ISSN 2348-375X

Unique Journal of Engineering and Advanced Sciences

Available online: www.ujconline.net

Research Article

HIGH PERFORMANCE FLASH STORAGE SYSTEM BASED ON VIRTUAL MEMORY AND WRITE BUFFER

Anbuselvi P[1*], Manjula Devi K[2]

[1]PG Scholar, Department of ECE, Nandha Engineering College, Erode-52
[2]AP, Department of ECE,Nandha Engineering College, Erode-52.

Received: 24-12-2013; Revised: 23-01-2014; Accepted: 22-02-2014

*Corresponding Author: **Anbuselvi. P,**
PG Scholar, Department of ECE, Nandha Engineering College, Erode-52 Email:anbube91@gmail.com

ABSTRACT

Many embedded and mobile device uses flash memory as a secondary storage system. Performance of flash memory depends on asymmetric speeds of read and write, limited number of erase times and the absence of in-place updates. The existing system virtual memory uses Controlled Limit Cycles (CLC) combine with Flash Aware Buffer (FAB) and Block Padding Least Recently Used (BPLRU) schemes for performance improvements. Cooperative write buffer cache (WBC) and virtual memory (VM) managements are used to improve the performance of flash memory. Management on virtual memory is designed to exploit write buffer status via reordering of the write sequences. Write buffer has been designed to improve the write performance of the flash memory. The proposed scheme flash memory incorporates VM-WBC. Partition Cluster Least Recently Used (PCLRU) technique is implemented in Write Buffer and Write Buffer Cache-Least Recently Used (WBC-LRU) technique is implemented in virtual memory. This enhanced work is implemented in very large scale integrated (VLSI) platform in the form of hardware description language (VHDL) using Xilinx tool. Thus the cooperative VM-WBC flash memory based system reduces write activities and improves the I/O performance.

Keywords: Partition cluster least recently used (PCLRU), virtual memory, write buffer, flash memory.

INTRODUCTION

Flash memory is a secondary storage system which is under the type of EEPROM (Electrically Erasable Programmable Read Only Memory). So it is used in electronics devices which share the similar characteristics, such as MP3 players, portable DVD players, digital cameras, PDAs, and even cellular [phone1-3s]. Though flash memory is still more expensive than magnetic disk yet, it has distinctive advantages such as small and lightweight form factor, solid-state reliability, low power consumption, and shock resistance. Frequent read and write operations possible in this flash memory. Write operation on Flash memory is closely related with erase operation and the endurance of the Flash memory[4]. At the same time, the speed of write is slower than that of read by eight times, on average, for Flash memory. Reducing the number of write activities on Flash memory is an efficient way to improve its performance and endurance. To improve the performance of Flash based storage systems, the write buffer has been provided in flash memories recently. At the same time, new virtual memory management strategies have been proposed in recent studies that consider the characteristics of Flash memory. Currently, approaches on these two memory layers are considered separately, which fail to explore the full potential of these two layers. In this paper, we propose cooperative management schemes for virtual memory and write buffer to maximize the performance of Flash-memory-based systems[5-9].

Flash memory is divided into two types, i.e., NOR and NAND. NOR flash memory supports the byte unit I/O and shows shorter read time and longer write time compared to NAND flash memory. It is mainly used as storage for program codes since it can be accessed by a byte unit. For such a trait, the BIOS of a computer system is usually stored on NOR flash memory. On the other side, NAND flash memory supports the page unit I/O with slower read time and faster writes time. NAND flash memory is mainly used for data storage and it is regarded as a replacement of hard disk. There are three basic operations in NAND flash memory: read, write, and erase. The read operation fetches data from a target page, while the write operation writes data to a target page. The erase operation resets all values of a target block to 1. In flash memory, once a page is written, it should be erased before it is written again, and this limitation is usually called *erase-before-write*. The read and write operations are performed by a page unit (512 Bytes or 2 Kbytes) and the erase operation is performed by a block unit (16 Kbytes or 128 Kbytes). A block consists of a fixed number of pages and a page holds a fixed number of sectors. Sector is the basic unit that can be seen by OS or other software. Although flash memory has various advantages over hard disk, it also has some limitations as storage. Among them, the followings are the important concerns in designing our buffering scheme[6].

