

Energy and Performance Optimization of Demand Paging With OneNAND Flash

Yongsoo Joo, *Member, IEEE*, Yongseok Choi, *Member, IEEE*, Jaehyun Park, *Student Member, IEEE*, Chanik Park, Sung Woo Chung, *Member, IEEE*, Eui-Young Chung, *Member, IEEE*, and Naehyuck Chang, *Senior Member, IEEE*

Abstract—New fusion memory devices consisting of multiple heterogeneous memory components in a single die or package offer efficient ways to optimize embedded systems in terms of energy, performance, and cost. Samsung Electronics recently announced the OneNAND fusion memory, in which a NAND flash array is integrated with dual SRAM buffers to provide a NOR-type I/O interface. OneNAND has the low cost and large capacity of a NAND flash but also permits eXecution-in-Place (XIP) like a NOR flash. The deployment of such devices requires careful system-level resource management because of their impact on energy consumption and performance, and existing memory optimization techniques, such as the demand paging used with NAND flash, may no longer be appropriate for systems with a fusion memory. We introduce a new online demand paging scheme that fully exploits the XIP capability of OneNAND flash by classifying pages as load preferred (residing in the on-chip SRAM) and XIP preferred (accessed directly from the OneNAND flash and discarded after use). This achieves, on average, a 26% reduction in energy consumption and a 19% increase in performance, compared with conventional NAND flash demand paging.

Index Terms—Demand paging, eXecute-In-Place (XIP), flash memory, OneNAND, page allocation, page replacement.

I. INTRODUCTION

THE ULTIMATE memory device for portable applications should offer balanced read and write performance, low cost, large capacity, low power consumption, and nonvolatility. In portable applications, read and write performance and nonvolatility are equally important in devices that aim to replace magnetic disks with their heavy and fragile mechanical components. Such a memory device would allow a system to have a single memory component. Unfortunately, after decades of

Manuscript received December 24, 2007; revised April 8, 2008. Current version published October 22, 2008. This work was supported by Samsung Electronics. An earlier version of this paper was presented at the International Conference on Hardware/Software Codesign and System Synthesis 2006 [1]. This paper was recommended by Associate Editor M. Poncino.

Y. Joo, J. Park, and N. Chang are with the Department of Electrical Engineering and Computer Science, Seoul National University, Seoul 151-742, Korea (e-mail: ysjoo@elpl.snu.ac.kr; jhpark@elpl.snu.ac.kr; naehyuck@elpl.snu.ac.kr).

Y. Choi is with the Digital Media R&D Center, Samsung Electronics, Suwon 442-742, Korea (e-mail: yongseok07.choi@samsung.com).

C. Park is with the Memory Division, Samsung Electronics, Hwaseong 445-701, Korea (e-mail: ci.park@samsung.com).

S. W. Chung is with the Division of Computer and Communication Engineering, Korea University, Seoul 136-713, Korea (e-mail: swchung@korea.ac.kr).

E.-Y. Chung is with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea (e-mail: eychung@yonsei.ac.kr).

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/TCAD.2008.2006081

semiconductor evolution, such a device is not yet available, and the current practical solution is to combine a fast but volatile RAM with a nonvolatile user-programmable memory such as a flash memory. Next-generation memories such as phase-change RAM and ferroelectric RAM may take us closer to a single-memory system, but they are not in mass production stages yet, and their performance cannot meet that of current RAMs.

Commercial portable embedded systems are highly optimized in terms of cost, and thus, both the die size of the chips and the chip count are crucial. Recently, systems-on-a-chip (SoCs), equipped with a microprocessor core, cache memory, SRAM, and even flash memory and peripherals, have been widely used to realize small systems cost-effectively.¹ The on-chip SRAM size directly affects the die size and thus the cost of an SoC, making optimization of the SRAM footprint a significant problem, which is exacerbated by the increasingly capable application programs demanded by users and by functionality such as digital rights management (DRM), which is now an important part of marketing and service strategies. Moreover, the requirement for unified services from high-end to low-end devices means that even entry-level systems have to support a full range of features.

Flash memory is cheaper than SRAM, but its extremely bad write performance restricts its use to read-only applications or those in which few writes are required. The applications of flash memory are effectively orthogonal to those of volatile RAMs, and the devices are not interchangeable. However, appropriate use of flash memory can mitigate the demand for SRAM. There are two types of flash memories: NOR and NAND. The NOR flash allows random access and therefore supports direct code execution (XIP: eXecute-In-Place), which can significantly reduce the requirement for on-chip SRAM without additional optimization. Nevertheless, modern designs for cost-effective portable systems often use cheap NAND flash to store large programs and data. However, NAND flash only allows page access, so that programs stored in NAND flash must be uploaded to the on-chip SRAM before execution, and XIP is not possible. Code shadowing, in which all the pages required by the program are uploaded to the on-chip SRAM in advance, is commonly used to mitigate this problem, but this does not help reduce the size of the on-chip SRAM. An alternative to

¹We do not consider off-chip SDRAMs, which are not suitable for low-cost lightweight systems because of significant extra power consumption and system complexity. The use of off-chip SDRAMs also raises other issues such as signal integrity, which eventually result in a noticeable increase in cost.

code shadowing is demand paging, which does not involve uploading all the code pages in advance. Instead, each page of a program is uploaded on demand, and this can dramatically reduce the on-chip SRAM size. However, the associated page fault penalty is so severe that demand paging cannot be used without a sophisticated method of handling page faults, unless the size of the working set is smaller than that of the on-chip SRAM.

One promising alternative to conventional flash memory is currently being developed by semiconductor vendors. The fusion memory is a combination of multiple heterogeneous memories on a single die or chip. For example, Samsung Electronics has announced a new fusion flash memory called OneNAND [2] that combines the advantages of a NOR flash and a NAND flash. Instead of a sequential interface, OneNAND has random-access dual SRAM buffers. This random-access capability offers great potential to enhance demand paging by the use of XIP.

The use of a fusion memory requires careful consideration because it changes not only the cost of an embedded system but also its energy consumption and performance. However, no in-depth system-wide analysis of the pros and cons of fusion memories has yet been attempted. The issues are complicated: For instance, conventional NAND flash demand paging cannot utilize the XIP capability of the OneNAND flash, and the use of its SRAM buffers for XIP is a challenge because there are only two of them and they are small; all the data coming from the NAND array must pass through an SRAM buffer.

In this paper, we introduce a new approach to demand paging that fully exploits the XIP capability of the OneNAND flash by selectively loading codes and data to the on-chip SRAM. Infrequently used codes and data are not loaded into the on-chip SRAM but directly accessed from the SRAM buffer of the OneNAND flash. Since codes and data kept in the SRAM buffer are soon overridden by the next page demand, an elaborate management technique is required to avoid an excessive number of page faults. Our approach to demand paging emphasizes the optimality of page allocation as well as of page replacement. We have performed extensive simulations which demonstrate that this method of demand paging has the potential to improve the energy consumption and performance of systems incorporating OneNAND flash. Results show an average improvement over conventional demand paging of 26% in terms of energy consumption and a 19% reduction in execution time.

II. RELATED WORK

A. Demand Paging Optimization of Flash Memory

A systematic compiler-assisted approach to demand paging for flash memory has recently been introduced [3]. In addition, a code overlay technique based on synchronous data flow has been proposed for low-end embedded systems that use NAND flash [4]. There is also a subpaging technique for NAND flash systems which tries to reduce the number of unnecessary write operations by only flushing away dirty subpages [5].

Along with the performance and cost optimization of the NAND demand paging, energy-aware demand paging has also become an important topic. New page replacement policies for

NAND demand paging, such as clean-first least recently used (LRU) [6], are based on accurate energy models of the paging system.

Currently, more substantial modifications of the system architecture are being attempted. Strong demand for further optimization of the memory subsystems of embedded applications motivated the development of the fusion memory, which consists of heterogeneous memory devices, such as SRAM and NAND flash, on the same die or package [2], [7]. A fusion memory can replace a number of legacy memory components, which is an effective way of reducing the chip count. Switching to a fusion memory may be expected to impact the performance of the system due to changes in the data paths, I/O capacitance, etc. However, there has not yet been a full analysis of the impact of using fusion memory on system energy consumption and performance. In addition, existing memory optimization methods, such as the demand paging algorithm used for NAND flash, are not optimized for systems with a fusion memory.

B. Memory System Optimization Based on Data Reuse Analysis

The method of demand paging for a OneNAND flash that we propose is closely related to the problem of data reuse, which has been addressed by many researchers [8]–[10], most of whom have used profile-based offline analysis to determine an allocation of memory objects, which then remains fixed during program execution. However, more recent work [11], [12] has sought to overcome the limitation of static schemes by changing the allocation dynamically at runtime.

Previous approaches are similar to our scheme in the sense that the target memory system is multilayered and the memory allocation policy plays an important role in energy and performance optimization. However, our scheme is differentiated from earlier research because we use the OneNAND buffers and XIP to change the memory hierarchy at runtime, whereas it is fixed in previous approaches. Depending on the pattern of accesses to a page, the OneNAND buffers may function as a page buffer for transferring data from a secondary storage to the main memory or as the main memory itself.

C. Page Replacement Policies

Conventional demand paging assumes a two-level memory hierarchy that consists of a main memory and a secondary memory. Since the secondary memory generally does not support random access, all pages are assumed to be accessed from the main memory, and the energy consumption and performance of conventional demand paging are primarily determined by the page replacement policy. Belady's MIN [13], which replaces the page that will be used farthest in the future, is known to be the optimal offline solution. However, MIN requires advance knowledge of the page request sequence, which is not available in real systems. Over several decades, a lot of effort has been put into the design of online algorithms with the aim of matching the performance of MIN without being able to predict the future.

The LRU, first-in/first-out (FIFO), and least frequently used (LFU) schemes are the oldest online page replacement policies. LRU and FIFO are known to be the most efficient online replacement algorithms, when assessed by competitive analysis [14]. However, the performance of FIFO is much lower than that of LRU for most real page request sequences. LRU is the optimal policy under the LRU stack depth distribution model, but it shows poor performance for sequential or cyclic page access patterns. LFU cannot adapt to nonstationary page request sequences since it does not consider their recency at all.

