Low-Energy Heterogeneous Non-volatile Memory Systems for Mobile Systems

Hyung Gyu Lee and Naehyuck Chang†
School of Computer Science and Engineering
Seoul National University, Korea
Email: {hglee, naehyuck}@cselab.snu.ac.kr

Abstract

Memory systems consume significant energy in hand-held embedded systems. Existing techniques for reducing memory energy requirements in low-power systems have addressed energy consumption when the system is turned on; but we also consider data retention energy during the power-off period.

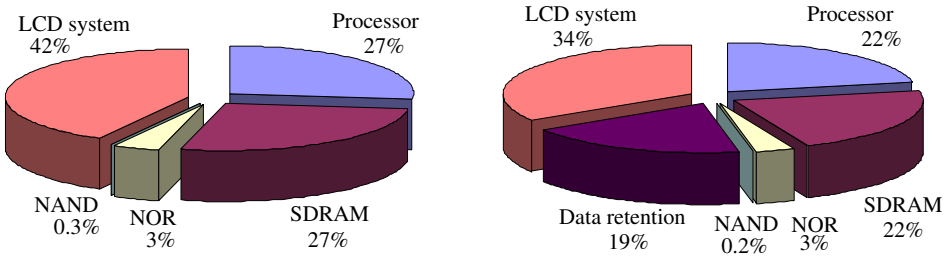
Semiconductor non-volatile memory is indispensable for hand-held devices that cannot afford magnetic disks due to excessive space, weight, cost and energy consumption. Current hand-held systems are generally equipped with more than one type of non-volatile storage device, such as battery-backed SDRAM, NOR Flash memory or NAND Flash memory, because each technology has its distinct and complementary features. In this paper, we introduce an energy-aware memory allocation in heterogeneous non-volatile memory systems to maximize the battery life. For this purpose, we first characterize cycle-accurate active mode energy and the data retention energy of non-volatile memory systems. Next, we present an energy-aware memory allocation for a given task set, taking into account arrival rate, execution time, code size, user data size and the number of memory transactions; we do this using trace-driven simulation. Experiments demonstrate that an optimized allocation can save up to 26% of the memory system energy compared with traditional allocation schemes.

I. Introduction

As the use of battery-operated embedded systems increases, optimizing the energy consumption of computing components has become as important as optimizing their performance. Energy optimization is important for all components, including the processor, cache, bus, memory, peripherals and the LCD system. Recently, even battery-operated portable systems get equipped with high-performance and large capacity memory systems to accommodate multimedia and other data-intensive applications; and thus the memory system become an increasingly dominant energy consumer, which motivates many techniques for reducing memory energy consumption. Fig. 1(a) shows an average-power breakdown for an embedded system during program execution. About 30% of power is consumed in the memory system.

Much research has been performed to reduce memory system energy consumption. Theoretical approaches have included developing a methodology for system-wide energy estimation for a hand-held embedded system [1]. Simunic et al provide a energy model that includes the

†Corresponding author. The RIACT at Seoul National University provide research facilities for this study. This work was partly supported by the Brain Korea 21 Project.



(a) Average power cosumption during program execution (b) Power consumption for 1 day considering data retention

Fig. 1. Power breakdowns of embedded system.[1]

memory system so as to achieve a energy-efficient design. There are also low-power memory partitioning (allocation) schemes. Memory partitioning among different-size sub-banks, with different energy densities, also contributes to low-power memory systems [2]. Finding an optimal configuration for a memory system can also reduce power consumption [3]. Proper clustering gathers busy blocks in the same banks, which helps to maximize the chances of the other banks being powered down [4], [5], [6]. However, all of their techniques focus only on the main memory device and usually consider only one type of RAM. Thus these approaches are most useful for only desktop applications, in which SDRAM main memory is involved in most memory accesses.

Recently, modern hand-held embedded systems are equipped with non-volatile primary storage for a fast boot, fast hibernation and the retention of user data during the power-off period while desktop computers are considered only for cold boot. Therefore, they require balanced read and write performance, no explicit erase operation, random access capability, cost effective large capacity, and true non-volatility so that the memory device can be used as a non-volatile primary storage. Unfortunately, there is still no mass-produced semiconductor device that satisfies all the above requirements. Thus, most hand-held systems are equipped with heterogeneous non-volatile memory devices combining different types of memory that work in a complementary manner. The heterogeneous nature of the memory system must be considered in pursuing energy reduction.

