level configuration data in a segment of successive nanodevice memory cells and store the corresponding second level configuration data in CMOS memory, we move Ptr_Head to the next available usable unit and go to Step 1.

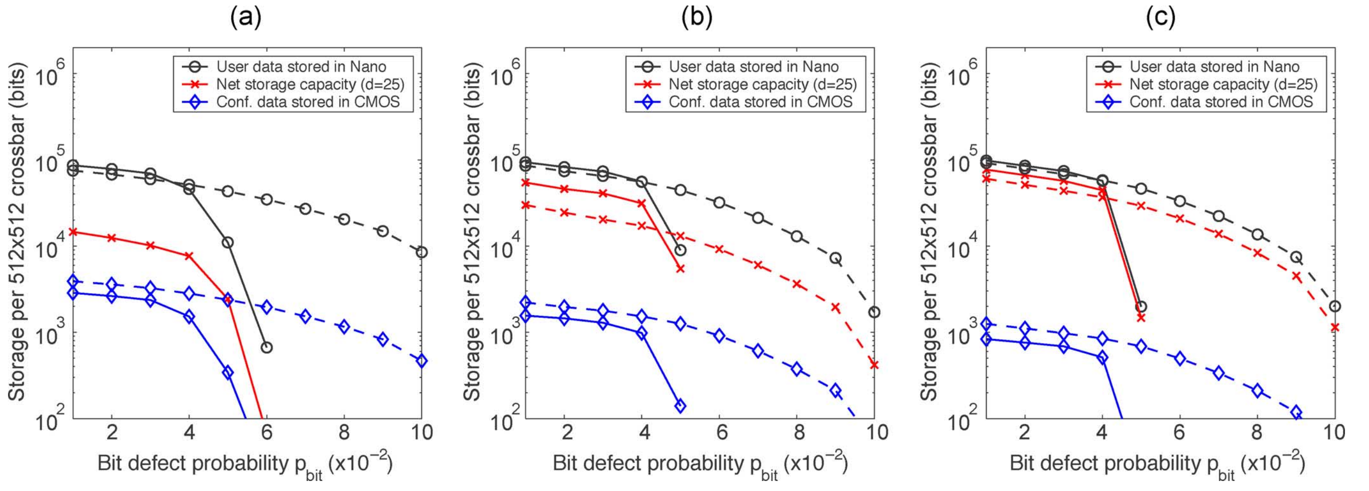

Fig. 9. Storage capacity comparisons of Approach-I and Approach-II with the transient fault rate of 1‱. In each figure the solid and dashed curves correspond to Approach-I and Approach-II, respectively. (a) 512-b (BCH on GF(2^{10})). (b) 1024-b (BCH on GF(2^{11})). (c) 2048-b (BCH on GF(2^{12})).