There have been many attempts to mitigate the disadvantages of these policies, such as LRU-K [15], least recently/frequently used [16], the low interference recency set [17], and the adaptive replacement cache [18]. However, most of them require complicated operations to take place at every memory reference, which makes their maintenance cost too high for practical use. For this reason, CLOCK [19], which is an approximate version of LRU, and its variants are still used in most operating systems, including MVS, Unix, Linux, and Windows, due to their low cost and good performance.

D. Page Allocation Policies

Our method of demand paging with a OneNAND flash requires not only a page replacement policy but also a page allocation policy to decide whether to allocate the demanded page to the on-chip SRAM or to the SRAM buffer of the OneNAND flash. However, there has been less research on page allocation, which is generally not required in conventional demand paging, because all the missed pages are allocated to the main memory.

The most recently frequently used (MRFU) policy [20] has lately been suggested for multilevel cache management in storage-area networks (SANs). MRFU automatically migrates the most frequently used data chunks to a separate SAN cache, such as a RAM disk, without any intervention by the SAN administrator. MRFU moves these “hot” chunks from the disk array to the SAN cache by tracking the access count of each chunk. To do this, MRFU maintains a priority queue that consists of a set of counters which records the accumulated number of accesses to each chunk. In addition, MRFU has a “cooling” mechanism that periodically decrements the page access counters to avoid the cache pollution effect; this approach is also used in LFU with dynamic aging (LFUDA) [21]. The cooling mechanism requires two parameters, which should be tuned to suit the access pattern: They are called the cooling period and the amount of cooling per period. Although MRFU has a similar structure to LFUDA, these two policies select the most and LFU data chunks in different ways. While it is adequate for LFUDA to select the LFU chunk from the priority queue, MRFU needs to determine a set of frequently used chunks. Thus, MRFU requires an additional threshold parameter. A data chunk is classified as frequently used if its access count exceeds this threshold. Since the optimal threshold value will be affected by a change of workload, a threshold adaptation algorithm is beneficial [20].

Although the issue of data migration in a multilevel SAN cache is similar to our problem, MRFU is not suitable for demand paging with OneNAND flash for two reasons.

- 1) The page size in demand paging with a OneNAND flash is orders of magnitude smaller than the data chunk size in SAN applications. This increases the frequency with which the priority queue must be updated, significantly compromising the performance gain from using MRFU.
- 2) We found experimentally that the threshold adaptation algorithm of MRFU fails to track changes to the page access pattern in the context of our method of demand paging. The adaptation algorithm of MRFU [20] works by periodically incrementing or decrementing the threshold by one. Although it works well for a workload that changes slowly (over tens of minutes), in our environment, the page access pattern changes in a matter of microseconds and keeps changing, which makes it impossible to use this type of algorithm.

In contrast to MRFU, our page allocation policy has a constant time complexity, which is why it is suitable for demand paging with OneNAND flash. We also propose a threshold configuration scheme which can cope effectively with drastic changes of page access pattern.

III. BACKGROUND

A. Demand Paging for Portable Embedded Systems

Flash memories can be classified into two types, NOR and NAND, according to their structure and access method. Each type has its own advantages and disadvantages [22]. While NOR flash supports random access at relatively high cost per unit capacity, NAND flash provides higher data density at a lower cost per unit capacity but only supports page access. A host, generally a microprocessor or a direct memory access (DMA) controller, accesses the data in a NAND flash array through the page buffer, so that the whole page has to be uploaded even if only a single word is required.

There are three popular ways to enable program execution from a flash memory, which are shown in Fig. 1. The simplest method [shown in Fig. 1(a)] is to use a NOR flash, which supports random access, and then to perform XIP directly from the flash memory. This is easy to implement and makes a useful saving in on-chip SRAM because it is not involved. The only, and obvious, drawback is the high cost of the NOR flash, which is certainly not economic for large amounts of user data. The addition of a NAND would serve this purpose, but this runs counter to modern trends, which favor the use of a single type of flash memory in a system. As the amount of user data in multimedia applications becomes larger and larger, this design constraint suggests a NAND-only configuration.

Since a NAND flash does not support XIP, codes must be copied to the main memory² before execution. This can be achieved by code shadowing, in which all the codes are copied

²The main memory might be an off-chip SDRAM, but we are focusing on lightweight portable embedded systems and will assume that there is only an on-chip SRAM.

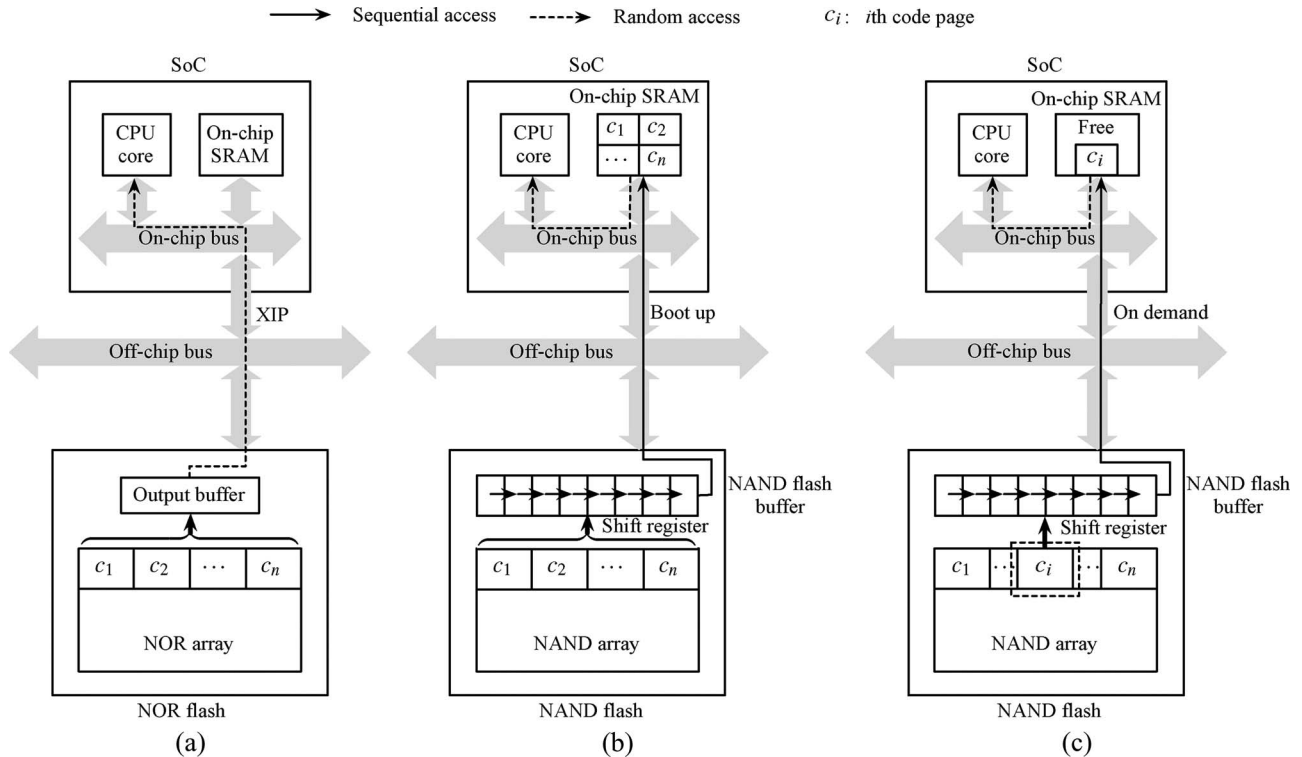


Fig. 1. Program execution from flash memory. (a) XIP: Directly execute codes from the flash memory. (b) Code shadowing: Copy the whole codes when boot up. (c) Demand paging: Copy a missed page on demand.

from the NAND flash to the on-chip SRAM during system boot [Fig. 1(b)]. Code shadowing provides good runtime performance, owing to the short access time of the on-chip SRAM. Indeed, in some high-performance systems, the codes in a NOR flash are shadowed, just because NOR is slower than SRAM. The primary disadvantage of code shadowing is that it requires extra SRAM to accommodate all the codes. In addition, the NAND flash is a wasted resource during execution, when it is not used at all.

Demand paging overcomes these problems by copying parts of the codes, called code pages, to the on-chip SRAM one by one, when they are required [Fig. 1(c)]. This makes it possible to execute a program that is larger than the on-chip SRAM, and thus, the size of the on-chip SRAM can be dramatically reduced compared with code shadowing. However, demand paging involves extra latency, due to the page fault that occurs every time that a required code page is not available from the on-chip SRAM. This latency is not predictable, making demand paging unsuitable for real-time tasks. Therefore, in practical portable embedded systems, time-critical tasks are typically pinned on the on-chip SRAM by code shadowing, and the rest of the on-chip SRAM, which is called a page cache, is used for demand paging of non-time-critical code such as DRM or the user interface. Note that demand paging of read-only pages is typically assumed in portable embedded systems [23], [24]. Demand paging of read-write pages is not generally adopted in such systems because it accompanies with frequent write operations to the flash memory, which significantly increase the page fault latency and its energy consumption. It also requires sophisticated wear-leveling and garbage collection mechanisms [25], which can be a burden for lightweight portable systems.

The on-chip SRAM size is critical in making demand paging effective. A small SRAM causes an unbearable number of page faults, while an SRAM that is too big wastes money. The size of the on-chip SRAM is necessarily determined during the design of an SoC, and any subsequent increase requires refabrication of the whole chip. However, there are likely to be many software revisions during the life of an SoC. Demand paging is usually able to accommodate the increase in code size that accompanies most software revisions, but if a revision results in any significant growth of the size of the working set, then even demand paging can be defeated. The method of demand paging that we propose for OneNAND flash can cope much better with increases in the size of working set caused by code revisions, as well as having lower energy consumption and reduced latency.

B. OneNAND Flash

Semiconductor vendors have recently announced fusion flash memories such as OneNAND [2] and M-DOC [7], which have an SRAM interface and randomly accessible buffers that allow them to execute startup code directly, without using a NOR flash or a mask ROM. Fig. 2 shows the internal structure of Samsung's OneNAND flash, which has three SRAM buffers through which all read and write operations to the NAND flash array must pass. One of these three buffers is dedicated to the boot loader, and we will only be concerned with the other two buffers which are used for normal data transfers.