Non-volatile memory systems consume energy when the power is on as well as when the power is off. Active-mode, idle mode and sleep mode energy are consumed when power is on, and back-up or erase and write energy for data retention are consumed when power is off. In reality, memory systems consume a significant amount of energy when the system is in the power-off state or in data retention mode. For example, low-power PDAs (Personal Data Assistants) are known to consume more than half of the energy in the battery for data retention. Fig. 1(b) shows a breakdown of power over one day of use that distinguishes the requirements of program execution and data retention. The data retention energy comprises a significant proportion of total energy consumption. However, most previous low-power techniques for the memory systems have focused on the reduction of energy consumption while power is on.

There is relatively small literature that deals with data retention energy. Palm Pilot opened a new era of PDAs powered by Hotsync technology. The active power consumption of a Palm Pilot Pro is reported to be between 130mW and 150mW in the worst case, while the data retention power (in sleep mode) is 26mW [8]. The Palm Pilot series are equipped with asynchronous DRAM devices and retain data by battery back-up. Using a 1,200mAH battery, the

[1]These graphs are cycle-accurate simulation results by the use of a system-level energy simulator [7].

manufacturer suggests that up to 10 hours' continuous is possible, and the power consumption in active mode appears to justify that data. However, data sheets suggest a 1mW data retention power for an asynchronous DRAM. The sleep-mode power cited by Newman and Hong [8] does not seem to be only the data retention power because we commonly experience roughly a month-long battery life with a Palm Pilot if we do not turn on the device. Even if we go for weeks without battery charging, the actual run time is a few hours, although the battery becomes flat more quickly if the device is used for intensive applications such as games. This suggests that more than half of the battery energy usually goes on data retention.

So, if we are aiming at a globally optimal memory system for battery-operated devices, we must consider the data retention energy as well as the data access energy, which means balancing the active-mode, idle-mode, sleep-mode and data-retention mode energies.

This paper reports the first trial to reduce the energy consumption of the whole memory systems in embedded applications with heterogeneous non-volatile storage systems, taking into account the energy for data retention as well as the energy for program execution. Our memory allocation schemes provide a high-level approach, because the time range of interest is several hours to several days. Existing microscopic (transaction level) techniques that reduces energy for program execution are applied together without affecting each other. We use stochastic process models of tasks and user behaviors. For this purpose, we first explore the energy consumption of non-volatile memory devices by cycle-accurate energy measurement and precise energy characterization. We then derive analytical energy models of various combinations of the non-volatile memory devices. Next, we perform a trace-driven energy simulation so that we can propose energy-efficient memory allocation schemes for the heterogenous non-volatile memory systems with given embedded applications and user behaviors. Experiments demonstrate a 10% to 26% saving of memory system energy compared with traditional allocation schemes.

II. Non-volatile memory systems

A. Semiconductor non-volatile memory devices

Magnetic memory devices are naturally non-volatile. The first generation of computers was equipped with magnetic core memory devices, and so were free from data loss during the power-off period. But, current magnetic memory devices are restricted to only the secondary storage because of its slow access time, variable access delay and limited random access capability. Recently, semiconductor non-volatile memory devices such as Flash memory have been announced to replace the magnetic disk for hand-held multimedia products that require mass storage device. Generally, semiconductor non-volatile memory devices outperform magnetic disks in terms of performance, volume, weight and power consumption in providing the capacity required by typical hand-held systems.

There are various types of semiconductor non-volatile memory devices, but we can classify them into two categories: naturally non-volatile memory devices, and battery-backed volatile memory devices. As regards naturally non-volatile memory devices, we will discuss only NOR Flash memory and NAND Flash memory, since EEPROM, Mask ROM, UVEPROM and FRAM are special purpose, non-user programmable, or before-or-after the mass-production stage.

Volatile memory devices are generally superior to non-volatile memory devices in terms of read and write performance. In order to combine this high performance with non-volatility, battery-backed volatile memory devices are widely used as the main memory and non-volatile

TABLE 1
MAJOR DIFFERENCES BETWEEN NON-VOLATILE MEMORY DEVICES.