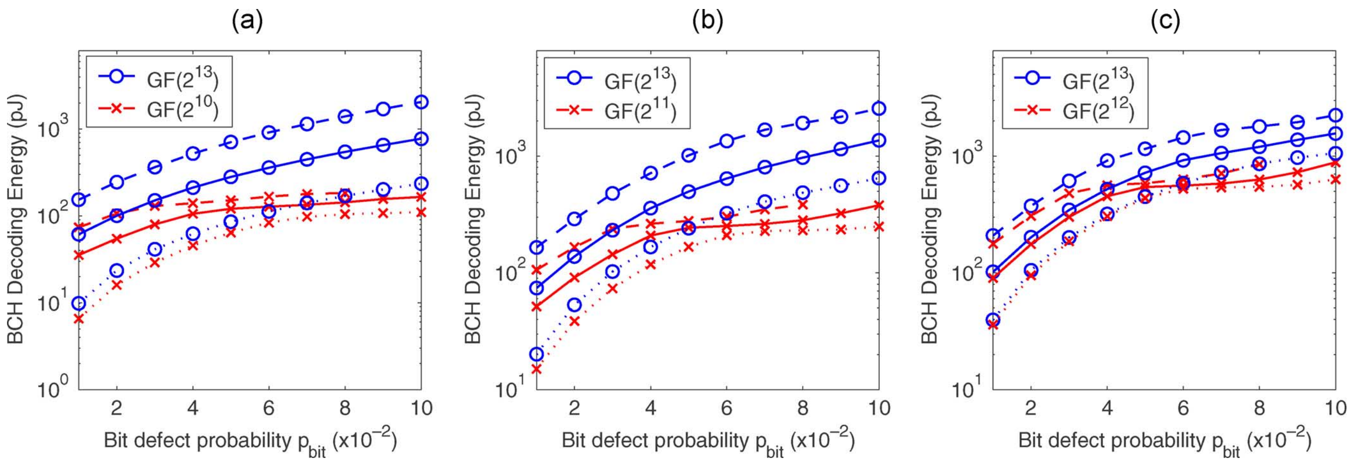

Fig. 10. Approach II: Decoding energy per user bit for the scenarios of 512-bit, 1024-bit, and 2048-bit user data per codeword. The dotted, solid, and dashed curves correspond to the transient fault rates of 0, 1‱ and 5‱, respectively. (a) 512-b. (b) 1024-b. (c) 2048-b.

To demonstrate the effectiveness of this proposed approach, we carried out simulations under the same configurations as used in Section III-A: each nanodevice memory cell array is 512 × 512; nanowire defect probability $p_{\text{wire}} = 0.3$; target block error rate $E_{\text{target}} = 1 \times 10^{-15}$; the factor $d = 25$; the same four BCH code groups are used; and the same three values of user data length l_u (i.e., 512, 1024, and 2048) are considered. We set the indivisible memory unit length l_c as 32 for $l_u = 512$ and 64 for $l_u = 1024$ and $l_u = 2048$.

Fig. 8 shows the simulation results of 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 assuming $d = 25$. In each figure the dotted, solid, and dashed curves correspond to the transient fault rates of 0, 1‱ and 5‱, respectively. We note that, for $l_u = 512$, the net storage capacity will be negative if we assume $d = 25$. Fig. 9 highlights the comparison between the above two different approaches in terms of effective storage capacity, which leads to the following observations.

- At relatively low and modest bit defect probabilities and/or transient fault rates, the two-level design approach can realize slightly better storage capacity meanwhile have less operational complexity and latency overhead.
- At relatively high bit defect probabilities and/or transient fault rates, the three-level hierarchical approach can achieve much better storage capacities.
- The three-level hierarchial approach can maintain more graceful (or smooth) storage capacity curves over wider ranges of defect probability and hence can better adapt to the potential defect statistics variations.

We also carried out the comparisons in terms of BCH decoding energy consumption and latency for the three-level hierarchical fault-tolerance approach. In this context, two BCH decodings (to decode the first and second level configuration data, respectively) should be performed in order to access one user data block. Figs. 10 and 11 show the comparisons among different scenarios on the decoding energy per user bit and decoding latency per codeword, respectively.

IV. Conclusion

In this paper, we presented two fault-tolerance design approaches that integrally address the defect tolerance and