This new architecture allows OneNAND flash to combine high density of data and improved I/O performance with random access to its page buffers. This random-access capability is designed to eliminate the use of nonvolatile random accessible

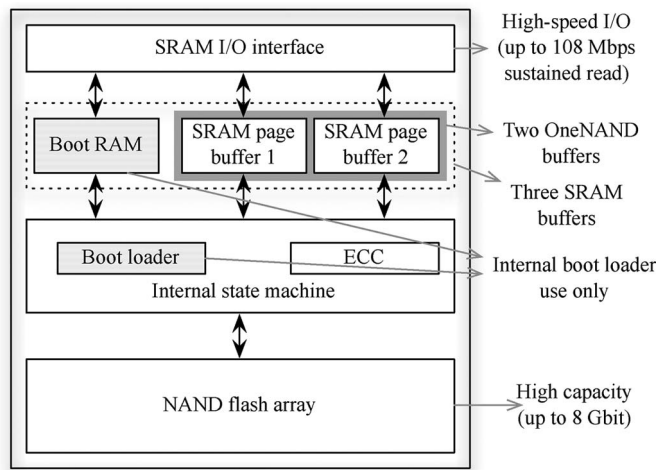


Fig. 2. Internal structure of the OneNAND flash.

storage, such as a NOR flash, during a boot-up sequence. Power-on reset automatically causes the initial boot code in the NAND flash array to be loaded into the OneNAND buffer, and this activates the uploading of the main boot loader to the on-chip SRAM without the help of a NOR flash. Another design feature of OneNAND is a buffer-level interleaving scheme, called dual buffering, for successive page transfers: The transfer of a page from the flash array to the OneNAND buffer can overlap with the transfer of another page from the OneNAND buffer to the on-chip SRAM. Dual buffering can significantly improve the throughput during data transfer.

Our proposal is to use XIP from the OneNAND buffer not only during the boot-up sequence but also during normal program execution. In the following sections, we will show how we exploit the random-access capability of the OneNAND page buffers to optimize the energy consumption and performance of demand paging.

IV. DEMAND PAGING WITH ONENAND FLASH

A. Conventional Demand Paging

Owing to its synchronous interface and support of dual buffering, simply replacing a NAND flash with a OneNAND flash can improve conventional demand paging. However, this does not exploit the random-access capability of the OneNAND buffer. As shown in Fig. 3(a), following a page fault, a new page is first loaded from the OneNAND buffer to the on-chip SRAM through sequential transfers, and then, it is always accessed from the on-chip SRAM until it is replaced.

The energy consumption and performance of conventional demand paging are solely determined by the page replacement policy because that is the only decision that can be made when a page fault occurs.

The number of page faults is a metric that is commonly used to evaluate the energy consumption and performance of a page replacement policy, rather than the total energy consumption or the total execution time. This is because minimizing the number of page faults always guarantees the best performance of conventional demand paging in terms of both energy consumption and execution time.

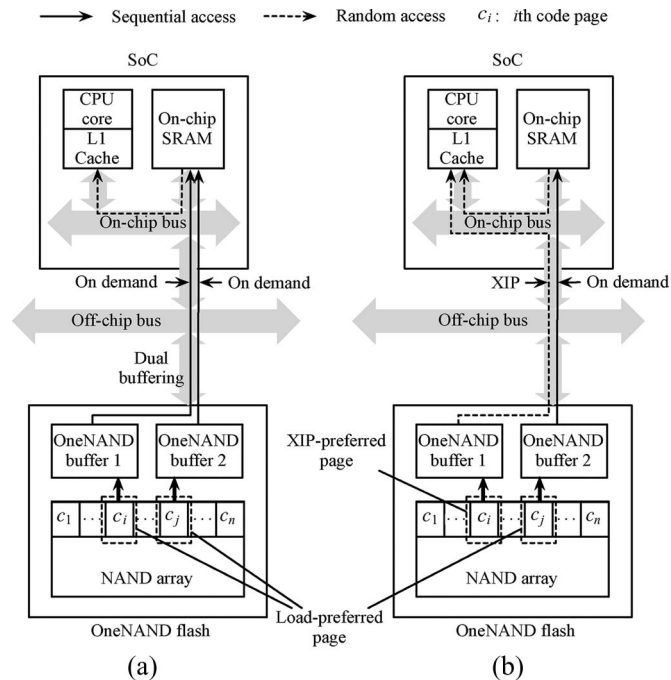


Fig. 3. Overall structure of demand paging with a NAND flash. (a) Conventional demand paging with a OneNAND flash. (b) Proposed demand paging with a OneNAND flash.

B. Proposed Demand Paging

The random-access feature of the OneNAND flash makes XIP feasible, allowing a code page to be executed directly from the OneNAND buffers. However, in most cases, a complete code block cannot be stored in the OneNAND buffers since they are quite small in the context of modern application programs. Moreover, a new page is always loaded into the OneNAND buffers, even if it will eventually be allocated to the on-chip SRAM: This forces the existing page in the OneNAND buffers to be evicted at every page fault. Thus, a page that is allocated to the OneNAND buffers can be expected to have a short lifetime. This limitation on the XIP feature of the OneNAND flash suggests selective upload of pages to the on-chip SRAM, taking account of their expected lifetime and utilization. Fig. 3(b) shows the proposed demand paging with a OneNAND flash. An infrequently used code page is executed from the OneNAND buffer, without ever being loaded into the on-chip SRAM, and is discarded after use [c_i in Fig. 3(b)]. However, a frequently used page is uploaded to the on-chip SRAM for current and future use [c_j in Fig. 3(b)]. We will call these *XIP-preferred pages* and *load-preferred pages*. Our demand paging policy allows the number of page faults to be significantly reduced without increasing the size of the on-chip SRAM. Selective upload also reduces the cost of loading a page from the OneNAND flash to the on-chip SRAM.

When a page fault occurs, our demand paging scheme requires page allocation and page replacement decisions to be made. A page allocation policy determines whether to upload the new page to the on-chip SRAM or to keep it in the OneNAND buffer. After boot-up, the on-chip SRAM is soon completely filled with load-preferred pages, at which point the acceptance of a new page requires a page replacement policy,

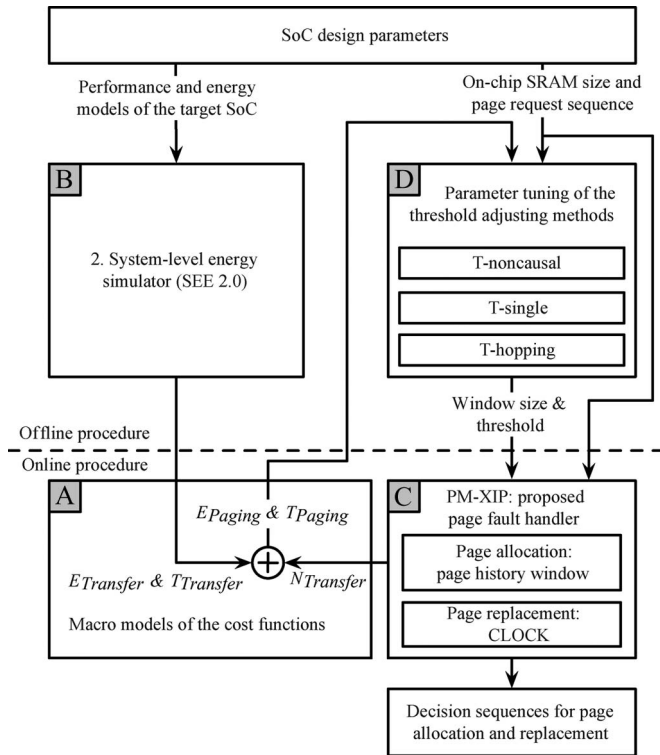


Fig. 4. Framework of the suggested solution methods.

which discards the least useful code page from the on-chip SRAM. However, minimizing the number of page faults does not guarantee optimal energy consumption and performance, because allocating and accessing a load-preferred page and an XIP-preferred page have different time and energy costs. Consequently, the energy consumption and performance of our demand paging policy is affected not only by the number of page faults but also by the proportion of load-preferred pages and of XIP-preferred pages.

C. Problem Statement

The problem of optimizing demand paging with a OneNAND flash can be stated as follows.

- 1) Derive cost functions which express the energy consumption and performance of demand paging: E_{Paging} is its energy consumption and T_{Paging} is its execution time.
- 2) Suggest a page fault handler for OneNAND demand paging, consisting of a page allocation policy and a page replacement policy, to minimize E_{Paging} and T_{Paging} for a given sequence of page requests.

V. OPTIMIZATION OF PROPOSED DEMAND PAGING

In this section, we suggest a set of solution methods to optimize the proposed OneNAND demand paging, as shown in Fig. 4. First, we construct macromodels to derive the cost functions E_{Paging} and T_{Paging} . Second, we develop a system-level energy simulator and a paging system simulator to determine the values of the derived macromodels. Third, we suggest a

new page fault handler, PM-XIP, which is composed of a page history window as a page allocation policy and CLOCK as a page replacement policy. Fourth, we suggest three threshold adjusting methods, namely, T-noncausal, T-single, and T-hopping, to configure the page history window. The following four sections describe the detail of each solution method.

A. Derivation of the Cost Functions

The cost functions E_{Paging} and T_{Paging} should be precise enough to give a correct indication of the energy consumption and performance of the proposed demand paging. At the same time, they should be able to be evaluated quickly, because repeated simulations are required to develop a new paging policy.

Constructing the cost functions requires an analytical model which can be evaluated rapidly. Simple analytical models of memory devices can easily be constructed, because most memory device vendors provide a datasheet which gives the timing specification and average energy consumption of each memory operation. However, a precise analytical model of detailed bus activities, including delays for arbitration and transaction initialization, is hardly feasible.

A cycle-accurate system-level simulator would overcome this difficulty, as it can precisely simulate the cycle-by-cycle behavior of the memory system, including the bus arbiter and the DMA controller. However, this approach is orders of magnitude slower than an analytical model, and thus, it is not suitable for the development of a new paging policy. We compromise by constructing macromodels of E_{Paging} and T_{Paging} using a cycle-accurate system-level simulator and a trace-driven paging system simulator.