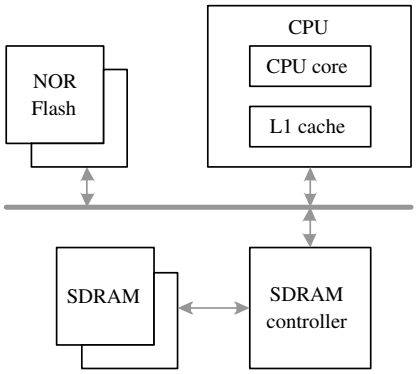
Factor		Battery-backed SDRAM	NOR Flash	NAND Flash
Performance	Read	Excellent	Good	Sufficient
	Write	Excellent	Very poor	Poor
Random access		Possible	Possible	Impossible
XIP		Supported	Supported	Not supported
Capacity		1Gb	256Mb	8Gb
Cost ($/MB)		0.13	0.61	0.16
True non-volatility		No	Yes	Yes
Data retention energy		Needed	Not needed	Not needed

memory system in embedded hand-held systems. Usually, SRAM, asynchronous DRAM and SDRAM can be used as non-volatile memory devices providing a back-up current is supplied during the power-off state. Low-power SRAMs are the most suitable for battery back-up because the back-up current is almost negligible, and no additional control is required except switching the power supply pins to the battery. However, we exclude low-power SRAM from discussion. Low-power SRAM of large capacity is rarely used for consumer products due to high cost, although it is ideal for battery back up. Only SDRAM is a commercially feasible solution for non-volatile main memory system. It provides a large capacity at reasonable cost, despite the double penalty of needing back-up current and refreshing. The self-refresh capability of SDRAM enables battery back-up while all other devices are off, including the memory controller. Asynchronous DRAM is also out of scope; it used to be supplied in low-end PDAs, which are no longer produced. In addition, refresh scheme is much more complex than the SDRAM. So we will discuss only the SDRAM as a battery-backed volatile memory device.

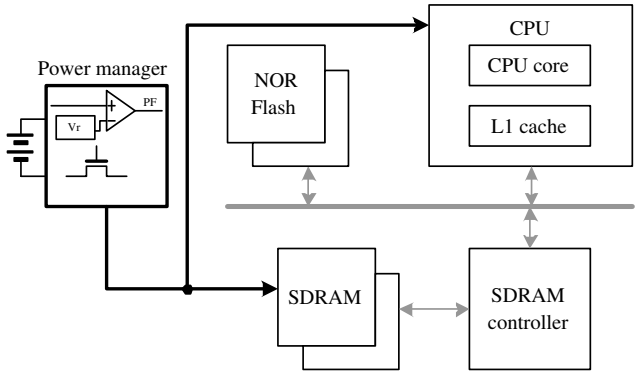
Table 1 summarizes the major differences between three types of non-volatile memory device. The battery-backed SDRAM is superior to the other memory devices in terms of performance, capacity and cost, but it requires additional energy for data retention during power-off period. The NOR Flash memory is the only naturally non-volatile memory device in mass production stage that allows XIP (eXecute In Place). But it is hard to use the NOR Flash memory for non-volatile primary storage due to its poor write performance even though it offers random access capability; it is mostly used for boot-up memory. The NAND Flash memory device has large capacity with low cost, but it is mostly used for secondary storage in CompactFlash, Smart Media and Memory Stick memory products, because it does not support random access both for read and write operations [9].

B. Heterogeneous non-volatile memory system

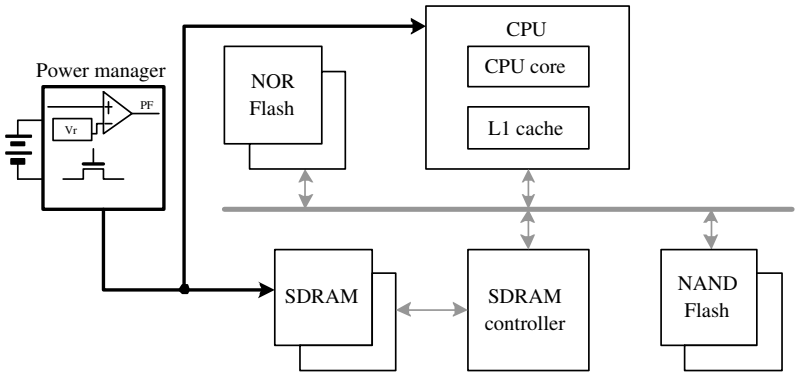
As described in previous sections, modern hand-held devices require non-volatile primary storage. However, no single semiconductor non-volatile device satisfies all the requirements for non-volatile primary storage, and a practical approach is to provide a heterogeneous non-volatile memory system consisting of various types of device which facilitate complementary operation.



