Two Fault Tolerance Design Approaches for Hybrid CMOS/Nanodevice Digital Memories

Fei Sun and Tong Zhang
ECSE Department, Rensselaer Polytechnic Institute, Troy, NY
Email: sunf@rpi.edu, tzhang@ecse.rpi.edu

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. 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. With the use of BCH codes as ECC, we carried out simulations to demonstrate 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 in CMOS domain.

1 Introduction

Although many emerging nanoscale devices show significant future promise to sustain the Moore's Law beyond the CMOS scaling limit, there is a growing consensus [1, 8] that, at least in the short term, they cannot completely replace CMOS technology. As a result, there is substantial demand to explore the opportunities for CMOS and molecular/nano-technologies to enhance and complement each other. This naturally leads to the paradigm of hybrid CMOS/nanodevice nanoelectronics [2, 4–6, 9, 11], where an array of nanowire crossbars, with wires connected by simple nanodevices at each crosspoint, sits on the top of a CMOS circuit. 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 have been well recognized as one of the biggest challenges in the emerging hybrid nanoelectronic era [8].

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 using error correcting codes (ECC). In order to provide good defect tolerance efficiency, the repair-only approach requires 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, power, and operational latency overhead in CMOS domain. 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 [3, 10]. In [10], 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 [3]. Nevertheless, the two main issues as discussed in the above, i.e., (1) integration of defect tolerance and transient fault tolerance and (2) consideration of the fault tolerance overhead in CMOS domain, have not been addressed in prior work.

In this paper, we propose two hybrid nanoelectronic digital memory fault tolerant system design approaches. We understand that, at this early stage of nanoelectronics when relatively few preliminary experimental data 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. Hence rather than 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 relatively high defect probabilities and high transient fault rates, and (2) they can automatically adapt to the variations of the defect probabilities 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).

The presented two design approaches integrally consider defect tolerance and transient fault tolerance and share the following three features: (1) a group of ECC is used for both defect tolerance and transient fault tolerance, (2) there are no explicitly designated spare columns, rows, or words in each nanodevice memory cell array (or nanowire crossbar), and (3) for the storage of each user data block with an unique memory logical address, its ECC encoding and mapping to the physical nanodevice memory cells are integrally determined. The first approach, referred to as two-level hierarchical fault tolerance, is straightforward and easy to implement, nevertheless the achievable storage capacity quickly drops as the defect density and/or transient fault rate increases. 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 increases, while it suffers from higher implementation complexity and longer operational latency. With the use of BCH codes, we evaluated and compared these two design approaches while taking into account of the storage overhead in CMOS domain.

2 BCH Code Preliminaries and Fault Model

Because of their strong random error correction capability, binary BCH codes [7] 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) BCH code can be shortened (i.e., eliminate a certain number, say s, of information bits) to construct a t-error-correcting $(n - s, k - s)$ BCH code.

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 t-error-correcting binary BCH code under GF(2^m) with code length of n, the product of decoder silicon area and decoding latency is approximately proportional to $n \cdot t \cdot m^2$. Moreover, we note that 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.

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 that will become defective and hence make all the other connected memory cells unfunctional, we consider such short memory cell defect as nanowire defect. 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}.

3 Two Fault Tolerance Design Approaches

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), which should be able to share the same hardware encoder and decoder.

Let l_u represent the number of user bits per block in the memory. We first construct a group of binary BCH codes, denoted as $\mathcal{C}$, under the same Galois Field GF(2^m) where

$2^m - 1 > l_u$. Each BCH code $C_i \in \mathcal{C}$ is shortened (if necessary) from one primitive BCH code 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 denote $t_{max} = \max\{t_i\}$.

Given the BCH code group and memory defect map, a fault tolerance system should determine (i) which BCH code should be used for protecting each l_u-bit user data block, and (ii) 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 fault tolerance 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.

3.1 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 next to the head of the successive segment (i.e., there might be some unused memory cells in between). The location of each segment 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 as two-level hierarchical fault tolerance design and, in the following, we present a procedure to implement this design approach:

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_{tf}, and target block error rate E_{target}.

Procedure: We first exclude all the defective nanowires from the nanodevice memory physical address space[1]. 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_{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

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

Step 3: If $t_c \geq t_{def} + t_{trans}$, 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 and 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 $t_c < t_{def} + t_{trans} \leq t_{max}$ (recall that t_{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 $t_{def} + t_{trans}$ 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 equal to the maximum number of correctable errors of the currently selected BCH code, and go to Step 2.

Step 5 If $t_{def} + t_{trans} > t_{max}$ (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.

Suppose each nanodevice memory cell array contains N memory cells and the code group $\mathcal{C}$ contains h different BCH codes. For each located segment, we need to store

[1] 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.