To obtain simple formulations of E_{Paging} and T_{Paging} , we will initially focus on portable embedded systems, assuming the following conditions.

- 1) The target system is made up of a CPU core with a level-1 I-cache and a D-cache, an on-chip SRAM main memory, and a OneNAND flash secondary memory [Fig. 3(b)].
- 2) Only the code pages are subject to demand paging, so that flushing dirty data pages to the OneNAND flash is not considered.
- 3) The logical page size is equal to the size of the OneNAND buffer, so that an instruction code page will exactly fit into the OneNAND buffer.
- 4) Migration of a page is performed only when a page fault occurs.

Under the aforementioned assumptions, we define four types of memory transfer, as shown in Table I. We will denote the number of occurrences of each transfer during program execution as N_{Transfer} . The energy consumption of each transfer is E_{Transfer} , and its execution time is T_{Transfer} . Fig. 5 shows the timing sequence for accessing an XIP-preferred page c_i and a load-preferred page c_j : First, c_i is requested and a page fault occurs, so the page fault handler initiates *Flash2Buf* for c_i and allocates it to the OneNAND buffer. After *Flash2Buf* has completed the execution, c_i is available in the OneNAND buffer, and the CPU accesses it directly through *Buf_Read*. When another page fault is caused by c_j , the page fault handler

TABLE I
NOTATION FOR DATA TRANSFERS IN OneNAND DEMAND PAGING

Transfer	Description
<i>Flash2Buf</i>	Page transfer from the flash array to the OneNAND buffer
<i>Buf2SRAM</i>	Page transfer from the OneNAND buffer to the on-chip SRAM
<i>SRAM_Read</i>	Word read from the on-chip SRAM
<i>Buf_Read</i>	Word read from the OneNAND buffer

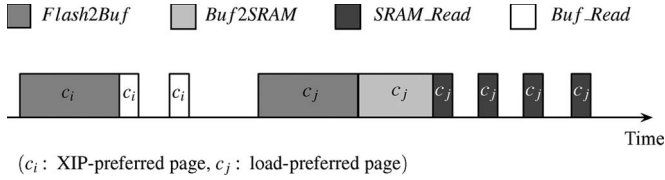


Fig. 5. Timing sequence for accessing an XIP-preferred page and a load-preferred page.

initiates *Flash2Buf* for c_j , but this time, it allocates c_j to the on-chip SRAM. When this run of *Flash2Buf* is complete, the page fault handler calls *Buf2SRAM* to move c_j to the on-chip SRAM. When *Buf2SRAM* has run, c_j is available in the on-chip SRAM, and the CPU accesses it through *SRAM_Read*.

If we assume that only one task is running on the target SoC, then the transfers listed in Table I do not occur simultaneously (i.e., they do not overlap). Therefore, we can formulate E_{Paging} and T_{Paging} as follows:

$$\begin{aligned}
 E_{\text{Paging}} = & N_{\text{Flash2Buf}} \cdot E_{\text{Flash2Buf}} \\
 & + N_{\text{Buf2SRAM}} \cdot E_{\text{Buf2SRAM}} \\
 & + N_{\text{Buf_Read}} \cdot E_{\text{Buf_Read}} \\
 & + N_{\text{SRAM_Read}} \cdot E_{\text{SRAM_Read}} \quad (1)
 \end{aligned}$$

$$\begin{aligned}
 T_{\text{Paging}} = & N_{\text{Flash2Buf}} \cdot T_{\text{Flash2Buf}} \\
 & + N_{\text{Buf2SRAM}} \cdot T_{\text{Buf2SRAM}} \\
 & + N_{\text{Buf_Read}} \cdot T_{\text{Buf_Read}} \\
 & + N_{\text{SRAM_Read}} \cdot T_{\text{SRAM_Read}} \quad (2)
 \end{aligned}$$

Note that there are periods in the sequence shown in Fig. 5 during which no memory transfer is in progress, because of successive cache hits or an idle state in the CPU. As these idle periods are independent of the energy consumption and performance of demand paging, we do not include their energy consumption in E_{Paging} or their execution time in T_{Paging} .

In fact, most operating systems perform task switching at the time of a page fault to avoid the CPU to wait for completion of page fault handling. Therefore, some transfers can be overlapped, which will affect the formulation of E_{Paging} and T_{Paging} . Suppose that Task A is waiting for a page to be available, allowing Task B to become active. We see first that *Flash2Buf* can be performed by Task A at the same time as *SRAM_Read* or *Buf_Read* by Task B. Second, *Buf2SRAM* can be initiated by Task A only after *Flash2Buf* has been run by Task A, as shown in Fig. 5, and thus, they cannot be

overlapped. Third, the running of *Buf2SRAM* by Task A cannot be overlapped with the running of *SRAM_Read* or *Buf_Read* by Task B since *Buf2SRAM* involves both the on-chip SRAM and the OneNAND buffer.

In this case, the formulation of (1) is not changed for the following reasons.

- 1) It is obvious that $E_{\text{Flash2Buf}}$ is not affected by $E_{\text{SRAM_Read}}$ (and vice versa).
- 2) We have observed from real measurements that the energy required to perform *Flash2Buf* and *Buf_Read* simultaneously is almost the same as the sum of $E_{\text{Flash2Buf}}$ and $E_{\text{Buf_Read}}$.

Therefore, we can use (1) to evaluate E_{Paging} .

On the other hand, T_{Paging} may be reduced since the execution times of some *SRAM_Reads* or *Buf_Reads* will be masked by *Flash2Buf*. Although we can revise (2) by counting only those reads that are not masked by *Flash2Buf*, it would need a lot of time on a system-level simulator to find the exact number of overlapped transfers. However, we discovered experimentally that the sum of the execution times of all the *SRAM_Reads* and *Buf_Reads* is typically less than 5% of the total execution time. Therefore, the time taken by the overlapping *SRAM_Reads* and *Buf_Reads* will be far less than 5%, and (2) is still able to give us a tight upper bound on the actual value of T_{Paging} .

B. Simulation Tools for Evaluating the Cost Functions

1) *System-Level Energy Simulator*: To determine the value of E_{Transfer} and T_{Transfer} in (1) and (2), we developed a system-level energy simulator, the SNU Energy Explorer (SEE) 2.0, which is a complete revision of the SEE Web [26]. SEE 2.0 was developed using the transaction-level modeling facilities of SystemC [27]. The components of SEE 2.0 include an instruction set simulator (ISS) module, a level-1 cache module, and an advanced microcontroller bus architecture advanced high-performance bus (AHB) module. All the modules of the SEE 2.0 except the ISS are coded in the SystemC language. SEE 2.0 employs a hardware ISS accelerator module that uses a real ARM9TDMI processor core and is connected to the main simulator kernel through the peripheral component interconnect bus of the host computer. This ARM9TDMI core is fully static, so that it can retain its state even if the clock frequency is reduced, or the clock is actually stopped, which allows cosimulation between the HW ISS module and the SystemC simulator kernel. The detailed specifications of each module are summarized in Table II.

We developed an on-chip SRAM module, a OneNAND flash module, and a DMA module and integrated them into the AHB module of the SEE 2.0 to construct the memory system of the target SoC. The timing and energy models of the on-chip SRAM module were obtained from the datasheet for Samsung 130-nm compiled memory [29]. The timing and energy models of the OneNAND flash were constructed using the SNU energy characterizer for memory devices (SECM) [31], which is an in-house energy measurement tool for various off-chip memory devices. Table III shows the values of E_{Transfer} and T_{Transfer} that are used in this paper. $E_{\text{Flash2Buf}}$ and $T_{\text{Flash2Buf}}$ are

TABLE II
SPECIFICATION OF THE SYSTEM-LEVEL ENERGY SIMULATOR

Component	Feature
Simulator kernel	• SystemC 2.0.1 [27]
CPU core	• Real ARM9TDMI microprocessor • Connected through PCI interface • Operating frequency: 400 MHz
Cache memory	• Level-1 I-cache and D-cache • Size: 4 KB • Set-associativity: 2-way • Cache line size: 8 words
System bus	• AMBA advanced high-performance bus (AHB) • AHB cycle-level interface (AHBCLI) • 32-bit address bus and 32-bit data bus • Operating frequency: 100 MHz
On-chip SRAM	• Samsung 130 nm compiled memory [29] • Size: 1024 KB • Bus width: 32 bits • Operating frequency: 100 MHz
OneNAND flash	• Samsung KFG5616X1A [30] • Size: 256 Mb • Bus width: 16 bits • Operating frequency: 50 MHz
DMA controller	• Supports INCR16 burst transaction • Internal FIFO size: 16 words

TABLE III
EXPERIMENTAL VALUES OF $E_{Transfer}$ AND $T_{Transfer}$

$E_{Transfer}$	Value (nJ)	$T_{Transfer}$	Value (μ s)
$E_{Flash2Buf}$	1038.74	$T_{Flash2Buf}$	29.33
$E_{Buf2SRAM}$	875.31	$T_{Buf2SRAM}$	13.76
E_{Buf_Read}	26.63	T_{Buf_Read}	0.38
E_{SRAM_Read}	0.78	T_{SRAM_Read}	0.08

TABLE IV
BENCHMARK APPLICATIONS

Application	Description	Unique pages	Trace length
gqview	Graphic viewer	534	500 000
xpdf	PDF viewer	433	500 000
LiVES	Video editor	537	500 000
gimp	Image manipulation	409	500 000

solely determined by the characteristic of the OneNAND flash and are directly measured by the SECM. The other values were obtained from the simulation results from the SEE 2.0.

2) *Paging System Simulator*: The term $N_{Transfer}$ in (1) and (2) is determined by the page allocation policy, the page replacement policy, the size of the on-chip SRAM, and the sequence in which the application program requests pages. We implemented a simulator to model the behavior of the paging system with the policies under development. It runs fast (e.g., simulation of 500 000 page accesses can be done in 0.54 s) because it is trace driven, which allows repeated simulations to be performed in a reasonable length of time. We used Valgrind [32] to obtain the page request sequences of various application programs; we modified Cachegrind, which is the cache simulation module of Valgrind, to capture the trace of level-1 cache misses. Table IV shows the benchmark applications used for the simulation. Note that the third column in Table IV includes only the read-only pages, because all the read and write pages are pinned to the on-chip SRAM.