• No in-place-update: The previous data should be erased first in order to overwrite another data in the same physical area. The worse problem is that the erase operation cannot be performed on the particular data selectively, but on the whole block containing the original data. Apparently, it is not efficient to perform costly erase operation on every data write and more sophisticated handling of write operation is required.

• Asymmetric operation latencies: For NAND flash memory, read time is faster about 8 times than write time. Because a write operation sometimes involves an erase operation, it may entail non-deterministic long delay. For this reason, it is important to reduce the number of write operations[7].

• Uneven wear-out: Each block can be erased only for a limited number of times, usually several hundreds of thousands times. Once the number is reached, the block cannot be used any more. Therefore it is necessary to distribute erase operations evenly over the whole blocks[8].

The rest of this paper is organized as follows. The background to this paper is presented in Section II. In Section III, the management technique on virtual memory is proposed. In Section IV, an approach on the write buffer is proposed to work seamlessly with the proposed approach in virtual memory. In Section V, we show the evaluation results of the proposed techniques. Finally, concluding remarks of this paper are presented in Section VI.

Background

In this paper, extended cooperating approaches for virtual memory and write buffer have the following main contributions.

1) A set of approaches is proposed for the virtual memory and the write buffer to work cooperatively. These approaches can also work independently, where the proposed approach for virtual memory can work with any kind of cluster-based write buffer management and the approach for write buffer can work with any kind of virtual-memory management approaches[9].

2) An optimization technique to improve the performance of write buffer is proposed, which makes the write buffer sensitive to the access patterns.

3) An improved dynamic window size approach is proposed for virtual memory, which makes the performance close to that of the static window size approach.

4) A new set of the evaluations of the proposed approaches is presented. With these evaluations, we are confident that the proposed approaches are effective in improving the performance of Flash-based systems. Experimental results show significant improvement in I/O performance and reduction of the number of erase and write operations compared to the state-of-art approaches. Recent work on the write buffer includes Flash-Aware Buffer (FAB), Block Padding Least Recently used (BPLRU), Controlled Limit Cycles (CLC). All of these works organize the pages in the write buffer into pages belong to the following figures. This figure shows the architectures of FAB and BPLRU with clusters organized as linked lists.

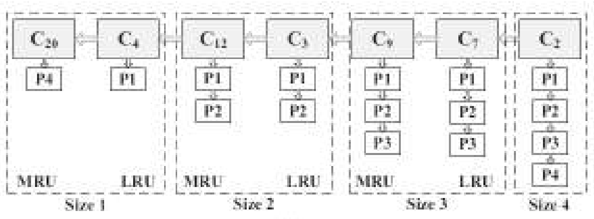

Figure 1: Data structures of FAB

In FAB, page clusters are linked on the basis of the size of the clusters. Clusters with the same size are listed in LRU order. The data structure for FAB is shown in Figure 1. During cluster replacement, FAB selects the largest cluster as the victim to evict to flash memory because evicting large clusters could minimize the additional overhead of the memory. If there is more than one cluster in the largest cluster slot, the LRU cluster is selected as the victim cluster. FAB always evicts clusters with the largest number of pages. It is suitable for sequential access applications, such as media players, which have strong spatial locality. However, FAB is not aware of the temporal locality of the access sequences. For example, if the pages in the largest cluster are updated, FAB would be inefficient, as it always evicts the largest cluster first[10].

BPLRU is another approach proposed on the write buffer to exploit access localities, and is efficient in improving the random write performance of Flash memory. BPLRU organizes the pages in the write buffer into page clusters. The clusters are linked in the LRU order. The data structures of BPLRU are shown in Figure 2.

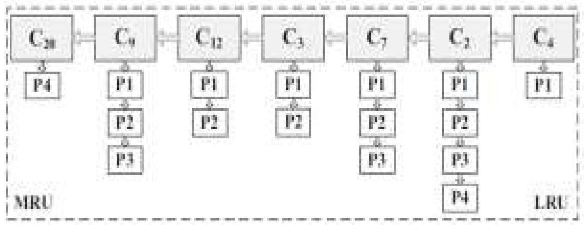

Figure 2: Data structures of BPLRU

In this approach, the spatial locality is exploited by clusters, and the temporal locality is exploited by the LRU list. However, this approach still has its shortcomings in evicting suitable clusters to the Flash memory. BPLRU could only evict the coldest clusters1 to flash memory first, while Flash memory prefers large clusters to minimize additional overhead. BPLRU applies block padding by reading out valid pages from the Flash memory and updating the whole block. This approach could induce a large amount of additional overhead. BPLRU applies block padding by reading out valid pages from the Flash memory and updating the whole block. This approach could induce a large amount of additional overhead. In order to solve this problem, CLC has been proposed. CLC proposes to combine FAB and BPLRU. It partitions the write buffer into two regions. One region is an