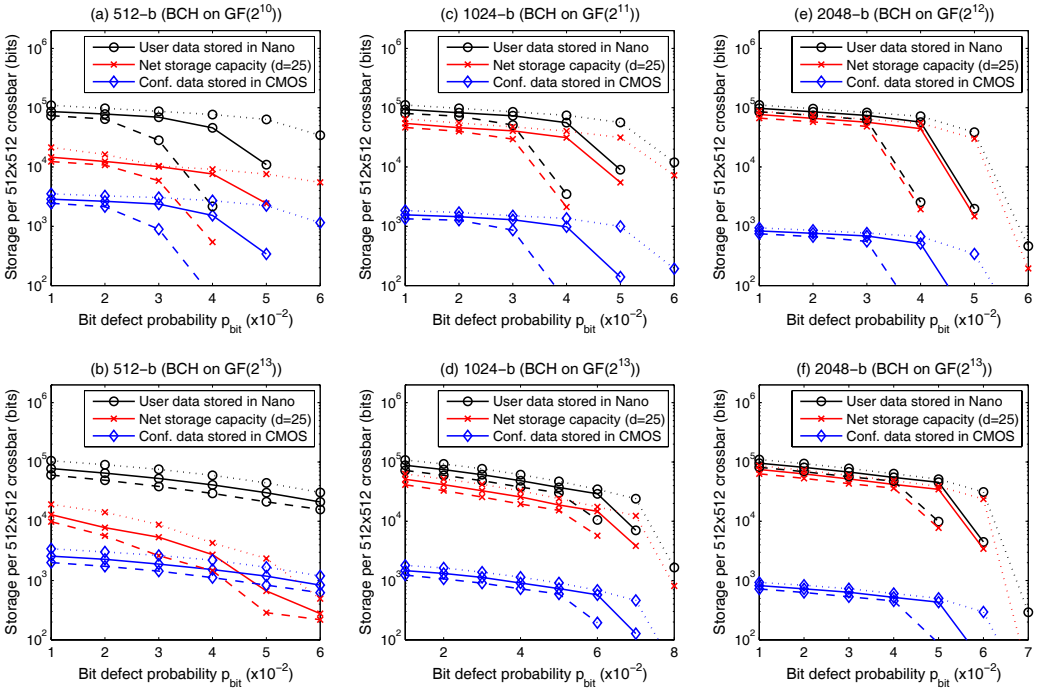

Figure 1. Simulation results on the average storage capacity per 512×512 nanodevice memory cell array using the two-level hierarchical fault tolerance approach. The dotted, solid, and dashed curves correspond to the transient fault rates of 0, 1‰, and 5‰, respectively.

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_{nano} and S_{CMOS}, respectively. To take into account of the storage overhead in CMOS domain, we define the *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. 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 $p_{wire} = 0.3$; 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 l_u, including 2048, 1024, and 512. We constructed four BCH code groups as listed in the following table, in which the parameter r_{max} represents the number of redundant bits required for correcting t_{max} errors. When constructing each

Table 1. 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})
Max. Length	1023	2047	4095	8191
t_{max}	57	106	198	366
r_{max}	510	1023	2038	4095

BCH code group, for each $t \in [1, t_{max}]$ we search the BCH code that can correct t errors with the least redundancy and put this BCH code into the code group (notice that it is possible that a few consecutive values of t may correspond to the same BCH code).

Fig. 1 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 stor-

age 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. As shown in the figures, using BCH code group on larger Galois Fields can directly improve the storage capacity (or fault tolerance efficiency) due to the stronger error correcting capability, which comes with the cost of higher ECC encoder and 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.

3.2 Approach II: Three-Level Hierarchical Fault Tolerance

In the above two-level hierarchical design approach, we always attempt to locate a consecutive memory cell segment to store the coded data block. Hence, with high bit defect probabilities, the total number of defective memory cells within a segment may accumulate very quickly and easily exceed the maximum error correcting capability. This will become more serious as transient fault rate increases. Therefore, as shown in Fig. 1, 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 consecutive 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 2, 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 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. 2.

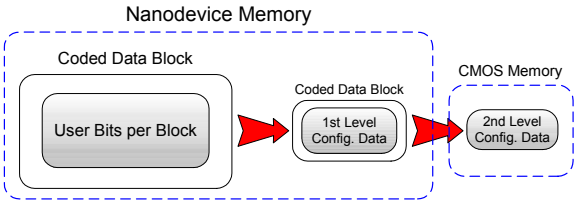

Figure 2. Storage hierarchy in the three-level hierarchical fault tolerance system.