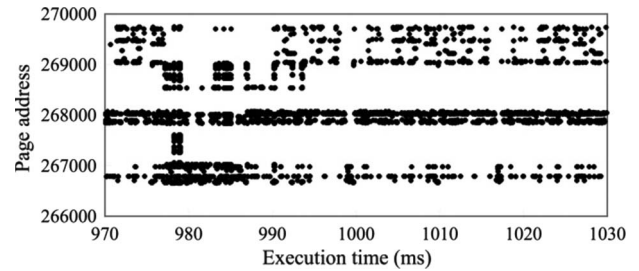


Fig. 6. Page request sequence of gqview, which is nonstationary.

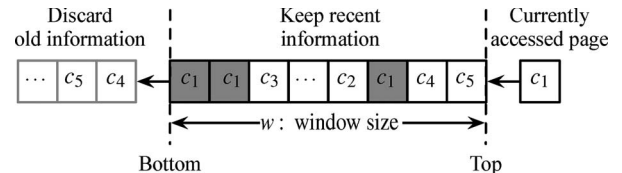


Fig. 7. Page history window.

C. PM-XIP: Proposed Page Fault Handler for OneNAND Demand Paging

PM-XIP is a new page fault handler for OneNAND demand paging. It consists of a page allocation policy and a page replacement policy, as outlined in Section IV-B.

A page replacement policy is a necessary part of conventional demand paging, and many sophisticated algorithms have been developed over several decades. However, the relationship between the performance of the page replacement policy and the page allocation policy has not been analyzed in previous work. We have observed experimentally that a page replacement policy, which has proved effective in conventional demand paging, also performs well for OneNAND demand paging. We therefore use CLOCK as the page replacement policy for PM-XIP, as it performs well and has a low computational overhead.

A different situation exists with page allocation: No existing policy meets the demands of paging with OneNAND flash. We have therefore developed a new page allocation policy which aims to fulfill the following requirements.

- 1) Predicts whether a page will be frequently used in the future. This requirement is shared with the page replacement policy. However, the page allocation policy has to estimate how frequently a page will be used in comparison to all the pages that are referenced, whereas the page replacement policy only needs to select the LFU page from all those in the on-chip SRAM.
- 2) Considers not only the frequency of page requests but also their recency. This is because page access patterns are not usually stationary, and thus, the access frequency of a page will change over time, as shown in Fig. 6.
- 3) Has a low storage and computation overhead, because we are aiming at lightweight portable embedded systems.

We use a page history window to satisfy these requirements. It is similar to the sliding window method that was introduced for receding-horizon control [33] and which has also been recently used to predict the future behavior of service requests in the dynamic power management of hard disk drives [34]. Fig. 7 shows the structure of the page history window. When a


```

PM-XIP( $S, w, t$ )
Input:  $S = (s_1, s_2, \dots, s_n)$ : a page request sequence,
 $w$ : size of the page history window,
 $t$ : a given threshold.
Output: Decision sequences for page allocation and
replacement.
1. do
2.   Fetch next requested page  $s_i$ .
3.   if ( $s_i$  is available from the on-chip SRAM)
4.     Perform SRAM_Read for  $s_i$ .
5.   elseif ( $s_i$  is available from the OneNAND buffer)
6.     Perform Buf_Read for  $s_i$ .
7.   else // a page fault occurs
8.     if (OneNAND buffer is full)
9.       Discard a victim page from the OneNAND
       buffer using CLOCK.
10.    endif
11.    Perform Flash2Buf for  $s_i$ .
12.    for (each page  $b_j$  in the OneNAND buffer)
13.       $n_{window}$  = the number of occurrences of  $b_j$ 
       in the page history window.
14.      if ( $n_{window} \geq t$ )
15.        if (on-chip SRAM is full)
16.          Discard a victim page from the
          on-chip SRAM using CLOCK.
17.        endif
18.        Perform Buf2SRAM for  $b_j$ .
19.      endif
20.    endfor
21.    go to line 3.
22.  endif
23.  Update the history window.
24. until (program is terminated.)
    
```

Fig. 8. Pseudocode of PM-XIP.

new page is accessed, it is pushed into the top of the window, and the oldest page in the page history window is discarded from the bottom of the window. A page allocation decision is made at every page fault by counting the number of occurrences of the missed page in the page history window and comparing the result with a predefined threshold. If the threshold has been reached, we assume that the page will continue to be frequently used in the near future, and allocate it to the on-chip SRAM (i.e., it is load preferred). Otherwise, we allocate it to the OneNAND buffer (i.e., it is XIP preferred). Due to its finite FIFO structure, the page history window can track both the frequency and recency of the page request sequence, at the cost of a reasonable computation and storage overhead.

The window size and the threshold are the primary control variables that determine the quality of the prediction made by the page history window. If the window is too large, the prediction is contaminated by old access patterns, which may be quite different from recent patterns. However, if the window is too small, there will be insufficient information for a good prediction. If the threshold is too high, frequently used pages can be misallocated to the OneNAND buffers, which may increase the number of page faults. However, if the threshold is too low, only a few pages can be allocated to the OneNAND buffer, and its XIP capability will not be fully exploited.

Fig. 8 shows the pseudocode of PM-XIP. If the required page is neither in the on-chip SRAM nor in the OneNAND buffers, a page fault occurs (line 7). The missed page is loaded into one of the OneNAND buffers (line 11), and the pages in both the OneNAND buffers are evaluated by the page history window

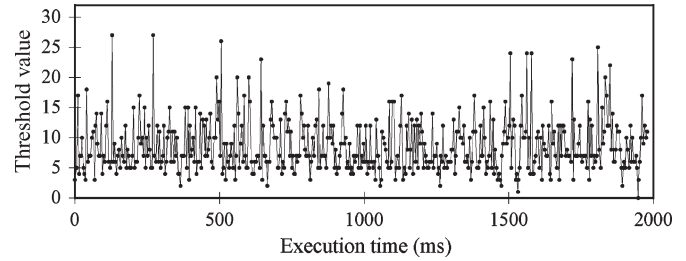
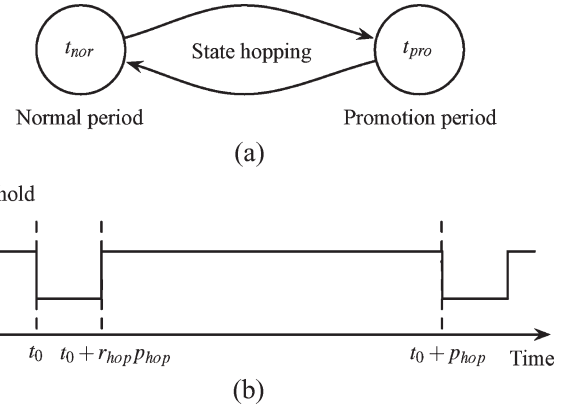

 Fig. 9. Threshold values determined by T-noncausal (application: *gqview*; page cache size: 24 kB; $w = 32$).


Fig. 10. Mechanism of T-hopping. (a) State transition diagram. (b) Timing diagram.

(lines 12 and 13). If there is no more free space in the OneNAND buffers or the on-chip SRAM, *CLOCK* selects a victim page and replaces it with the required page (lines 9 and 16).

Note that a frequently used page can be allocated to the OneNAND buffer if there are not enough access records for that page in the page history window (e.g., when it is accessed for the first time). However, the page history window quickly learns its access pattern, and the hot page has a chance to be loaded to the on-chip SRAM at every page fault since PM-XIP evaluates all the pages in the OneNAND buffers (line 12). Therefore, the hot page will soon move from the OneNAND buffer to the on-chip SRAM.

D. Determining the Size of the Page History Window and Its Threshold

We sized the page history window experimentally, while considering the tradeoff between prediction quality and storage overhead, and the size is not changed at runtime. To determine the threshold value, we considered using an offline method, T-noncausal, and two online methods, T-single and T-hopping.

1) *T-Noncausal*: T-noncausal periodically readjusts the threshold by looking ahead. The simulation begins with a zero threshold for the current period of time. This has length p_{non} and is determined before the simulation. After the simulation is finished, T-noncausal records the resulting values of E_{Paging} and T_{Paging} and rolls back to the beginning of the current period while incrementing the threshold by one. T-noncausal repeats the simulation until the threshold becomes equal to the size of the page history window, and then selects

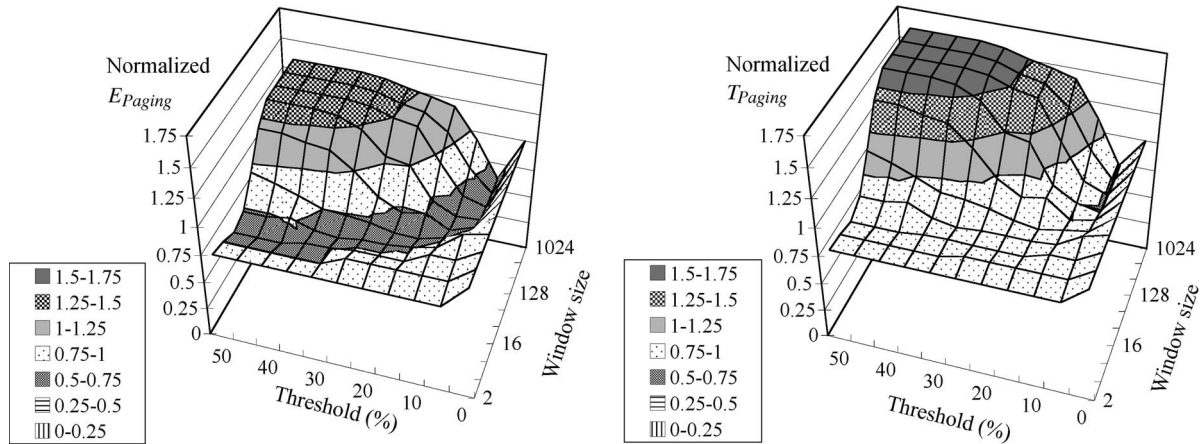


Fig. 11. Values of E_{Paging} and T_{Paging} for PM-XIP, normalized to conventional demand paging, for various window sizes and threshold values (application: gqview; page cache size: 24 kB).

the threshold value for that period which minimizes E_{Paging} or T_{Paging} . T-noncausal repeats this procedure for the next period and so on, until it reaches the end of the page request sequence.

