per unit power per unit area, the MORA RC shows over 4× higher throughput efficiency compared to AsAP and AsAP-II architectures. While pure MOPS numbers for MORA are comparable to those of competing architectures, MORA achieves significantly higher MOPS/mm$^2$ implying higher throughput efficiency and processing density.

It is also worth noting that the actual advantage of the proposed architecture comes in the form of throughput efficiency and resource cost. For instance, when performing 8 × 8 2-D-DCT, the proposed architecture shows a performance of 9.945 MSamples/s and 21.36 MSamples/s compared to 2.8 MSamples/s reported for the SmartCell architecture. In terms of cycle count comparison, the proposed RC requires 72 cycles with a latency of 36 cycles when processing 8 DCT blocks in parallel, compared to 64 and 96 cycles required for single block computation by the Montium [5] and RaPid [6] processors, respectively. To get a fair idea of throughput efficiency per power and area resources, we have also included an estimation of performance per unit power per unit area. This figure of merit accounts for both the architecture as well as circuit design decisions and gives a fairly accurate idea of the overall resource efficiency and resource cost of the architecture.

## IV. Conclusion

This brief presented recent findings in the design and performance evaluation of a low-complexity low-cost reconfigurable processor to be part of MORA, our coarse-grained reconfigurable array. The processor operating at its optimum performance-power point was found to be capable of delivering a peak throughput of 75 MOPs/mW. When evaluated for popular benchmark algorithms, the processor offered over 4× advantage in resource efficiency compared to recently proposed architectures and seems a promising solution for resource-efficient reconfigurable computing.

## References

[1] S. Chalamalasetti, S. Purohit, M. Margala, and W. Vanderbauwhede, "MORA: Architecture and programming model for a resource efficient coarse grained reconfigurable processor," in *Proc. ACM NASA/ESA Conf. Adapt. Hardw. Syst.*, 2009, pp. 389–396.

[2] W. Vanderbauwhede, M. Margala, S. Chalamalasetti, and S. Purohit, "Programming model and low level language for a coarse grained reconfigurable multimedia processor," in *Proc. Int. Conf. Eng. Reconfig. Syst. Algorithms*, 2009, pp. 375–380.

[3] S. Purohit, M. Lanuzza, S. Perri, P. Corsonello, and M. Margala, "Design and evaluation of an energy-delay-area efficient data path for coarse grained reconfigurable computing systems," *J. Low Power Electron.*, vol. 5, no. 3, pp. 326–338, Oct. 2009.

[4] Z. Yu, M. J. Meeuwsen, R. W. Apperson, O. Sattari, M. Lai, J. W. Webb, E. W. Work, D. Truong, T. Mohsenin, and B. M. Baas, "AsAP: Asynchronous array of simple processors," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 695–705, Mar. 2008.

[5] M. Butts, "Synchronization through communication in a massively parallel processor array," *IEEE Micro*, vol. 27, no. 5, pp. 32–40, Sep.–Oct. 2007.

[6] C. Liang and X. Huang, "SmartCell: An energy efficient coarse-grained reconfigurable architecture for stream-based applications," *EURASIP J. Embedd. Syst.*, vol. 2009, pp. 1–15, Jan. 2009.

[7] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, P. Mejia, A. Tran, J. Webb, E. Work, Z. Xiao, and B. Baas, "A 167-processor 65 nm computational platform with per-processor dynamic supply voltage," in *Proc. Symp. VLSI Circuits*, 2008, no. C 3.1, pp. 1–2.

[8] P. Heysters and G. Smit, "Mapping of DSP algorithms on the MONTIUM architecture," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2003, pp. 180–185.

[9] C. Ebeling, D. C. Cronquist, and P. Franklin, "RaPiD-reconfigurable pipelined datapath," in *Proc. 6th Int. Workshop Field-Program. Logic Appl.*, 1996, pp. 126–135.