(a) SDRAM + NOR Flash



(b) Battery-backed SDRAM + NOR Flash



(c) Battery-backed SDRAM + NOR Flash + NAND Flash

Fig. 2. Typical configurations of the heterogeneous non-volatile memory systems.

Fig. 2 shows three typical types of heterogeneous memory system. Most non-volatile memory systems in embedded systems belong to one of these three categories. Fig. 2(a) shows a non-volatile memory system consisting of a NOR Flash memory and SDRAM. The NOR Flash memory only support an asynchronous interface, which does not provide enough bandwidth for microprocessors operated at a high clock frequency. Thus, they are typically employed in a shadow memory configuration, which means copying the whole contents in the NOR Flash memory to SDRAM before use. The recently introduced Intel Synchronous StrataFlash [10] supports 66MHz burst-mode read operation and has thus brought the NOR Flash memory into XIP. XIP is a more integrated way of using the different types of memory: codes and data are

TABLE 2
SUMMARY OF ENERGY CONSUMPTION OF MEMORY DEVICES (READ OR WRITE: *nJ*/4-BURST OR PAGE ACCESS, STANDBY OR POWER DOWN: *mW*, @66MHZ WITH 32BIT ADDRESS AND DATA BUSES).

Mode	Energy (nJ)/Time (ns)		
	SDRAM	NOR	NAND
Read	70.1/120	33.6/210	$36.9/1.11 \times 10^3$
Write	51.6/120	$12.7 \times 10^3/5.49 \times 10^4$	$233/7.16 \times 10^3$
Erase	NA	$14.1 \times 10^3/6.10 \times 10^4$	$64.4/1.95 \times 10^3$
Standby	45.1mW	0.33mW	0.13mW
Power Down	1.8mW	N/F*	N/F

*N/F: Not Feasible

stored in the NOR Flash memory and the SDRAM respectively. Before power-off or the end of task execution, persistent data must be written back to the NOR Flash memory.

As commercial products cannot justify the cost of large capacity NOR Flash memory, non-volatile mass storage with battery back up of the SDRAM is provided, as shown in Fig. 2(b). The SDRAM is already present as primary storage, so adding a battery back up is a simple way to achieve a large amount of non-volatile storage: but we have to provide the data retention energy. This configuration is basic to modern PDAs.

Fig. 2(c) consists of the battery-backed SDRAM, the NOR Flash memory and the NAND Flash memory. The NAND Flash memory device can only be used for the secondary storage in cooperation with the primary storage. Although a low-cost memory architecture for NAND XIP has recently been reported [9], we exclude it from discussion due to incomplete results as yet. This configuration may not require the data retention energy for the SDRAM when we back up the whole contents in the SDRAM to the NAND Flash memory before power off.

III. ENERGY CHARACTERIZATION OF NON-VOLATILE MEMORY DEVICES

This paper covers three different types of non-volatile memory devices; battery-backed SDRAM, NOR Flash memory, and NAND Flash memory. We derive the energy consumption of non-volatile memory devices during data access by the use of the state-machine-based energy characterization of Joo et al [11], so that we can model both the active and idle energy correctly. We measure the energy consumption of three types of SDRAM: a standard SDRAM, a low-power SDRAM and a mobile SDRAM. The mobile SDRAM is a new arrival that achieves 30% energy reduction in active modes and 80% energy reduction during data retention [12], [13]. On average, 10mW is required to back up two 16-bit standard SDRAM devices while low-power version of the SDRAM requires only half that (5.3mW). Mobile SDRAM is capable of selective self-refresh by bank: that reduces the data retention energy when the data occupies only a part of the device. The energy for data retention is 1.0mW, 1.3mW and 1.8mW for one, two and four banks respectively.

NOR Flash memory is optimized for the read operation and has the least energy density during read operation among the devices mentioned in this paper. But we may not directly compare the energy consumption with that of the SDRAM because the NOR Flash memory exhibits 110ns to 150ns latency in every read operation while the SDRAM has a latency of only 60ns.