T-noncausal cannot be implemented in practice unless the page request sequence is known in advance, but it can serve as a reference to compare T-single and T-hopping. We claim that T-noncausal is not an optimal offline solution but a near-ideal method of periodical threshold adjustment. To the best of our knowledge, no optimal offline algorithm for page allocation is yet known unlike Belady's MIN for page replacement. A decision of page allocation may affect the optimal decision of page replacement; Belady's MIN is not optimal any more for the demand paging with OneNAND flash. In other words, just combining Belady's MIN and the optimal page allocation algorithm, if any, does not guarantee the true optimal solution for demand paging with OneNAND flash.

2) *T-Single*: We can expect to know the specifications of an embedded system, including the on-chip SRAM size and the target applications. We can therefore select a good threshold value t_{sin} by offline analysis of a sample trace (i.e., part of the whole page request sequence) with the given size of the on-chip SRAM. The chosen value of t_{sin} is not changed at runtime. The analysis of the sample trace is similar to that performed by T-noncausal. The simulation is repeated while changing the threshold from zero to the size of the page history window, and then, the best threshold value is selected. Consequently, the efficiency of this method, which we call T-single, is totally dependent on how well the sample trace represents the whole page request sequence.

3) *T-Hopping*: In practice, the page request pattern of an application often changes abruptly, for example, when a program enters a new execution phase. Consequently, the optimal threshold value may vary significantly as time elapses, as shown in Fig. 9. Therefore, a single threshold is usually inadequate, and we need to develop a threshold adaptation algorithm that can track the change of the page request pattern.

The mechanism of T-hopping, as shown in Fig. 10, involves a normal period and a promotion period, which are, respectively, associated with the two threshold values t_{nor} and t_{pro} ($t_{\text{nor}} > t_{\text{pro}}$). The parameters p_{hop} and r_{hop} ($0 < r_{\text{hop}} < 1$) control the

length of both the periods. In the normal period, T-hopping uses the threshold value t_{nor} to classify the most frequently accessed pages as load preferred. However, some pages may not be classified as a load-preferred page, even though they deserve to be loaded into the on-chip SRAM, which may result in a significant number of page faults. In order to compensate for these incorrect page allocations, T-hopping periodically enters a promotion period and lowers the threshold to t_{pro} for a short time. This allows some additional pages to be allocated to the on-chip SRAM. Of course, some undeserving pages may be loaded into the on-chip SRAM during the promotion period, but they will be used less frequently than the other pages in the on-chip SRAM, and will therefore soon be evicted, avoiding serious memory pollution.

The parameter values of T-hopping should be carefully selected in order to achieve substantial energy saving and performance improvement. Otherwise, its promotion periods may rather increase the number of page faults so that T-hopping performs worse than conventional demand paging. At the same time, the configuration process of T-hopping should be simple enough so that it can be easily implemented in practice. Considering these concerns, we suggest an offline analysis using a small-size sample trace, like T-single. We will show in the following section that T-hopping generally outperforms T-single and is less sensitive to the sample trace in spite of its complicated structure, compared with T-single.

VI. PERFORMANCE EVALUATION

A. Effect of Configuring the Page History Window on E_{Paging} and T_{Paging}

We started by investigating the energy consumption and performance of PM-XIP by changing the window size from 2 to 1024 and its threshold from 0% to 50% of that size. These parameters are then fixed during runtime. The simulation results are shown in Fig. 11, where the y -axes are normalized to the results from conventional demand paging, which we call ConvPaging.

The simulation results show that the efficiency of PM-XIP strongly depends on the window size and the threshold value.

TABLE V
ENERGY CONSUMPTION AND EXECUTION TIME OF PM-XIP FOR
THREE THRESHOLD VALUES (APPLICATION: gqview;
PAGE CACHE SIZE: 24 kB; $w = 32$)

Performance index	Threshold value		
	0	2	20
$N_{Flash2Buf}$	60 865	55 665	144 067
$N_{Buf2SRAM}$	60 865	36 478	5
N_{SRAM_Read}	500 000	322 669	91 589
N_{Buf_Read}	0	177 331	408 411
$N_{Flash2Buf} \cdot E_{Flash2Buf}$ (mJ)	63.22	57.82	149.65
$N_{Buf2SRAM} \cdot E_{Buf2SRAM}$ (mJ)	53.28	31.93	0.01
$N_{SRAM_Read} \cdot E_{SRAM_Read}$ (mJ)	0.39	0.25	0.07
$N_{Buf_Read} \cdot E_{Buf_Read}$ (mJ)	0.00	4.72	10.87
E_{Paging} (mJ)	116.89	94.72	160.60
$N_{Flash2Buf} \cdot T_{Flash2Buf}$ (ms)	1785.17	1632.65	4225.49
$N_{Buf2SRAM} \cdot T_{Buf2SRAM}$ (ms)	837.50	501.94	0.07
$N_{SRAM_Read} \cdot T_{SRAM_Read}$ (ms)	40.00	25.81	7.33
$N_{Buf_Read} \cdot T_{Buf_Read}$ (ms)	0.00	67.39	155.20
T_{Paging} (ms)	2662.67	2227.79	4388.08

When we set (w, t) to $(32, 5)$, E_{Paging} and T_{Paging} were reduced by 33.2% and 24.2%, respectively. However, when we set (w, t) to $(128, 52)$, they increased by 33.3% and 58.9%. We also observe that the shapes of two graphs in Fig. 11 closely resemble each other, which suggests that a page history window which is optimized to reduce E_{Paging} may also be nearly optimal for T_{Paging} . We confirmed by experiments that this observation is valid not only for gqview but also for the other applications. We will therefore use E_{Paging} as our primary metric.

Table V provides further information about the simulations shown in Fig. 11. The results in the second column are equivalent to those we would expect for ConvPaging, because the threshold is set to zero so that all the missed pages are allocated to the on-chip SRAM (i.e., the number of page faults $N_{Flash2Buf}$ equals the number of pages loaded from the OneNAND flash to the on-chip SRAM, $N_{Buf2SRAM}$). The third column contains results for a threshold of 2, when 65.5% of the missed pages were allocated to the on-chip SRAM and the remainder to the OneNAND buffer. This allocation reduces both $N_{Flash2Buf}$ and $N_{Buf2SRAM}$. There is, of course, an overhead in accessing the code pages in the OneNAND buffer, which is slower and more energy-consuming than the on-chip SRAM, but this is outweighed by the saving in energy and improved performance. E_{Paging} and T_{Paging} are reduced by 19.0% and 16.3%, respectively, compared with ConvPaging. The fourth column contains the results for a threshold value of 20, which only allows a few of the most frequently accessed pages to be allocated to the on-chip SRAM. Despite the significant reduction in $N_{Buf2SRAM}$, E_{Paging} and T_{Paging} are not reduced because of the growth of $N_{Flash2Buf}$. Table V suggests that the energy consumption and performance of PM-XIP are more sensitive to the threshold value than to the size of the page history window.

B. Determination of Window Size

We investigated how the size of the page history window affects the efficiency of PM-XIP. In these experiments, we

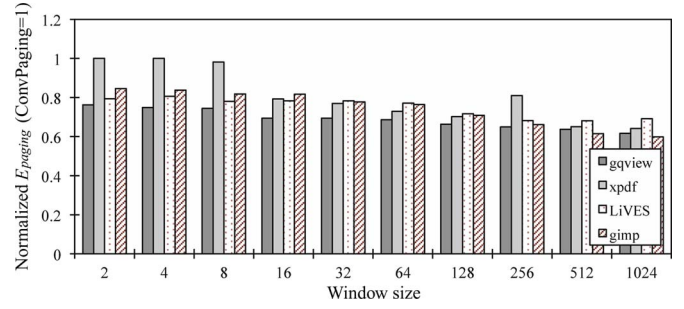


Fig. 12. Energy consumption of PM-XIP, normalized to conventional demand paging, for various window sizes (page cache size: 32 kB).

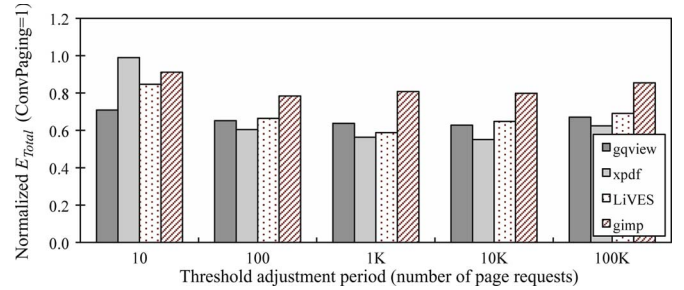


Fig. 13. Effect of the threshold adjustment period on the energy consumption of T-noncausal (page cache size: 24 kB; $w = 32$).

used T-single to determine the threshold value and varied the window size as shown in Fig. 12. The results show that PM-XIP generally uses less energy with a larger window size, but above 16, there is little further benefit (note that the x -axis of Fig. 12 has a logarithmic scale). Moreover, Fig. 11 shows that, as the window size becomes larger, the range of the threshold values for which PM-XIP is superior to ConvPaging is reduced, increasing the risk that decisions based on a threshold will not reduce E_{Paging} and T_{Paging} . In addition, the size of the page history window largely determines the overhead of PM-XIP. If we have a small page history window (e.g., fewer than 32 entries), a range of efficient implementations become available such as a dedicated hardware FIFO module or a cache locking scheme (i.e., a dedicated area of the on-chip SRAM). However, if the page history window has, for example, 1024 entries, it needs 4 kB of memory space, and these fast hardware methods become too expensive. We therefore set the window size to 32 for the remaining experiments. We think that this value represents a good compromise between theoretical and practical efficiencies.

C. Determination of Threshold Value

We investigated the energy variations of T-noncausal, T-single, and T-hopping with different configurations. Intuitively, we would expect that smaller values of p_{non} allow T-noncausal to follow changes to the page access pattern more accurately. Fig. 13 shows the efficiency of T-noncausal for various threshold adjustment periods. In most cases, shortening p_{non} reduces E_{Paging} as expected, but too short a period actually increases E_{Paging} , and the optimal period ranges from 100 to 10k depending on the page request sequence.