FAB-like region and the other is a BPLRU-like region. In CLC, the authors use 10% of clusters as BPLRU-like region and 90% as FAB-like region. Clusters could be transferred from BPLRU-like region to FAB-like region and clusters with the largest size in FAB-like region are selected as victim clusters, which is cold and large. In this paper, we propose a novel approach for the write buffer, which has a better tradeoff in the locality of write accesses and the size of victim clusters, to further improve the overall system performance by working with the virtual memory management collaboratively[11-14].

Write operations sent to the write buffer come from its upper storage layer, that is, virtual memory. If we want to reorder the write sequences sent to the write buffer, a new management approach on virtual memory should be exploited. This approach should be aware of the locality information of the write buffer and should reorder the write sequences accordingly without significantly sacrificing the performance of the virtual memory. However, because the size of main memory is limited and write operations are dependent on each other, to achieve the performance of the ideal case is not practical. Currently, Flash-memory-aware virtual memory approaches are mostly designed on the basis of the observation of asymmetric speeds of write and read operations. For example, CFLRU evicts pages following the eviction rules to delay write activities as much as possible. CFDC is an approach that takes the characteristics of FTLs into consideration. In addition to evicting the clean pages first, CFDC clusters the dirty pages in virtual memory to adapt to the characteristics of FTLs. However, these actions do not help in reordering the write sequences for the write buffer. In order to solve this problem, proposes cooperative management of the virtual memory and the write buffer. However, there are still many problems that need to be improved in[15].

1) The management approach of virtual memory needs specific information from the write buffer (the number of large clusters of the write buffer). In this case, the approach for virtual memory can only work with the proposed approach of the write buffer.

2) The management approach of write buffer is designed to work with the management of virtual memory only, without considering the performance of Flash memory.

3) The performance of the individual component is unknown to the designer since the two components are designed to work with each other.

Motivated by these problems, we present an extended version of the earlier work in this paper. Here, we first limit the dependency between the management of virtual memory and the write buffer. Second, a new access rule is added to the management of the write buffer to make it not only improve the performance but also work seamlessly with the management approach of virtual memory. Finally, the performance of these two designed approaches is evaluated.

Write-Buffer-Cache Virtual Memory Management

General architecture considered in this paper is shown in the following below figure.3.

Flash memory is equipped with a write buffer to improve its write performance. A FTL is integrated into the Flash memory controller. The Flash translation layer in this paper can be any kind of block-mapping- or hybrid-mapping based FTLs, such as BIST-aided scan test and FAST. Above the Flash memory storage system is a host system with virtual memory. Write operations write dirty pages from the virtual memory into the write buffer and flush dirty data to the Flash memory at the end. Read operations read data from both the write buffer and Flash memory. The on-chip RAM as the write buffer here can be a battery-backed DRAM or emerging nonvolatile memories .such as phase change memory and magneto resistive random access memory in order to avoid data loss when systems power is off. This paper proposes a new management approach for virtual memory, which makes use of the state information of the write buffer to reorder the write sequences. This section presents the proposed cooperating management approach on virtual memory[16].

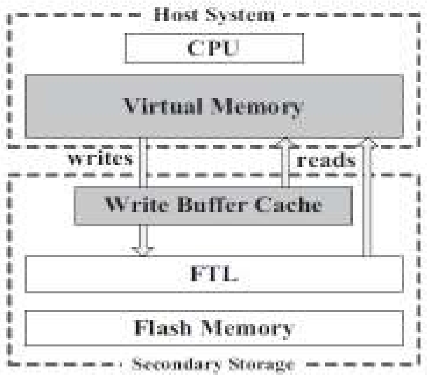

Figure 3: System Architecture.