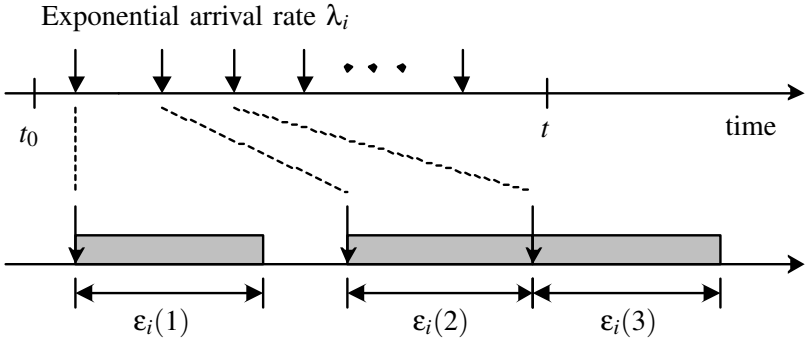


Fig. 3. Exponential arrival model of a task τ_i.

The NAND Flash memory is much more energy efficient than the NOR Flash memory in handling large amount of data. The NAND Flash memory shows fairly good energy density both for read and write operations. This allows frequent update of the NAND Flash memory without significant energy loss. But every read or write operation must be accompanied by the same number of transactions with the SDRAM.

Table 2 summarizes the energy consumption of three different non-volatile memory devices. Although the atomic access sizes are different in each case, we convert each energy and delay value so that it corresponds to the same amount of data. Unlike the other devices, the SDRAM shows a distinct energy consumption during idle mode. The idle-mode energy consumption represents a large proportion of the memory system energy because the SDRAM stays mostly in idle mode even when the cache hit ratio is typical. The energy consumption of the SDRAM during power-down implies energy for data retention.

IV. Energy characterization of heterogeneous non-volatile memory systems

A. System-wide energy model

The energy consumption of an embedded system during the interval $[t_0, t)$ can be expressed as:

$$E_T(t) = \sum_{i=1}^{N} (E_{X_i}(t) + E_{O_i}(t)) + E_R(t). \tag{1}$$

We partition the total system energy into energy for task execution, $E_{X_i}(t)$, energy required for backing up and restoring code and data, $E_{O_i}(t)$, and energy for data retention during power-off, $E_R(t)$. Let us assume that $t_0 = 0$, without loss of generality. The goal is then to find a memory allocation having the minimum $E_T(t)$ for a given task set, $T = (\tau_1, ..., \tau_N)$.

We consider a task model with independent exponential arrival rates, as shown Fig. 3. A task τ_i is a 5-tuple, $(\lambda_i, \varepsilon_i, C_i, D_i, \rho_i)$, where λ_i is the arrival rate, ε_i is the average execution time, C_i is the code size, D_i is the data size, and ρ_i is 0 if the data is read-only, otherwise is 1.

At each invocation j, the task τ_i takes $\varepsilon_i(j)$, and thus its average execution time is given by

$$\varepsilon_i = \frac{1}{k} \sum_{j=1}^{k} \varepsilon_i(j). \tag{2}$$

TABLE 3
SUMMARY OF THE ENERGY-ASSOCIATED COEFFICIENTS (@66MHZ WITH 32BIT ADDRESS AND DATA BUSES).

Device	Coefficient	Value	Description
CPU	P_C	100 (mW)	Average power in active mode
SDRAM	E_{RSD}	70.2 (nJ/4words)	Energy for a 4-burst read operation
	E_{WSD}	51.6 (nJ/4words)	Energy for a 4-burst write operation
	P_{ISD}	45.1 (mW)	Power in idle mode
	E_{FSD}	99.2 (nJ/refresh)	Energy for a refresh operation
	P_{TSD}	1.8 (mW)	Power in self-refresh mode (data retention)
NOR Flash	E_{RNR}	33.6 (nJ/4words)	Energy for a 4-word read operation
	E_{WNR}	107 (μJ/16words)	Energy for a 16-word write operation
NAND Flash	E_{RND}	1.18 (μJ/512bytes)	Energy for a page mode read operation
	E_{WND}	9.51 (μJ/512bytes)	Energy for a page mode write operation

TABLE 4
SUMMARY OF THE DELAY-ASSOCIATED COEFFICIENTS (@66MHZ WITH 32BIT ADDRESS AND DATA BUSES).