# Error Rate-Based Wear-Leveling for NAND Flash Memory at Highly Scaled Technology Nodes

Yangyang Pan, Guiqiang Dong, and Tong Zhang

*Abstract*—**This brief presents a NAND Flash memory wear-leveling algorithm that explicitly uses memory raw bit error rate (BER) as the optimization target. Although NAND Flash memory wear-leveling has been well studied, all the existing algorithms aim to equalize the number of programming/erase cycles among all the memory blocks. Unfortunately, such a conventional design practice becomes increasingly suboptimal as inter-block variation becomes increasingly significant with the technology scaling. This brief presents a dynamic BER-based greedy wear-leveling algorithm that uses BER statistics as the measurement of memory block wear-out pace, and guides dynamic memory block data swapping to fully maximize the wear-leveling efficiency. Simulations have been carried out to quantitatively demonstrate its advantages over existing wear-leveling algorithms.**

*Index Terms*—**Error rate, process variation, solid state drive, wear leveling.**

## I. Introduction

NAND Flash memory is subject to gradual memory cell wear-out caused by programming/erase (P/E) operations. This leads to a P/E cycling endurance limit that continuously degrades with technology scaling. Since erase operations are carried out in the unit of block and gradual memory cell wear-out is reflected as increasing storage raw bit error rate (BER), designers have been developing techniques to embrace the effects of technology scaling from two aspects: 1) more powerful error correction code (ECC) and signal processing solutions that can tolerate higher raw BER [1], [2] and 2) effective wear-leveling solutions that can equalize wear-out pace among all the blocks as much as possible.

Since the wear-out pace of each memory block is reflected as its raw BER, wear-leveling should ideally aim to equalize the raw BER among all the blocks. However, all the existing wear-leveling algorithms [3], [4] simply use the number of P/E cycles as the equalization target. Although memory block raw BER heavily depends on the number of P/E cycles, given the maximum allowable raw BER, different memory blocks may have largely different endurance due to fabrication process variation. Such a conventional design practice is a reasonable simplification for older technology nodes, where the memory block P/E cycling endurance is large (e.g., several hundreds of thousands) and process variation is relatively very small. However, it becomes increasingly suboptimal for newer technology nodes (e.g., 35 nm and below), where the memory block P/E cycling endurance has significantly dropped (e.g., few tens of thousands or even few thousands) and process variation has become relatively much more significant.

Very intuitively, the above discussion suggests that we should explicitly use memory block BER as the equalization target in wear-leveling at highly scaled technology nodes. Design of wear-leveling algorithms is inherently subject to a tradeoff between wear-

Manuscript received February 10, 2012; revised May 21, 2012; accepted July 21, 2012. Date of publication September 4, 2012; date of current version June 21, 2013.

The authors are with the Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180 USA (e-mail: dongguiqiang@gmail.com; yyangpan@gmail.com; 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/TVLSI.2012.2210256

leveling efficiency and garbage collection efficiency, and prior work has well demonstrated the advantages of dynamic wear-leveling schemes [3]–[8]. In this brief, we present a dynamic BER-based greedy (DBG) wear-leveling algorithm. Besides explicitly using BER as equalization target, it applies BER-based dynamic block data swapping to further on-the-fly equalize the memory block wear-out pace among all the memory blocks. We carried out trace-based simulations to demonstrate the effectiveness of the proposed BER-based wear-leveling algorithm. Results clearly show the advantages of such BER-based wear-leveling over existing wear-level strategies, and further demonstrate the wear-leveling efficiency versus garbage collection efficiency tradeoffs.

## II. Background and Prior Work

The objective of wear-leveling is to equalize the wear-out process among all the blocks, which can maximize the overall NAND Flash memory lifetime. Conventional wear-leveling algorithms make all the blocks undergo almost the same amount of P/E cycles. In the run time, each block may contain both valid and invalid pages. Before one block can be erased, the data in all its valid pages must be copied to other blocks, after which all the pages in this block can be considered as garbage and hence this block can be erased. This process is referred to as garbage collection. The essential task of wear-leveling is to determine which block should be erased and hence become ready to store new data. Various wear-leveling algorithms can be categorized as either static or dynamic, depending on whether run-time data access characteristics are taken into account. In the following, we briefly describe several well-known existing wear-leveling algorithms. Readers are referred to a survey paper [9] for much detailed discussions.