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:

Input: the number of user bits per block l_u, indivisible memory unit length l_c, BCH code group $\mathcal{C}$ and the de-

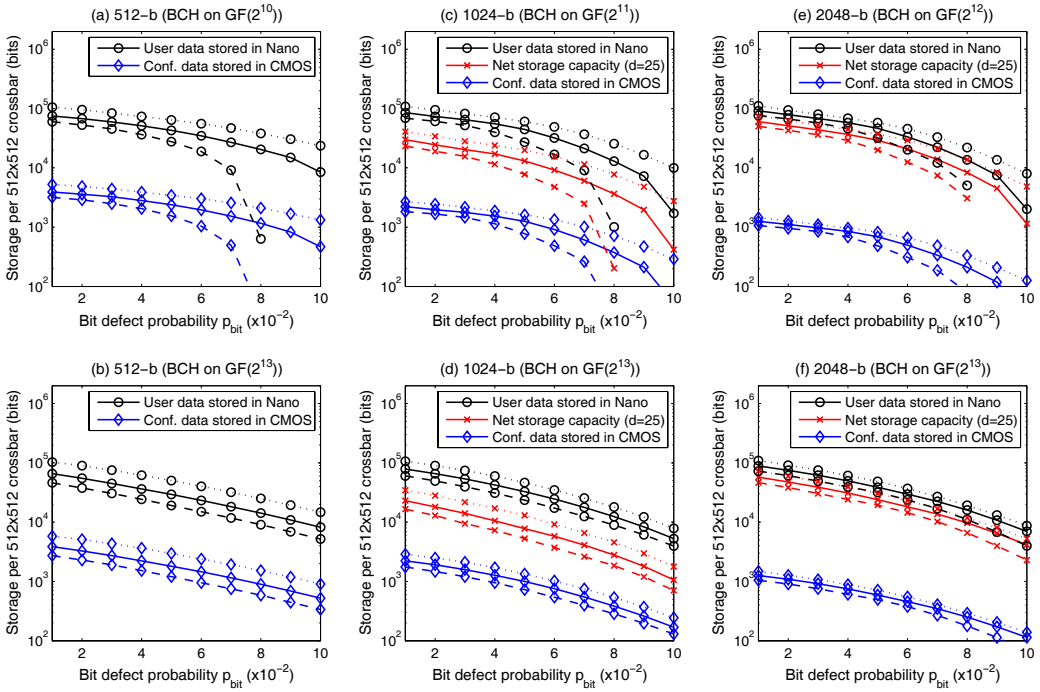

Figure 3. Simulation results on the average storage capacity per 512×512 nanodevice memory cell array using the three-level hierarchical fault tolerance approach. The dotted, solid, and dashed curves correspond to the transient fault rates of 0, 1‰ and 5‰, respectively.

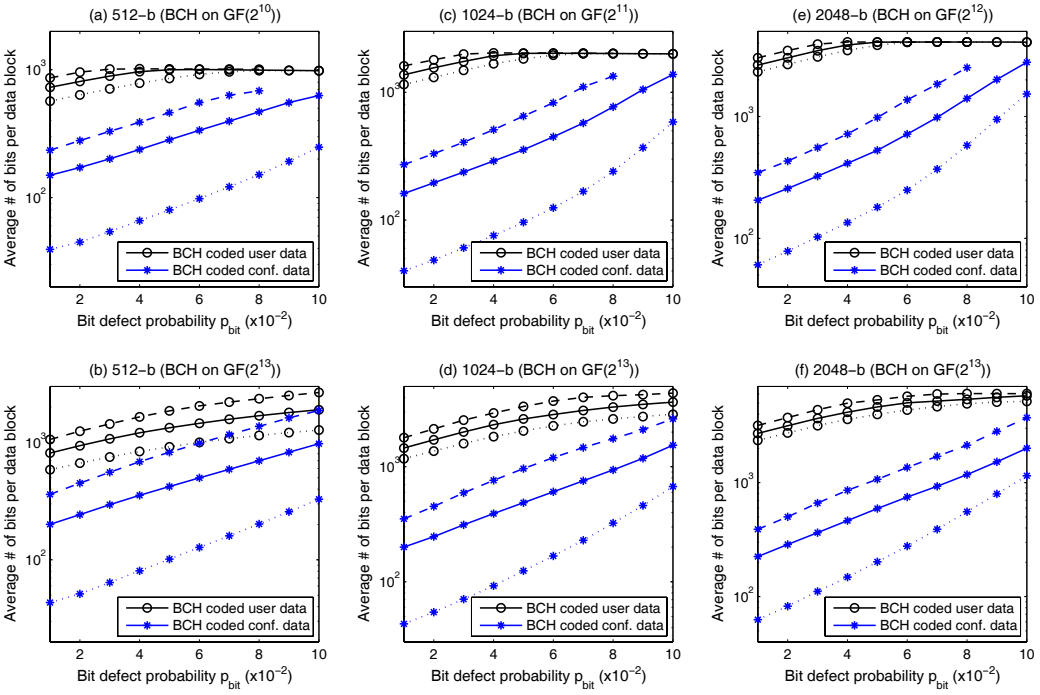

Figure 4. Simulation results on the average number of bits per BCH coded user data block and BCH coded configuration data block. The dotted, solid, and dashed curves correspond to the transient fault rates of 0, 1‰ and 5‰, respectively.