A. Write-Buffer-Cache-Aware Virtual Memory Management

As stated in the previous sections, a write buffer has been provided in Flash-memory-based systems to improve the write performance. However, current work on virtual memory is not aware of the existence of the write buffer. Conventional managements on virtual memory consider hard disks as the storage media. The aim of these approaches is to improve the hit ratio of the virtual memory. Taking LRU as an example, it exploits the locality of applications by keeping the most recently used data in main memory. However, when the storage medium is changed to flash memory, the scenario becomes different. A flash memory has asymmetric speed of read and writes operation. This characteristic is explored by CFLRU which delays write operations in virtual memory as long as possible to reduce the number of write activities to the Flash memory. With the write buffer provided in Flash memory, CFLRU is not designed to be effective in improving the efficiency of the write buffer. As the write buffer is a very important component for high-performance Flash-based storage systems, this paper proposes a new approach for virtual memory, which is aware of the existence of the write buffer and reorders the write sequences to further improve the performance of systems. The proposed approach is *write buffer- cache LRU* (WBCLRU).

Principles of WBCLRU: WBLRU exploits the locality information in the write buffer to improve overall performance by reordering write sequences to suit the locality requirements of the write buffer. Virtual memory writes dirty pages to the write buffer when page faults take place. Different from the current work on virtual memory, WBLRU delays write back operations of dirty pages from virtual memory to the write buffer only when both the following conditions are met.

1) The selected victim dirty page does not belong to the current cluster set in the write buffer, where cluster set is the set including all of the clusters in the write buffer.

2) No new cluster is allowed to be created in the write buffer.

Partition Cluster Least Recently Used Management for Write Buffer

Several management techniques on the write buffer have been proposed. The most recent work includes FAB, BPLRU, and CLC. Among these approaches, CLC is a hybrid approach between BPLRU and FAB, and is the most promising approach for write buffer management. CLC partitions the write buffer into two regions. During cluster eviction, CLC selects the largest cold cluster as the victim cluster from the FAB-like region. When a cluster is accessed in CLC, the cluster will be moved to the most recently used (MRU) position of the whole cluster list. CLC can efficiently explore the temporal and spatial locality of write sequences when LRU or CFLRU is applied in virtual memory. However, when WBLRU is applied in virtual memory, there are several potential problems[17].

1) Both FAB and CLC take the size of clusters into consideration. However, when WBLRU is applied as virtual memory management, temporal and spatial locality is more important. Size-sensitive approaches are not able to fully exploit the locality properties.

2) BPLRU takes temporal and spatial locality as the most important factor. However, as discussed in CLC, BPLRU cannot select the large and cold cluster to evict.

3) CLC partitions the write buffer into two fixed regions, which are not adaptive to the applications characteristics. For example, if the clusters in the write buffer are mostly cold, CLC would not be able to select cold and the largest cluster to evict.

Based on these observations, this paper proposes a new write buffer management technique, *partition cluster LRU* (PCLRU), to work seamlessly with WBCLRU.

1) Principles of PCLRU: Following the typical write buffer organization, PCLRU organizes cache lines as clusters. An example of the organization of the write buffer is shown in Figure. 4.

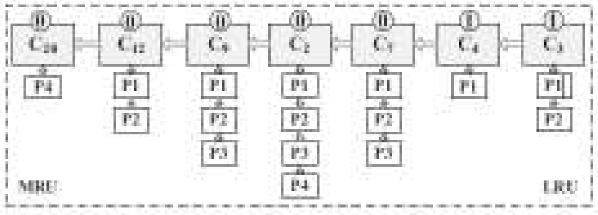

Figure 4: Organization of the write buffer.

The cache line size is set to be the page size of the Flash memory. All the cache lines from the same logical block are grouped into one cluster. All the clusters are organized as a linked list. The design principle is that PCLRU should exploit locality as much as possible and evict clusters as large as possible at the same time. In the rest of this section, we discuss the access rules for exploiting locality and eviction rules for clusters.

2) Access Rules of PCLRU: Different from the LRU list in previous work, PCLRU differentiates small clusters and large clusters. In PCLRU, clusters in the write buffer are logically partitioned into a *large cluster set* (LCS) and a *small cluster set* (SCS) based on the size of the clusters. An example of cluster 6 partition is shown in Figure 4, where $CS = \{C2, C4, C7, C3, C12, C9, C20\}$, $LCS = \{C2, C7, C9\}$, and $SCS = \{C4, C3, C12, C20\}$. This example assumes that the *threshold of large cluster*, *TLC*, equals 3. Different access rules are designed for different sets of clusters.