The simplest wear-leveling algorithm is the so-called static random (SR) algorithm, which randomly chooses a block to erase regardless of how many times the block has been erased and how many valid pages are contained in the block. Due to its random nature, the SR algorithm can keep the P/E cycling of different blocks almost the same throughout the memory lifetime. Nevertheless, it suffers from poor garbage collection efficiency. The static wear-ignore greedy (SWIG) algorithm aims to erase the block with the minimum number of valid pages, regardless of how many times the block has been erased. Clearly, this algorithm can improve the garbage collection efficiency at the penalty of wear-leveling efficiency. The static wear-aware greedy (SWAG) algorithm determines which block to be erased by jointly considering both the wear-leveling efficiency and garbage collection efficiency. In particular, it aims to erase the block with the minimal number of valid pages subject to a P/E cycling constraint.

By on-the-fly identifying the “hot” and “cold” data, dynamic wear-leveling algorithms [10] periodically swap the hot and cold data among different blocks in order to improve the wear-leveling efficiency. The dynamic erase-number-based greedy (DEG) algorithm identifies the “hotness” of each block based on how long has elapsed since the block has been erased, and periodically swaps the data on the coldest and hottest blocks. It always erases the coldest block during wear-leveling.

## III. Proposed Wear Leveling Algorithm

Continuous Flash memory technology scaling causes increasingly severe inter/intra-die process variation, leading to significant variability of critical memory cell device parameters, such as oxide thickness and gate width/length. This results in significant variations of memory cell threshold voltage distribution over P/E cycling [11]. Therefore, under the same P/E cycling, pages in different memory blocks may have largely different worst-case raw BER. Equivalently, given the same ECC, different memory blocks may have largely different P/E cycling endurance. To quantitatively demonstrate such inter-block variation, Fig. 1 shows the normalized bit error number per 1-MB block probability distribution based on measurements among 1000 blocks of a 35-nm multi level cell (MLC) NAND Flash memory chip under the same P/E cycling of 15 K. Each block has 64 physical wordlines and 256 4-kB logical pages. Similar measurement results have been reported in [12].

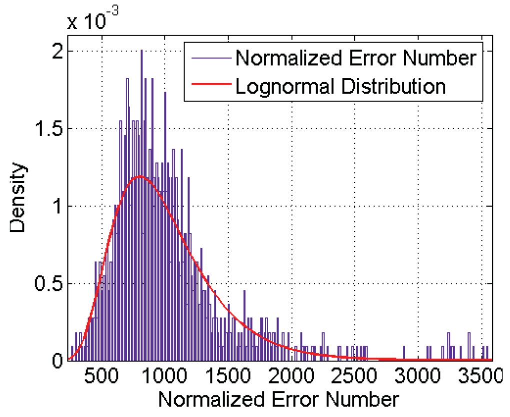

Fig. 1. Normalized bit error number per block with 1-MB capacity probability distribution function, based on measurements among 1000 blocks of a 35-nm MLC NAND Flash memory chip under the same P/E cycling of 15 K.

The significant inter-block P/E cycling endurance variation makes conventional wear-leveling algorithms largely suboptimal, since conventional wear-leveling algorithms assume an equal P/E cycling endurance of all the blocks and aim to equalize the P/E cycling number among all the blocks. In the presence of significant inter-block P/E cycling endurance variation, such a conventional design practice makes NAND Flash memory lifetime limited by the worst block and leaves many blocks largely under-utilized. Assume a Flash memory system contains total $M$ memory blocks, and let $D_i$ denote the achievable P/E cycling endurance of the $i$-th block. Ideally, this memory system can sustain total $\sum_{i=1}^{M} D_i$ block erase operations. If we use conventional wear-leveling schemes, the total number of erase operations is upper bounded by $\min(D_i)\cdot M$, i.e., the percentage of NAND Flash memory under-utilization is lower bounded by

$$\frac{\sum_{i=1}^{M} D_i - \min(D_i)\cdot M}{\sum_{i=1}^{M} D_i}. \tag{1}$$

With the technology scaling, the inter-block P/E cycling endurance variation becomes larger and meanwhile the magnitude of P/E cycling endurance continues to reduce. Therefore, conventional wear-level schemes will become increasingly suboptimal and result in increasingly larger memory under-utilization. Very intuitively, in order to address this issue, wear-leveling algorithms should explicitly take into account of inter-block P/E cycling endurance variation. Due to the better performance of dynamic wear-leveling schemes, this brief only focuses on incorporating the awareness of inter-block P/E cycling endurance variation into dynamic wear-leveling. Nevertheless, the proposed design strategy can be straightforwardly applied to improve static wear-leveling.