Device	Coefficient	Value	Description
SDRAM	T_{RSD}	120 (ns/4words)	Delay for a 4-burst mode read operation
	T_{WSD}	120 (ns/4words)	Delay for a 4-burst mode write operation
NOR Flash	T_{RNR}	210 (ns/4words)	Delay for a 4-word read operation
	T_{WNR}	464 (μs/16words)	Delay for a 16-word write operation
NAND Flash	T_{RND}	35.8 (μs/512bytes)	Delay for a page mode read operation
	T_{WND}	288 (μs/512bytes)	Delay for a page mode write operation

The energy consumed during task execution is given by

$$E_{Xi}(t) = \lambda_i t \cdot (\varepsilon_i P_C + Em_i), \tag{3}$$

where P_C is the power consumption of the CPU, and Em_i is the energy consumption of the memory system during task execution. We derive average active-mode power values of P_C from the data sheets and/or by measurement. Derivation of Em_i and E_{O_i} is dependant on the type of memory devices involving in the memory operation, and we describe them in the following subsections.

The data retention energy, $E_R(t)$, is the battery back-up energy only for the SDRAM and is independent of the amount of data. So $E_R(t)$ is given by

$$E_R(t) = (t - t_A)P_{TSD}, \tag{4}$$

where t_A is the total execution time of the task set T. Since we assume a single microprocessor, t_A is given by

$$t_A = \sum_{i=1}^{N} \lambda_i t \cdot (\varepsilon_i + t_{Li} + t_{Si}). \tag{5}$$

TABLE 5
MEMORY ALLOCATION METHODS FOR THE HETEROGENOUS NON-VOLATILE MEMORY SYSTEMS (C: CODE, D: DATA).

Allocation	SDRAM	NOR	NAND	Constraint for execution
1	C, D			No
2		C, D		No
3			C, D	Load C and D on SDRAM
4	C	D		No
5	D	C		No
6	C		D	Load D on SDRAM
7	D		C	Load C on SDRAM
8		C	D	Load D on SDRAM
9		D	C	Load C on SDRAM

The variables t_{Li} and t_{Si} represent elapsed time for backing-up and restoring the code and data if necessary. Tables 3 and 4 summarize the energy and delay coefficients of the equations in this paper. A static analysis is used to derive t_{Li} and t_{Si} in Equation (5), by the use of Table 2.

The memory energy consumption, Em_i, is greatly dependent on memory allocation and memory access characteristics, which are again determined by each application program. Table 5 shows nine possible memory allocation schemes for heterogeneous non-volatile memory systems. We can summarize the memory energy consumption as three cases, which are described in the following subsections. We obtain the number of instruction cache misses, Mc_i, the number of missed reads of the data cache, Mr_i, and the number of missed writes or flushes of the data cache, Mw_i, by performing bus-functional cycle-accurate simulation. We use average values of Mc_i, Mr_i and Mw_i for one second.

B. Battery-backed SDRAM

The energy consumption of a burst-mode read access, E_{RSD}, occurs when there is a cache miss in the instruction cache and a read miss in the data cache. Energy consumption for a burst-mode write access is denoted by E_{WSD}. The SDRAM consumes a leakage power, P_{ISD}, at all times except in self-refresh mode. In practice, the leakage power in active mode is greater than in idle mode. But we represent the leakage power of the idle, refresh and the access cycles by a single variable, P_{ISD}, for convenience, and the differences between the active mode leakage power and the idle mode leakage power are included in the active mode energy.

The memory system energy consists of energy for instruction fetch, data read and data write, and is given by

$$Em_i = \varepsilon_i \cdot (Mc_i E_{RSD} + Mr_i E_{RSD} + Mw_i E_{WSD} + P_{ISD}) + \lceil \frac{\varepsilon_i}{15.6us} \rceil E_{FSD}. \quad (6)$$

In this configuration, the overhead energy, E_{O_i}, is assigned a value of zero, since all memory accesses are routed to the SDRAM. That can be written:

$$E_{O_i}(t) = 0 \ for \ \forall t. \quad (7)$$

C. Battery-backed SDRAM and NOR Flash memory

As mentioned in previous sections, NOR Flash memory, such as Intel StrataFlash, supports the XIP without noticeable performance degradation. Thus cooperative functioning of the SDRAM and the NOR Flash memory is achieved by having the code and data stored in the NOR Flash memory and the SDRAM respectively. In this case, the memory system energy is given by

$$Em_i = \varepsilon_i \cdot (Mc_i E_{RNR} + Mr_i E_{RSD} + Mw_i E_{WSD} + P_{ISD}) + \lceil \frac{\varepsilon_i}{15.6us} \rceil E_{FSD}. \quad (8)$$