1) For a small cluster, if it is accessed, it is moved to the MRU position to capture future cluster hits.

2) For a large cluster, it is not moved to the MRU position of the cluster list when it is accessed to avoid slow retirement of large clusters. The accessed large cluster is moved to a position right before the first large cluster in the cluster list.

Based on these two write buffer access rules, PCLRU has better tradeoffs in exploring the locality of write accesses and size of victim clusters as all clusters are accessed in LRU order and large size clusters are more likely moved to the LRU position. Figure 5 shows the processes of cluster updates with the assumption of $TLC = 3$. For case 1, $C7$, which is a large cluster, is inserted before the first large cluster $C9$ when it is accessed. For case 2, if $C4$, which is a small cluster, is accessed, it is inserted at the MRU position. With this approach, large clusters can be retired more quickly. Because the linked list is organized in the LRU order, cold large clusters are more likely to be moved to the LRU position, so the proposed approach always selects the large cold clusters to evict.

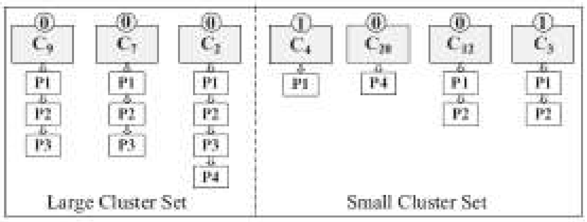

Figure 5: Logical partition of write buffer based on the size of clusters. The threshold of the large cluster TLC is assumed to be equal to 3.

TLC is a very important parameter in PCLRU, as it determines how the clusters are accessed. PCLRU changes *TLC* adaptively on the basis of the state of the write buffer. If there are too many large clusters in the write buffer, *TLC* is increased to avoid too many clusters being transferred to the LCS. If there is not enough number of large clusters, *TLC* is reduced to allow more large clusters.

3) Eviction Rules of PCLRU: When the write buffer is full and a new page is written to the write buffer, PCLRU selects a cluster to evict. Most of recent work, such as BPLRU and CLC, organizes the write buffer as cluster lists. BPLRU selects the LRU cluster to evict and CLC selects the largest cluster in the FAB-like region. Different from these approaches, we take both the size of clusters and hot/cold of clusters into consideration. The aim of the proposed eviction

rules is to evict the cold and large clusters first by delaying the eviction of small clusters. We propose eviction rules for PCLRU as follows. PCLRU selects the victim cluster from the LRU position.
1) If the selected cluster belongs to the LCS, the selected cluster is flushed to the Flash memory right away.
2) If the selected cluster is a small cluster, it is kept in the write buffer and given chances to grow large.
3) If the selected small cluster has been given a number of predefined chances, it is flushed to the Flash memory.
Details on the eviction of clusters from the write buffer can be found in. Compared with the work in, PCLRU is proposed to not only work seamlessly with the WBCLRU but also be sensitive to the access patterns. In the experimental section, a detailed explanation of the results is presented to show the significance of the PCLRU.

Performance Evaluation

In order to show the performance of Flash memory, we compare the number of write and erase operations and I/O performance of Flash memory. The size of the virtual memory and write buffer has different effects on the performance of Flash memory. For a large-sized virtual memory, read performance can be much better, while for a large-sized write buffer, write performance can be improved. In this paper, a 16-MB virtual memory is applied as the main memory. The size of the write buffer is varied from 1–7 MB to show the results.
We do not present the performance impact on the size of the virtual memory, which can be found in CFLRU. With the increase in the size of the virtual memory, the read and write performance is increased since more reads and writes are hit in the memory. The number of erase writes is reduced for increasing size of the buffer. This is shown in the figure 6 for different set of values.

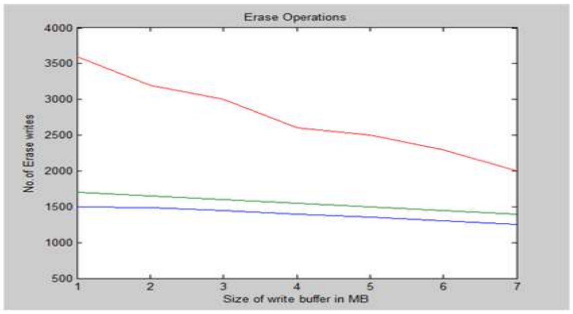

Figure 6: Performance Analysis for three different set of values.