In this brief, we propose a DBG wear-leveling algorithm. The key is to explicitly use the raw BER statistics of each memory block as the measurement of its wear-out pace. We note that the raw BER statistics can be obtained and updated when one block has been read

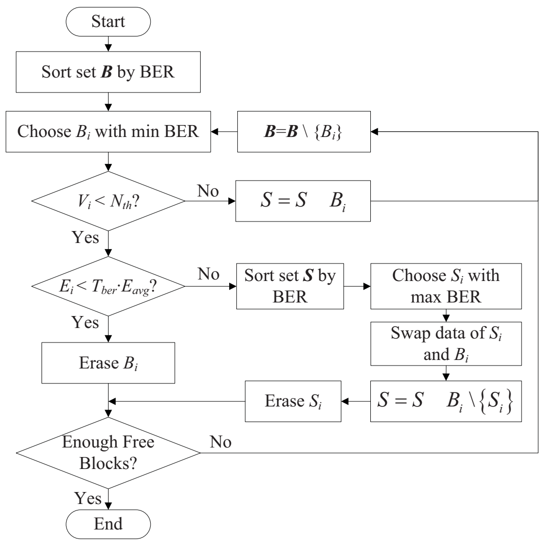

Fig. 2. Illustration of the operational flow diagram of the proposed BER-based wear-leveling algorithm.

most recently, i.e., when any page within one block is read, it will be processed by an ECC decoder at the controller, and the raw BER statistics can be obtained by comparing the input and output of the ECC decoder. In addition, we apply BER-based dynamic block data swapping to further on-the-fly equalize the memory block wear-out pace among all the blocks. Let $V_i$ and $E_i$ denote the number of valid pages and BER of the $i$-th block, let $E_{\text{avg}}$ denote the average BER of all the blocks, and $T_{\text{ber}} \in [0, 1]$ represent a pre-defined threshold factor. In addition, let $N_{\text{page}}$ denote the total number of pages per block, we define $N_{\text{th}} \in [0, N_{\text{page}}]$ as a valid page number threshold. Fig. 2 shows the flow diagram of this proposed dynamic BER-based wear-leveling algorithm.

In Fig. 2, $B$ denotes all the dirty blocks to be cleaned, and $S$ denotes the dirty blocks with high error rate and valid pages number bigger than a pre-defined threshold $N_{\text{th}}$. This dynamic wear-leveling algorithm employs run-time data swapping to equalize the BER among all the blocks while jointly considering the tradeoff among the garbage collection efficiency, wear-leveling efficiency, and data swapping overhead. In the proposed algorithm, instead of cleaning the blocks in $S$, we swap the data of the block in $S$ (i.e., hot data) with the data of blocks in $B$ with low error rate and valid pages number smaller an $N_{\text{th}}$ (i.e., cold data). As shown in Fig. 2, it uses two pre-defined parameters, $T_{\text{ber}}$ and $N_{\text{th}}$, to adjust the tradeoff, where $T_{\text{ber}}$ controls the allowable variation of BER among all the blocks and $N_{\text{th}}$ explicitly sets a constraint to ensure garbage collection efficiency. As we reduce the parameter $T_{\text{ber}}$ and/or increase the parameter $N_{\text{th}}$, we can improve the wear-leveling efficiency at the penalty of garbage collection efficiency, and meanwhile more data swapping operations will occur, leading a higher data swapping overhead. Since practical workloads may have varying data access characteristics in the run time, the NAND Flash memory controller can on-the-fly monitor the garbage collection efficiency, wear-leveling efficiency, and data swapping overhead, based on which the controller can adjust these parameters accordingly.

## IV. Simulation Results

We carried out trace-based simulations to demonstrate the effectiveness of the proposed BER-based wear-leveling algorithm. We use the SSD module [9] in DiskSim [13], and use three workload traces, including Iozone, Postmark, and a synthetic workload trace [9] consisting of 25 M read requests and 25 M write requests, among which 50% requests are random and the other 50% requests are sequential. In the SSD module, each chip package contains two dies that share an eight-bit I/O bus and a number of common control signals, and each die contains four planes and each plane contains 2048 blocks. Each block contains 64 4-kB pages. We assume the use of SLC NAND Flash memory, for which the latencies of program, read, and erase operations are 200 $\mu$s, 25 $\mu$s, and 1.5 ms, respectively. Following the ONFI 2.0 specification [14], we set the interface bus bandwidth of NAND Flash chip as 133 Mb/s.