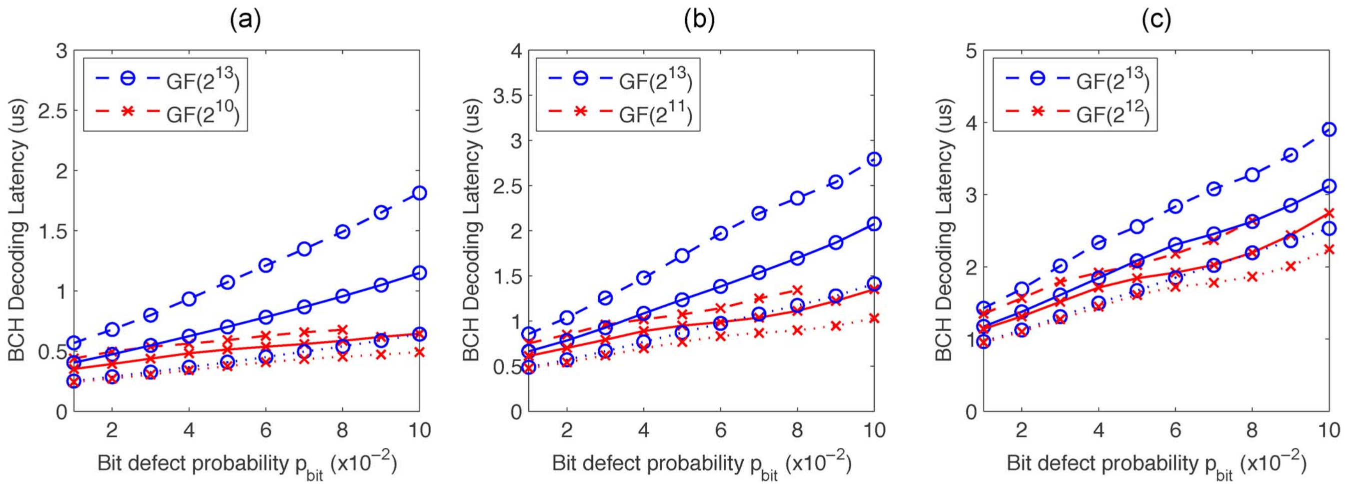

Fig. 11. Approach II: Decoding latency per codeword for the scenarios of 512-bit, 1024-bit, and 2048-bit user data per codeword. The dotted, solid, and dashed curves correspond to the transient fault rates of 0, $1‰$, and $5‰$, respectively. (a) 512-b. (b) 1024-b. (c) 2048-b.

transient fault tolerance for hybrid CMOS/nanodevice digital memories. To accommodate the high defect probabilities and transient fault rates, the developed approaches have several key features that have not been used in conventional digital memories, including the use of a group of BCH codes for both defect tolerance and transient fault tolerance, and integration of BCH code selection and dynamic logical-to-physical address mapping. These two fault-tolerance design approaches seek different tradeoffs among the achievable storage capacity, robustness to defect statistics variations, implementation complexity, and operational latency and CMOS storage overhead. Simulation results demonstrated that the developed approaches can achieve good storage capacity, while taking into account of the storage overhead in CMOS domain, under high defect probabilities (above 1%) and transient fault rates (up to $5‰$), and can readily adapt to large defect statistics variations. To evaluate the BCH code coding system implementation overhead, we designed the corresponding BCH decoders at 0.13 μm CMOS technology node. Based on the postlayout results, we projected the BCH decoder implementation metrics including silicon area, decoding latency, and energy consumption, at future 32 nm CMOS technology. The results show that the BCH implementation overhead in CMOS domain will not be significant even though for very strong BCH codes.

References

[1] 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, Apr. 2003.

[2] Z. Yu, W. Wu, G. Y. Jung, D. L. Olynick, J. Straznicky, X. Li, Z. Li, W. M. Tong, J. A. Liddle, S. Y. Wang, and R. S. Williams, "Fabrication of 30 nm pitch imprint moulds by frequency doubling for nanowire arrays," *Nanotechnology*, vol. 17, pp. 4956–4961, Oct. 2006.

[3] M. A. Reed, "Molecular-scale electronics," *Proc. IEEE*, vol. 87, no. 4, pp. 652–658, Apr. 1999.

[4] T. Rueckes *et al.*, "Carbon nanotube-based nonvolatile random access memory for molecular computing," *Science*, vol. 289, pp. 94–97, 2000.

[5] G. M. Whitesides and B. Grzybowski, "Self-assembly at all scales," *Science*, vol. 295, pp. 2418–2421, 2002.

[6] N. A. Melosh *et al.*, "Ultra high-density nanowire lattices and circuits," *Science*, vol. 300, pp. 112–115, 2003.

[7] M. A. Reed, "Molecular electronics: Back under control," *Nature Mater.*, vol. 3, pp. 286–287, May 2004.