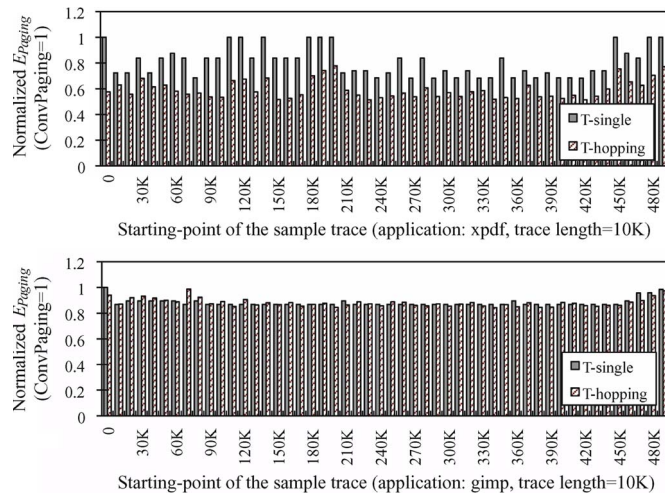


Fig. 14. How the energy consumptions of T-single and T-hopping depend on the position of the sample trace (page cache size: 24 kB; $w = 32$).

We observed the energy variation of T-single and T-hopping by changing the starting point for acquisition of the sample trace, which has a length of 2% of the page request sequence. Fig. 14 shows the simulation results for xpdf and gimp. For xpdf, T-single fails to reduce E_{Paging} for many sample traces, suggesting that these samples do not represent the access pattern of xpdf very well. The energy consumptions of T-hopping and T-single dip for the same traces, but T-hopping saves more energy than T-single. The results for gimp were rather different. The amount of energy saved by T-single is not much affected by the location of the sample trace. This indicates that the page access pattern of gimp changes relatively little over time, which is advantageous for T-single. In this situation, the promotion period of T-hopping might be expected to be an unnecessary overhead; however, T-hopping actually shows comparable energy consumption to T-single, which means that its overhead is not so significant. To summarize, T-hopping is less sensitive to the position of the sample trace and generally superior to T-single.

D. Effect of Changing Working Set Size

In Section III-A, we said that an increase in code size, due to software upgrade after an SoC has been fabricated, can significantly make worse the performance of demand paging. We investigated experimentally whether PM-XIP can alleviate this problem. Starting with a situation in which LiVES is executed on a 48-kB page cache, we added the other applications one by one. We emulated a multiprogramming environment by modifying the paging system simulator to support periodic switching between different page request sequences. Table VI shows that PM-XIP can reduce both E_{Paging} and T_{Paging} when many applications are being executed simultaneously in an on-chip SRAM of fixed size.

E. Effect of Changing Page Replacement Policy

To observe the effect of the page replacement policy on the energy consumption of PM-XIP, we implemented MIN,

TABLE VI
EFFECT OF INCREASING CODE SIZE ON THE ENERGY CONSUMPTION OF PM-XIP (PAGE CACHE SIZE: 48 kB; $w = 32$)

Applications	ConvPaging	T-hopping	Improvement
E_{Paging} (mJ)			
LiVES	23.10	19.75	14.5%
LiVES+xpdf	89.86	69.80	22.3%
LiVES+xpdf+gimp	118.59	84.69	28.6%
LiVES+xpdf+gimp+gqview	472.45	292.76	38.0%
T_{Paging} (ms)			
LiVES	551.18	493.97	10.4%
LiVES+xpdf	2085.41	1723.10	17.4%
LiVES+xpdf+gimp	2732.15	2241.92	17.9%
LiVES+xpdf+gimp+gqview	10760.92	7624.55	29.1%

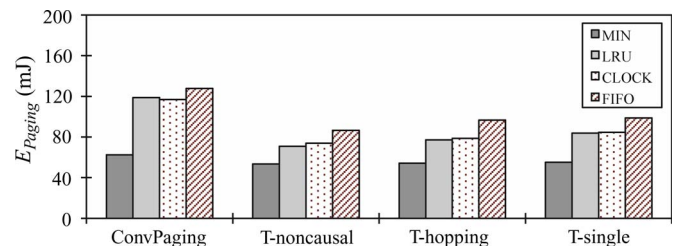


Fig. 15. Effect of the page replacement policy on the energy consumption of PM-XIP (application: gqview; page cache size: 24 kB; $w = 32$).

LRU, and FIFO as alternatives to CLOCK. Fig. 15 shows the energy consumptions of ConvPaging and PM-XIP with the three threshold decision methods. In ConvPaging, the energy consumptions of CLOCK and LRU are comparable, but FIFO does worse. This ordering does not change very much under PM-XIP, suggesting that a page replacement policy which is effective for conventional demand paging also works well for demand paging with OneNAND flash. If a better page replacement policy were to become available, we could easily substitute it for CLOCK to improve the energy consumption and performance of PM-XIP.

F. Overall Energy Consumption and Performance of PM-XIP

We compared the overall energy consumption and performance of PM-XIP for various applications and on-chip SRAM sizes, and the results are shown in Table VII. We used the best value of p_{non} for each page request sequence to configure T-noncausal, and we extracted a 10K sample trace from the center of the page request sequence of each application for T-single and T-hopping. Table VII shows that T-single achieves an average 19% reduction of E_{Paging} and a 14% reduction of T_{Paging} . T-hopping outperforms T-single for various applications and on-chip SRAM sizes, achieving an average 26% reduction of E_{Paging} and a 19% reduction of T_{Paging} . We see that T-hopping did better than ConvPaging for LiVES with 32 and 48 kB of page caches, whereas T-single did worse with these configurations, which is in keeping with the results shown in Fig. 14. We performed the same simulations for 8- and 16-kB level-1 cache sizes and confirmed similar performance for all these cache sizes. However, those results are omitted because of the limited space.

TABLE VII
OVERALL ENERGY CONSUMPTION AND PERFORMANCE OF PM-XIP ($w = 32$, E_{Paging} , AND T_{Paging} ARE NORMALIZED TO THOSE OF ConvPaging)

Target SoC		ConvPaging		T-noncausal			T-single			T-hopping					
Application	Page cache	E_{Paging} (mJ)	T_{Paging} (ms)	E_{Paging}	T_{Paging}	p_{non}	E_{Paging}	T_{Paging}	t_{sin}	E_{Paging}	T_{Paging}	t_{nor}	t_{pro}	p_{hop} (μs)	r_{hop}
gqview	16 KB	146.41	3327.21	0.656	0.768	500	0.736	0.829	5	0.681	0.798	14	3	64.0	0.10
	32 KB	87.66	2004.65	0.645	0.760	1600	0.757	0.820	3	0.687	0.793	10	2	4.0	0.12
	48 KB	49.41	1143.62	0.665	0.783	300	0.803	0.884	3	0.750	0.828	6	0	0.5	0.08
	64 KB	27.43	648.78	0.670	0.793	300	0.936	0.869	1	0.800	0.821	6	0	64.0	0.18
xpdf	8 KB	163.88	3720.58	0.520	0.621	800	0.636	0.767	12	0.549	0.645	14	1	8.0	0.02
	16 KB	87.48	2000.64	0.467	0.547	600	0.751	0.897	6	0.580	0.649	12	0	64.0	0.16
	24 KB	51.93	1200.28	0.457	0.533	600	0.682	0.798	4	0.574	0.641	12	0	4.0	0.12
	32 KB	20.49	492.62	0.588	0.661	1100	0.769	0.777	1	0.687	0.724	4	0	2.0	0.20
LiVES	16 KB	105.45	2405.25	0.589	0.685	200	0.791	0.819	2	0.706	0.788	6	1	0.5	0.08
	32 KB	49.82	1152.80	0.612	0.706	1200	1.000	1.000	0	0.789	0.829	4	0	1.0	0.20
	48 KB	23.10	551.18	0.736	0.824	1300	1.000	1.000	0	0.850	0.894	4	0	1.0	0.20
	64 KB	11.86	298.28	0.872	0.949	1900	1.000	1.000	0	0.992	1.022	4	0	1.0	0.20
gimp	16 KB	100.40	2291.41	0.763	0.849	300	0.836	0.855	2	0.830	0.932	10	2	4.0	0.20
	32 KB	57.22	1319.47	0.561	0.670	1800	0.836	0.869	2	0.723	0.851	12	2	32.0	0.18
	48 KB	33.98	796.23	0.329	0.404	1300	0.486	0.558	2	0.452	0.528	4	1	64.0	0.18
	64 KB	3.09	100.76	0.954	0.994	1500	1.000	1.000	0	1.120	1.122	8	0	1.0	0.20
Average		N/A	N/A	0.630	0.722	N/A	0.814	0.864	N/A	0.736	0.804	N/A	N/A	N/A	N/A
Reduction		N/A	N/A	37.0%	27.8%	N/A	18.6%	13.6%	N/A	26.4%	19.6%	N/A	N/A	N/A	N/A

It is possible that T-hopping performs worse than conventional demand paging, for example, in the case of gimp with 64-kB page cache. This is because we tuned the parameters of T-hopping with the sample trace obtained from a part of the page request sequence, rather than with the whole sequence, which is not feasible in practice. Note that, even in such a case, the absolute amount of the performance overhead caused by T-hopping is not so significant.

VII. CONCLUSION

Current state-of-the-art fusion memory devices integrate multiple heterogeneous memory components to reduce the complexity of embedded systems. However, fusion memory can also affect the energy consumption and performance of a system, and this is an aspect that has not yet been fully investigated. Furthermore, existing paging schemes, which were designed for legacy memory devices, may not give the best results with fusion memory devices.

We have presented a new demand paging scheme that is designed for Samsung’s recent OneNAND fusion flash memory, which integrates a NAND flash and dual SRAM buffers, within the context of an embedded system. We have also introduced a new page fault handler for demand paging, PM-XIP, which fully exploits the advanced features of OneNAND flash. Our simulation results show that PM-XIP outperforms conventional demand paging, on average, by 26% in terms of energy consumption and by 19% in terms of execution time. We expect that the combination of the OneNAND demand paging system with a novel page fault handler such as PM-XIP will also allow significant reduction in the size of the on-chip SRAM in low-to mid-end embedded systems, in which cost effectiveness is critical.