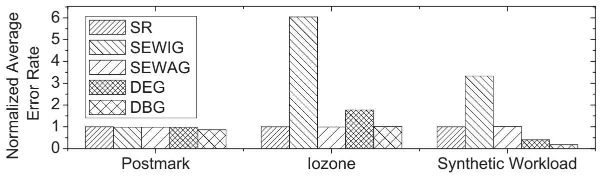

Fig. 3. Normalized average BER when different algorithms are used.

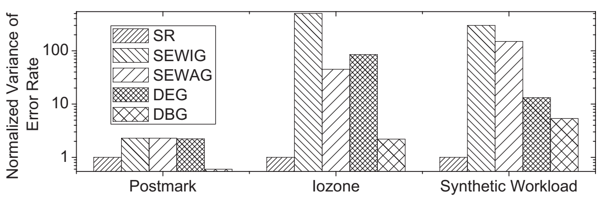

Fig. 4. Normalized variance of BER when different algorithms are used.

### A. Modeling of Process Variation Impacts

We model the impact of process variation on NAND Flash memory error rate distribution according to our measurement results on 35-nm MLC device. As shown in Fig. 1, under the P/E cycling of 15 K, the block BER could spread more than 13×, i.e., among the 1000 blocks, the minimum block BER is about $8.3 \times 10^{-5}$ and the maximum block BER is about $1.1 \times 10^{-3}$. Measurement results show that both BER and BER spread grow with the number of P/E cycles, and we approximately model the block BER as $e = 10^{-s \cdot N}$, where $N$ is the number of P/E cycles and $s$ is one parameter reflecting different BER growth rates among different memory blocks due to process variation. The maximum and minimum values of $s$ are $s_{\text{max}} = 4.6 \times 10^{-4}$ and $s_{\text{min}} = 2.8 \times 10^{-4}$, hence the mean of $s$ is $\mu = (s_{\text{max}} + s_{\text{min}})/2 = 3.7 \times 10^{-4}$. Measurement results suggest that we may approximately assume the parameter $s$ follows a bounded Gaussian distribution, i.e., let $f(s)$ denote the probability density function of $s$, we have

$$f(s) = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(s-\mu)^2}{2\sigma^2}}, & \text{if } s_{\text{min}} \geq s \geq s_{\text{max}} \\ 0, & \text{else.} \end{cases} \tag{2}$$

The standard deviation $\sigma$ is set as $9 \times 10^{-5}$. Assuming the ECC can tolerate up to $10^{-3}$ of BER, we have that P/E cycling endurance of all the memory blocks fall into the range of [15 000, 24 600].

### B. Wear-Leveling Efficiency

In the presence of significant process variation, wear-leveling efficiency of NAND Flash memory should be directly evaluated in terms of block BER instead of P/E cycling number, i.e., a wear-leveling algorithm, which can result in less average and variance of error rates of all the blocks, has a better wear-leveling efficiency. Figs. 3 and 4 show the simulation results on average and variance of BER

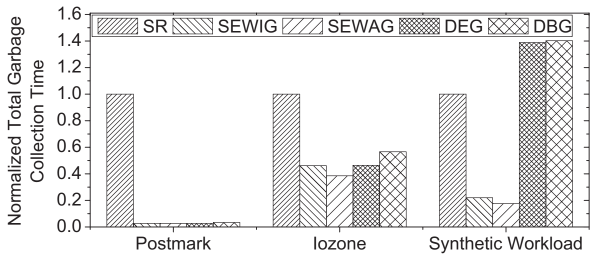

Fig. 5. Normalized total garbage collection time (including block erase time).