gree m of the underlying GF(2^m), nanodevice memory cell array defect map, transient fault rate p_{tf}, and target block error rate E_{target}.

Procedure: Again, 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 fall 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 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 3.1.

Step 3: If $t_c \geq t_{def} + t_{trans}$ (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 $t_c < t_{def} + t_{trans} \leq t_{max}$, then select a BCH code from $\mathcal{C}$ that can correct $t_{def} + t_{trans}$ 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 equal to the maximum number of correctable errors of the currently selected BCH code, and go to Step 2.

Step 5 If $t_{def} + t_{trans} > t_{max}$, 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 3.1) 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.

To demonstrate the effectiveness of this design approach, we carried out simulations under the same configurations as used in Section 3.1: each nanodevice memory cell array is 512×512; nanowire defect probability $p_{wire} = 0.3$; target block error rate $E_{target} = 1 \times 10^{-15}$; 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., 2048, 1024, and 512) 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. 3 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 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 $d = 25$. Comparing the simulation results in Fig. 3 and Fig. 1, we have 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.

Nevertheless, due to the operation on the first level configuration data stored in the nanodevice memory, the three-level fault tolerance approach is subject to longer memory access latency. Since the memory access latency is approximately proportional to the BCH codeword length, Fig. 4 shows the comparisons between the average length of BCH coded user data blocks and BCH coded first level configuration data, based on which we may approximately quantify the memory access latency overhead. The simulation results suggest that, at low defect probabilities and/or transient fault rates, the latency overhead incurred by the first level configuration data is much less (by 1 or 2 orders of magnitudes) than that of the coded user data block; while as defect probabilities and/or transient fault rates increase, the latency overhead incurred by the first level configuration data quickly increases and fall into the same order of magnitudes as that of the coded user data block.

Finally, we note that both design approaches are subject to a trade-off between the BCH coding system complexity and achievable storage capacity: as we construct BCH codes under larger Galois Fields to obtain stronger error correcting capability and hence larger storage capacity, the BCH coding system (particularly decoder) implementation complexity will increase accordingly. Such trade-off can be approximately quantified based on the rule that the product of BCH decoder silicon area and decoding latency is approximately proportional to $n \cdot t \cdot m^2$, where n is the code length, t is the maximum number of correctable errors, and m is the degree of the underlying binary Galois Field.

4 Conclusions

In this paper, we presented two fault tolerance design approaches that integrally address the defect tolerance and 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 ECC for both defect tolerance and transient fault tolerance, absence of explicitly designated spare rows, columns and/or words, and integration of ECC selection and logical-to-physical address mapping. These two fault tolerance design approaches seek different trade-offs among the achievable storage capacity, robustness to defect statistics variations, implementation complexity, and operational latency and CMOS storage overhead. By using BCH codes as ECC, we 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 (close to 1%) and transient fault rates (up to 5‰), and can readily adapt to large defect statistics variations. Current work is being directed to practically evaluating the BCH code coding system implementation silicon cost and latency overhead, and investigating more comprehensive comparison among different design approaches with various configurations while taking into account of the CMOS storage and BCH coding system implementation overhead.

References

[1] S. I. Association. *The International Technology Roadmap for Semiconductors (ITRS)*. http://public.itrs.net/Files/2003ITRS/Home2003.htm, 2003.

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

[3] A. DeHon, S. C. Goldstein, P. J. Kuekes, and P. Lincoln. Nonphotolithographic nanoscale memory density prospects. *IEEE Transactions on Nanotechnology*, 4:215–228, March 2005.

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

[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*, 97(3):034301, 2005.

[6] 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/.

[7] S. Lin and D. J. Costello. *Error Control Coding: Fundamentals and Applications (2nd Ed.)*. Prentice Hall, 2004.

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

[9] 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*, 91:1940–1957, Nov. 2003.

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

[11] M. Ziegler and M. Stan. CMOS/nano co-design for crossbar-based molecular electronic systems. *IEEE Transactions on Nanotechnology*, 2:217–230, Dec. 2003.