[8] Semiconductor Industry Association, The International Technology Roadmap for Semiconductors (ITRS) [Online]. Available: http://public.itrs.net/Files/2003ITRS/Home2003.htm 2003

[9] Silicon nanoelectronics and beyond: Challenges and research directions ver. 1.1, Aug. 2004.

[10] S. Goldstein and M. Budiu, "NanoFabrics: Spatial computing using molecular electronics," in *Proc. Int. Symp. Computer Architecture*, Jul. 2001, pp. 178–189.

[11] 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," *Proc. IEEE*, vol. 91, no. 11, pp. 1940–1957, Nov. 2003.

[12] A. DeHon, "Array-based architecture for FET-based, nanoscale electronics," *IEEE Trans. Nanotechnol.*, vol. 2, no. 1, pp. 23–32, Mar. 2003.

[13] M. M. Ziegler and M. R. Stan, "CMOS/nano co-design for crossbar-based molecular electronic systems," *IEEE Trans. Nanotechnol.*, vol. 2, no. 4, pp. 217–230, Dec. 2003.

[14] K. K. Likharev and D. B. Strukov, "CMOL: Devices, circuits, and architectures," in *Introducing Molecular Electronics*, G. Cuniberti, Ed. *et al.* Berlin, Germany: Springer, 2005 [Online]. Available: http://129.49.56.136/likharev/personal/

[15] P. J. Kuekes, D. R. Stewart, and R. S. Williams, "The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits," *J. Appl. Phys.*, vol. 97, no. 3, p. 034 301, 2005.

[16] W. Wu *et al.*, "One-kilobit cross-bar molecular memory circuits at 30-nm half-pitch fabricated by nanoimprint lithography," *Appl. Phys. A*, vol. 80, pp. 1173–1178, 2005.

[17] D. B. Strukov and K. K. Likharev, "Prospects for terabit-scale nanoelectronic memories," *Nanotechnology*, vol. 16, pp. 137–148, Jan. 2005.

[18] A. DeHon, S. C. Goldstein, P. J. Kuekes, and P. Lincoln, "Nonphotolithographic nanoscale memory density prospects," *IEEE Trans. Nanotechnol.*, vol. 4, no. 2, pp. 215–228, Mar. 2005.

[19] D. B. Strukov and K. K. Likharev, "Defect-tolerant architectures for nanoelectronic crossbar memories," *J. Nanosci. Nanotechnol.*, vol. 7, no. 1, pp. 151–167, Jan. 2007.

[20] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2004.

[21] R. E. Blahut, *Algebraic Codes for Data Transmission*. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[22] H. O. Burton, "Inversionless decoding of binary BCH codes," *IEEE Trans. Inf. Theory*, vol. IT-17, no. 4, pp. 464–466, Jul. 1971.

[23] Semiconductor Industry Association, The International Technology Roadmap for Semiconductors (ITRS) [Online]. Available: http://www.itrs.net/Common/2005ITRS/Home2005.htm 2005

Fei Sun (S'06) received the B.S. and M.S. degrees in electrical engineering from Xian Jiaotong University, China, in 2000 and 2003, respectively. He has been working toward the Ph.D. degree in the electrical, computer and systems engineering department at Rensselaer Polytechnic Institute, Troy, NY, since 2003.

His research interests include VLSI architectures for communication and storage systems, and fault-tolerant system design for semiconductor memory. Currently he is working on power efficient high throughput trellis detector architecture design for read channels.

Tong Zhang (S'98–M'02) received the B.S. and M.S. degrees in electrical engineering from the Xian Jiaotong University, Xian, China, in 1995 and 1998, respectively. He earned Ph.D. in electrical engineering at the University of Minnesota in 2002. Currently he is an assistant professor in electrical, computer and systems engineering department at Rensselaer Polytechnic Institute. His current research interests include algorithm and architecture codesign for communication and data storage systems, variation-tolerant signal processing IC design, fault-tolerant system design for digital memory, and interconnect system design for hybrid CMOS/nanodevice electronic systems.