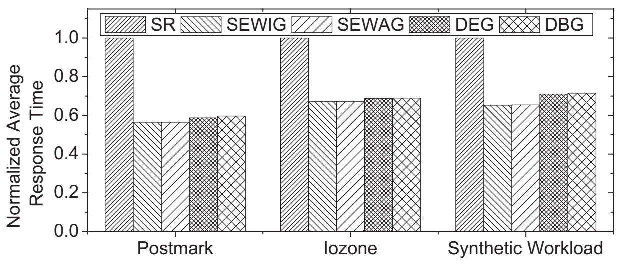

Fig. 6. Simulated SSD average response time.

among all the blocks when different wear-leveling algorithms are used. Regarding the parameters used in the proposed DBG algorithm, we set $T_{\text{ber}} = 0.8$ and $N_{\text{th}} = 20$. Besides the proposed BER-based DBG algorithm, we consider the SR algorithm, SWIG algorithm, SWAG algorithm, and DEG algorithm, which are all briefly discussed in Section II. To facilitate the comparison, the average and variance of BER are normalized against the case of using the SR algorithm.

The results clearly show that the proposed BER-based DBG algorithm almost consistently outperforms the other algorithms in terms of both average and variance of BER. This is a very reasonable result since the proposed DBG algorithm explicitly uses the block BER as the metric during wear-leveling. The effectiveness of the conventional DG algorithm tends to vary over different workloads, and is consistently worse than that of the proposed DBG algorithm. In addition, the simplest SR algorithm has better wear-leveling efficiency than other conventional static algorithms. This is mainly because SR algorithm does not take into account of garbage collection efficiency at all. The static algorithm SWIG has the worst wear-leveling efficiency, since it most heavily trades wear-leveling efficiency for garbage collection efficiency.

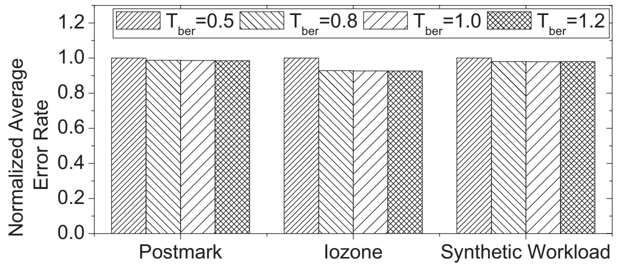

Fig. 7. Normalized average of error rate with different $T_{\text{ber}}$.

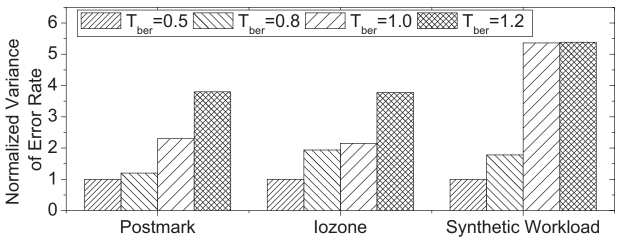

Fig. 8. Normalized variance of error rate with different $T_{\text{ber}}$.

### C. Garbage Collection Efficiency

We further studied and compared the garbage collection efficiency of different algorithms. Intuitively, in order to ensure a higher wear-leveling efficiency (i.e., more equalized BER among all the blocks), we may have to move and copy more data among all the blocks, leading to a lower garbage collection efficiency. Therefore, any wear-leveling algorithm is inherently subject to a tradeoff between wear-leveling efficiency and garbage collection efficiency. Fig. 5 shows the normalized amount of time devoted to garbage collection (including block erase).

Combining the results shown in Figs. 3 and 5, we can clearly see that the algorithms with higher wear-leveling efficiency (including SR, DEG, and proposed DBG algorithms) induce longer garbage collection time (i.e., lower garbage collection efficiency). The SR algorithm tends to have the worst garbage collection efficiency. This is because it randomly chooses the blocks without considering the valid pages on the blocks. The other four algorithms explicitly take into account of the number of valid pages in each block, hence they tend to have better garbage collection efficiency. The two static algorithms, SWIG and SWAG, have better garbage collection efficiency than the two dynamic algorithms, DEG and the proposed DBG. This is because dynamic algorithms invoke run-time data swapping to improve wear-leveling efficiency, which nevertheless induces more data movement and hence lower garbage collection efficiency.