CONCLUSION

This paper proposed cooperative virtual memory and write buffer management in Flash-based systems. This approach designed to improve the performance of Flash-memory-based systems. Unlike previous work, where virtual memory and write buffer managements are designed separately, this paper proposed a new replacement algorithm for virtual memory, which cooperates with the write buffer and reorders the write sequences sent to the write buffer. A replacement algorithm on the write buffer was also proposed to work seamlessly with the proposed approach in virtual memory. Simulation results shows that significant improvement in I/O performance and reduction in the number of erase and write operations could be obtained compared with the state-of-art approaches.

REFERENCES

1. Ahn S, Kim H. BPLRU: A buffer management scheme for improving random writes in flash storage in Proc. 6th USENIX Conf.File Storage Technol. 2008; 1.
2. Chung TS, Lee SW, Park DJ, Lee DH, Park S, Song HJ. A log buffer-based flash translation layer using fully-associative sector translation. ACM Trans. Embed. Comput. Syst. 2007; 6(3): 1.
3. Hu XY, Eleftheriou E, Haas R, Iliadis I, Pletka R. Write amplification analysis in flash-based solid state drives in Proc, SYSTOR.Israeli Experim. Syst. Conf. 2007; 19.
4. Huang Y, Liu T, Xue CJ. Register allocation for write activity minimization on non-volatile main memory for embedded systems. J.Syst. Arch. 2012; 58 (1): 13.
5. Jung H, Shim H, Park S, Kang S, Cha J. LRU-WSR: Integration of LRU and writes sequence reordering for flash memory, IEEE Trans.Consumer Electron. 2008; 54(3): 1215.
6. Jo H, Kang JU, Park SY, Kim JS, Lee J. FAB: Flash-aware buffer management policy for portable media players, IEEE Trans.Consumer Electron. 2006; 52(2): 485.
7. Kang S, Park S, Jung H, Shim H, Cha .J. Performance trade-offs in using NVRAM write buffer for flash memory-based storage devices. IEEE Trans. Comput. 2009; 58(6): 744.
8. Kim J, Kim JM, Noh S, Min SL, Cho Y. A space-efficient flash translation layer for compact flash systems, IEEE Trans. ConsumerElectron. 2002; 48(2): 366.
9. Li z, Jin P, Su X, Cui K, Yue L. CCF-LRU: A new buffer replacement algorithm for flash memory, IEEE Trans. Consumer Electron. 2009; 55(3): 1351.
10. Ou Y, Harder T, Jin P. CFDC: A flash-aware replacement policy for database buffer management, in Proc. 5th Int. Workshop DataManage. New Hardw. 2009; 15.
11. Park SY, Jung D, Kang JU, Kim JS, Lee J. CFLRU: A replacement algorithm for flash memory, in Proc. Int. Conf. Compil., Arch. Synth. Embed. Syst. 2006; 234.
12. Qin Z, Wang Y, Liu D, Shao Z, Guan Y. MNFTL: An efficient flash translation layer for MLC NAND flash memory storage systems, in Proc. 48th Design Autom. Conf. 2011; 17.
13. Shi L, Li J, Xue CJ, Yang C, Zhou X. EXLRU: A unified write buffer cache management for flash memory, in Proc. ACM Int.Conf. Embed. Softw. 2011; 339.
14. Shi L, Xue CJ, Hu J, Tseng WC, Zhou X, Sha EH. Write activity reduction on flash main memory via

smart victim cache, in Proc. 20th Symp. Great Lakes Very Large Scale Integr. Syst. 2010; 91.

15. Seo D, Shin D. Recently-evicted-first buffer replacement policy for flash storage devices, IEEE Trans. Consumer Electron. 2008; 54(3): 1228.
16. Seol J, Shim H, Kim J, Maeng S. A buffer replacement algorithm exploiting multi-chip parallelism in solid state disks, in Proc. Int. Conf. Compil., Arch., Synth. Embed. Syst. 2009; 137.
17. Wang Y, Liu D, Qin Z, Shao Z. An endurance-enhanced flash translation layer via reuse for NAND flash memory storage systems, in Proc. Design, Autom. Test Eur. Conf. Exhibit. 2011; 1.

Source of support: Nil, Conflict of interest: None Declared