ACKNOWLEDGMENT

The authors would like to thank the ICT at Seoul National University for providing research facilities for this study.

REFERENCES

- [1] Y. Joo, Y. Choi, C. Park, S. W. Chung, E. Chung, and N. Chang, “Demand paging for OneNAND flash execute-in-place,” in *Proc. Int. Conf. Hardware/Software Codes. Syst. Synthesis*, 2006, pp. 229–234.
- [2] *OneNAND Features & Performance*, Dec. 2005, Samsung Electronics Co. Ltd. [Online]. Available: <http://www.samsungelectronics.com>
- [3] C. Park, J. Lim, K. Kwon, J. Lee, and S. L. Min, “Compiler-assisted demand paging for embedded systems with flash memory,” in *Proc. ACM Int. Conf. Embedded Softw.*, Sep. 2004, pp. 114–124.
- [4] H.-W. Park, K. Oh, S. Park, M.-M. Sim, and S. Ha, “Dynamic code overlay of SDF-modeled programs on low-end embedded systems,” in *Proc. Conf. Des. Autom. Test Eur.*, 2006, pp. 945–946.
- [5] H.-W. Tseng, H.-L. Li, and C.-L. Yang, “An energy-efficient virtual memory system with flash memory as the secondary storage,” in *Proc. Int. Symp. Low Power Electron. Des.*, 2006, pp. 418–423.
- [6] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee, “CFLRU: A replacement algorithm for flash memory,” in *Proc. Int. Conf. Compilers, Arch. Synthesis Embedded Syst.*, 2006, pp. 234–241.
- [7] “M-Systems DOC H3,” *M-Systems*, 2006. [Online]. Available: <http://www.m-systems.com>
- [8] P. R. Panda, N. D. Dutt, and A. Nicolau, “Efficient utilization of scratchpad memory in embedded processor applications,” in *Proc. Eur. Des. Test Conf.*, Mar. 1997, pp. 7–11.
- [9] J. P. Diguët, S. Wuytack, F. Cathoor, and H. D. Man, “Formalized methodology for data reuse exploration in hierarchical memory mappings,” in *Proc. Int. Symp. Low Power Electron. Des.*, Aug. 1997, pp. 30–35.
- [10] M. Kandemir and A. Choudhary, “Compiler-directed scratch pad memory hierarchy design and management,” in *Proc. Des. Autom. Conf.*, Jun. 2002, pp. 690–695.
- [11] I. Issenin, E. Brockmeyer, M. Miranda, and N. Dutt, “Data reuse analysis technique for software-controlled memory hierarchies,” in *Proc. Des. Autom. Test Eur. Conf.*, Feb. 2004, pp. 202–207.
- [12] B. Egger, C. Kim, C. Jang, Y. Nam, J. Lee, and S. L. Min, “A dynamic code placement technique for scratchpad memory using postpass optimization,” in *Proc. Int. Conf. Compilers, Arch. Synthesis Embedded Syst.*, 2006, pp. 223–233.
- [13] L. A. Belady, “A study of replacement algorithms for virtual storage computers,” *IBM Syst. J.*, vol. 5, no. 2, pp. 78–101, 1966.
- [14] D. D. Sleator and R. E. Tarjan, “Amortized efficiency of list update rules,” in *Proc. Annu. ACM Symp. Theory Comput.*, 1984, pp. 488–492.
- [15] E. J. O’Neil, P. E. O’Neil, and G. Weikum, “The LRU-K page replacement algorithm for database disk buffering,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1993, pp. 297–306.
- [16] D. Lee, J. Choi, H. Choe, S. Noh, S. Min, and Y. Cho, “Implementation and performance evaluation of the LRFU replacement policy,” in *Proc. Euromicro Conf.*, 1997, pp. 106–111.
- [17] S. Jiang and X. Zhang, “LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance,” in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2002, pp. 31–42.

- [18] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proc. USENIX Conf. File Storage Technol.*, 2003, pp. 115–130.
- [19] F. J. Corbato, *A Paging Experiment With the Multics System*. Cambridge, MA: MIT Press, 1969, pp. 217–228. In Honor of Philip M. Morse.
- [20] I. Ari, M. Gottwals, and D. Henze, "Performance boosting and workload isolation in storage area networks with SANCACHE," in *Proc. NASA Conf. Mass Storage Syst. Technol.*, May 2006, pp. 263–273.
- [21] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, "Evaluating content management techniques for web proxy caches," *ACM SIGMETRICS Perf. Eval. Rev.*, vol. 27, no. 4, pp. 3–11, Mar. 2000.
- [22] *Two Technologies Compared: NOR vs. NAND*, 2003. White Paper, 91-SR-012-04-8L, Rev 1.1: M-Systems.
- [23] H. Kim, J. In, D. Ham, S. Yoon, and D. Shin, "Virtual-ROM: A new demand paging component for RTOS and NAND flash memory based mobile devices," in *ISCIS*, vol. 4263. Berlin, Germany: Springer-Verlag, 2006, pp. 677–686.
- [24] *Demand Paging on Symbian OS*, Symbian Ltd. [Online]. Available: <http://www.symbian.com/symbianos/demandpaging/index.html>
- [25] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," *IEEE Trans. Consum. Electron.*, vol. 48, no. 2, pp. 366–375, May 2002.
- [26] I. Lee, Y. Choi, Y. Cho, Y. Joo, H. Lim, H. G. Lee, H. Shim, and N. Chang, "Web-based energy exploration tool for embedded systems," *IEEE Des. Test Comput.*, vol. 21, no. 6, pp. 572–586, Nov./Dec. 2004.
- [27] *SystemC 2.0.1 Language Reference Manual*, 2002. [Online]. Available: <http://www.systemc.org>
- [28] *AMBA AHB Cycle Level Interface (AHB CLI) Specification*, 2003. [Online]. Available: <http://www.arm.com/products/solutions/ahbcli.html>
- [29] *STD150 0.13 μm 1.2V CMOS Standard Cell Library for Pure Logic Products*, Feb. 2004, Samsung Electronics, Co. Ltd. [Online]. Available: <http://www.samsungelectronics.com>
- [30] *KFG5616x1A x16 OneNAND Specification*, Dec. 2005, Samsung Electronics, Co. Ltd. [Online]. Available: <http://www.samsungelectronics.com>
- [31] Y. Joo, Y. Cho, D. Shin, J. Park, and N. Chang, "An energy characterization platform for memory devices and energy-aware data compression for multilevel-cell flash memory," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, no. 3, pp. 1–29, Jul. 2008.
- [32] N. Nethercote and J. Seward, "Valgrind: A program supervision framework," *Electron. Notes Theor. Comput. Sci.*, vol. 89, no. 2, pp. 1–23, 2003.
- [33] W. Kwon, D. Byun, and O. Kwon, "Receding horizon tracking control as a predictive control and its stability properties," in *Proc. Amer. Control Conf.*, 1988, pp. 2070–2075.
- [34] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. D. Micheli, "Dynamic power management for nonstationary service requests," *IEEE Trans. Comput.*, vol. 51, no. 11, pp. 1345–1361, Nov. 2002.



Yongsoo Joo (S'01–M'08) received the B.S. and M.S. degrees in computer engineering and the Ph.D. degree in electrical and computer engineering from Seoul National University, Seoul, Korea, in 2000, 2002, and 2007, respectively.

He is currently a Postdoctoral Researcher with the Department of Electrical Engineering and Computer Science, Seoul National University. His research interests include energy-aware memory system design and system-level energy and performance estimation.



Yongseok Choi (S'01–M'08) received the B.S. and M.S. degrees in computer engineering and the Ph.D. degree in electrical and computer engineering from Seoul National University, Seoul, Korea, in 2000, 2002, and 2007, respectively.

Since November 2007, he has been with the Digital Media R&D Center, Samsung Electronics, Suwon, Korea. His research interests include embedded systems design and system-level low-power design.



Jaehyun Park (S'08) received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea, in 2006, where he is currently working toward the Ph.D. degree in electrical engineering and computer science.

His research interests include low-power embedded system design and system-on-chip design.



Chanik Park received the B.S. and M.S. degrees in computer engineering and the Ph.D. degree in electrical and computer engineering from Seoul National University, Seoul, Korea, in 1995, 1997, and 2002, respectively.

Since 2004, he has been a Senior Engineer with the Memory Division, Samsung Electronics, Hwaseong, Korea. His research interests include embedded storage architecture and high-performance and reliable solid-state drive-based NAND flash memories with the assistance of hardware/software codesign.



Sung Woo Chung (M'06) received the B.S. degree in computer engineering and the Ph.D. degree in electrical and computer engineering from Seoul National University, Seoul, Korea, in 1996 and 2003, respectively.

Since 2006, he has been an Assistant Professor with the Division of Computer and Communication Engineering, Korea University, Seoul. His research interests include technology-aware design and architectural supports for flash memories.



Eui-Young Chung (M'06) received the B.S. and M.S. degrees in electronics and computer engineering from Korea University, Seoul, Korea, in 1988 and 1990, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 2002.

From 1990 to 2005, he was a Principal Engineer with SoC R&D Center, Samsung Electronics, Yongin, Korea. He is currently an Associate Professor with the School of Electrical and Electronic Engineering, Yonsei University, Seoul. His research

interests include system architecture and VLSI design, including all aspects of computer-aided design with the special emphasis on low-power applications and flash memory applications.



Naehyuck Chang (M'97–SM'05) received the B.S., M.S., and Ph.D. degrees from the Department of Control and Instrumentation, Seoul National University, Seoul, Korea, in 1989, 1992, and 1996, respectively.

He is currently an Associate Professor with the Department Electrical Engineering and Computer Science, Seoul National University.

Dr. Chang is a Senior Member of the Association for Computing Machinery. He serves or served as a member of the technical program committee of

ACM SIGDA and IEEE Circuits and Systems Society conferences and symposiums such as DAC, ICCAD, ISLPED, DATE, CODES+ISSS, ISQED, GLS-VLSI, ASP-DAC, and so on. He is currently an Associate Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, *Journal of Low-Power Electronics*, and *Journal of Embedded Computing*.