### D. System Performance

The above results demonstrate that the proposed BER-based DBG algorithm can noticeably improve the wear-leveling efficiency over existing solutions, which can lead to a longer solid state drive lifetime. This is increasingly desirable as continuous technology scaling inevitably degrades the NAND Flash memory cycling endurance. However, the above results also show that the proposed DBG algorithm has relatively low garbage collection efficiency, especially compared with conventional static wear-leveling algorithms. From the system perspective, different garbage collection efficiency is reflected as different impact on the system speed performance, since a lower garbage collection efficiency corresponds to more data movement that may more noticeably interfere with normal I/O requests. Therefore, system-level response time can be used as a more relevant metric to evaluate and compare different wear-leveling algorithms in terms of garbage collection effects.

Fig. 6 shows the simulated average response time. The SR algorithm consistently has much worse system speed performance compared with all the other algorithms. Hence, the SR algorithm is not practically attractive, in spite of its relatively good wear-leveling efficiency. The results show that the proposed DBG algorithm can achieve almost the same average response time as the conventional DG algorithm. Compared with the two static algorithms, SWIG and SWAG, these two dynamic wear-leveling algorithms only suffer around 3% system speed degradation. With its noticeable advantages on wear-leveling efficiency and almost negligible disadvantages on system speed performance, this proposed BER-based DBG algorithm can be a preferable solution for future solid state storage systems.

### E. Sensitivity Analysis

In the proposed BER-based wear-leveling algorithm, the parameter $T_{\text{ber}}$ plays a very important role in determining the overall perfor-

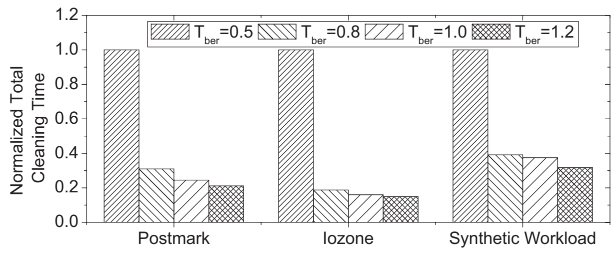

Fig. 9. Normalized total garbage collection time (including block erase time) with different $T_{\text{ber}}$.

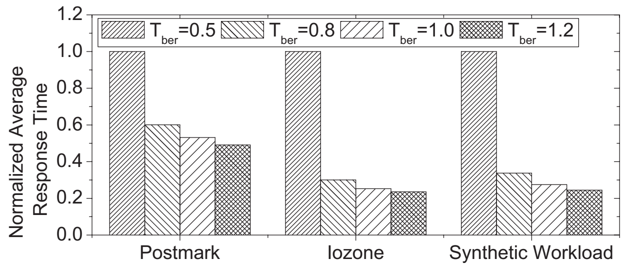

Fig. 10. Normalized average system response time with different $T_{\text{ber}}$.

mance. As we reduce the value of factor $T_{\text{ber}}$, we can increase the wear-leveling efficiency and decrease the garbage collection efficiency. The above simulations were carried out by fixing $T_{\text{ber}}$ as 0.8. In the following, we show further simulation results with four different $T_{\text{ber}}$, i.e., 0.5, 0.8, 1.0, and 1.2. As shown in Figs. 7 and 8, a larger $T_{\text{ber}}$ increases the variance of the error rate and keeps the average error almost the same, thus decreasing the wear-leveling efficiency. Meanwhile, as shown in Figs. 9 and 10, a larger $T_{\text{ber}}$ reduces the data swapping operation and reduces the total erasing time, thus increasing the garbage collection efficiency and the system speed performance. The simulation results suggest $T_{\text{ber}}$ of 0.8 appear to be an appropriate configuration considering the tradeoff between wear-leveling efficiency and system speed performance.

## V. Conclusion

Conventional NAND Flash memory wear-leveling algorithms are becoming increasingly suboptimal as the technology scaling continue to degrade memory P/E cycling endurance and increase process variation. As the very natural option, we must replace the number of memory block P/E cycles with the memory block BER as the optimization target of wear-leveling algorithms for NAND Flash memory at highly scaled technology nodes. Following this simple intuition, this brief presented a dynamic BER-based wear-leveling algorithm. Based upon measurement results of 35-nm NAND Flash memory chips and an SSD simulator, we carried out extensive simulations and showed that the proposed BER-based algorithm can noticeably improve the wear-leveling efficiency while maintain almost the same impact on the memory system speed performance compared with conventional design solutions.

## References

[1] H. Choi, W. Liu, and W. Sung, “VLSI implementation of BCH error correction for multilevel cell NAND Flash memory,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 5, pp. 843–847, May 2010.

[2] C. Yang, Y. Emre, and C. Chakrabarti, “Product code schemes for error correction in MLC NAND Flash memories,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 12, pp. 2302–2314, 2012.

[3] E. Gal and S. Toledo, “Algorithms and data structures for Flash memories,” *ACM Comput. Surv.*, vol. 37, no. 2, pp. 138–163, 2005.

[4] A. B. Aroya and S. Toledo, “Competitive analysis of Flash-memory algorithms,” in *Proc. Ann. Eur. Symp.*, 2006, pp. 100–111.

[5] M. Wu and W. Zwaenepoel, “eNVy: A nonvolatile main memory storage system,” in *Proc. 14th Workshop Workstation Operat. Syst.*, Oct. 1993, pp. 116–118.

[6] L.-P. Chang, T.-W. Kuo, and S.-W. Lo, “Real-time garbage collection for Flash-memory storage systems of real-time embedded systems,” *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 4, pp. 837–863, 2004.

[7] M.-L. Chiang and R.-C. Chang, “Cleaning policies in mobile computers using Flash memory,” *J. Syst. Softw.*, vol. 48, no. 3, pp. 213–231, 1999.

[8] H. Kim and S. Lee, “An effective Flash memory manager for reliable Flash memory space management,” *IEICE Trans. Inform. Syst.*, vol. 85, no. 6, pp. 950–964, 2002.

[9] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, “Design tradeoffs for SSD performance,” in *Proc. USENIX Ann. Tech. Conf.*, 2008, pp. 57–70.

[10] J.-W. Hsieh, T.-W. Kuo, and L.-P. Chang, “Efficient identification of hot data for Flash memory storage systems,” *ACM Trans. Storage*, vol. 2, no. 1, pp. 22–40, Feb. 2006.

[11] A. Spessot, A. Calderoni, P. Fantini, A. S. Spinelli, C. M. Compagnoni, F. Farina, A. L. Lacaita, and A. Marmiroli, “Variability effects on the VT distribution of nanoscale NAND Flash memories,” in *Proc. IEEE Int. Rel. Phys. Symp.*, May 2010, pp. 970–974.

[12] J. Fitzpatrick, “Flash secrets revealed,” in *Proc. Flash Memory SUMMIT*, Aug. 2010.

[13] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, “The disksim simulation environment version 4.0 reference manual,” Carnegie Mellon Univ., Pittsburgh, PA, Rep. CMU-PDL-08-101, May 2008.

[14] *Open NAND Flash Interface Specification* (2011). [Online]. Available: http://onfi.org/

# Reduced Power Transition Fault Test Sets for Circuits With Independent Scan Chain Modes

Irith Pomeranz

***Abstract*— This brief considers circuits with multiple scan chains where each scan chain can operate in shift, functional, or hold mode independently of the other scan chains. For circuits where the hardware overhead of controlling the scan chains independently is acceptable, this brief describes a procedure whose goal is to generate a test set that achieves the same transition fault coverage as a test set that consists of both broadside and skewed-load tests, but where the shift mode is used as few times as possible during the first patterns of the tests. This allows the circuit to operate closer to its functional operation conditions, and reduces the power dissipation during the second patterns of the tests, which are applied at-speed.**

***Index Terms*— Design-for-testability, full-scan circuits, switching activity, transition faults, two-pattern tests.**

## I. Introduction

Two-pattern tests are applied to scan circuits in order to detect delay faults. A two-pattern test can be represented as $<s_1v_1, s_2v_2>$, where $s_1v_1$ is the first pattern of the test, and $s_2v_2$ is the second pattern. For $i = 1$ and 2, $s_i$ is a state and $v_i$ is a primary input vector. The state $s_1$ is the scan-in state. The first pattern, $s_1v_1$, is

Manuscript received December 16, 2011; revised April 28, 2012; accepted June 26, 2012. Date of publication July 31, 2012; date of current version June 21, 2013.

The author is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: pomeranz@ecn.purdue.edu).

Digital Object Identifier 10.1109/TVLSI.2012.2207137