Another alternative is to store not only the code but also the read-only user data, such as MP3 music files or JPEG images, in the NOR Flash memory. The memory system energy is then given by

$$Em_i = \varepsilon_i \cdot (Mc_i E_{RNR} + \eta Mr_i E_{RNR} + (1-\eta) Mr_i E_{RSD} + Mw_i E_{WSD}), \quad (9)$$

where η is the proportion of NOR Flash memory accesses among the missed read transactions from the data cache.

When a read transaction occurs in the NOR Flash memory, instead of in the SDRAM, it requires an additional access time, $T_{RNR} - T_{RSD}$, and thus the energy consumption of the microprocessor increases proportional to $T_{RNR} - T_{RSD}$, and the overhead energy can then be written as

$$E_{O_i}(t) = \varepsilon_i Mc_i \cdot (T_{RNR} - T_{RSD})(P_C + P_{ISD}) + \lceil \frac{\varepsilon_i Mc_i \cdot (T_{RNR} - T_{RSD})}{15.6us} \rceil E_{FSD} \quad (10)$$

or

$$\varepsilon_i \cdot (Mc_i + \eta Mr_i)(T_{RNR} - T_{RSD})(P_C + P_{ISD}) + \lceil \frac{\varepsilon_i \cdot (Mc_i + \eta Mr_i)(T_{RNR} - T_{RSD})}{15.6us} \rceil E_{FSD}. \quad (11)$$

D. Battery-backed SDRAM and NAND Flash memory

As the NAND Flash memory is the secondary storage, it must cooperate with the SDRAM primary storage. There are two choices: either code and data are both stored in the NAND Flash memory or the code is stored in the SDRAM and the data is stored in the NAND Flash memory. The former involves additional time and energy for loading the code and the data, and the latter only involves time and energy for loading the data. After execution, the changed data is transferred back to the NAND Flash memory. We assume that all the data is written back with a proportion of changed content, since there is usually a file system on the NAND Flash memory. (This rather overestimates the migration overhead in later sections, and thus underestimates the energy reduction). After loading the code and data to the SDRAM, all operations are executed on the SDRAM; so Em_i remains the same on a battery-backed memory system which consists of SDRAM only. In this configuration, it requires back up and restore operations to execute the application. This overhead energy consumption can now be written as

$$E_{O_i}(t) = \lambda_i t \cdot ((t_{Li} + t_{Si}) P_C + (El_i + Es_i)). \quad (12)$$

The time for loading the code and the data is given by

$$t_{Li} = \lceil \frac{C_i + D_i}{512\text{bytes}} \rceil T_{RND} + \lceil \frac{C_i + D_i}{4\text{words}} \rceil T_{WSD} \quad (13)$$

TABLE 6
THE CHARACTERISTICS OF THE USER APPLICATIONS (CPU: @200MHZ, MEMORY: @66MHZ, I-CACHE: 4KB, D-CACHE: 8KB).

App.	Mc_i	Mr_i	Mw_i	C_i (KB)	ρ_i
MP3	144,601	277,679	186,469	240	0
MPEG4	46,520	838,088	435,767	392	0
CJPEG	77,924	1,277,528	377,087	98	1
DJPEG	623,341	918,762	458,271	104	0

TABLE 7
EXAMPLES OF THE USER BEHAVIOR PROFILE (MINUTES).

Profile No.	state	t	MP3 ($\lambda_i t$, ε_i)	MPEG4 ($\lambda_i t$, ε_i)	CJPEG ($\lambda_i t$, ε_i)	DJPEG ($\lambda_i t$, ε_i)
1	S	720	7, 11.4	7, 10	5, 6	7, 7.1
	s1	120	1, 20	1, 5	0, 0	2, 5
	s2	180	2, 10	1, 10	1, 10	2, 5
	s3	60	1, 5	2, 5	1, 5	1, 10
	s4	180	1, 10	2, 20	0, 0	0, 0
	s5	60	1, 10	0, 0	2, 5	1, 10
	s6	120	1, 15	1, 5	1, 5	1, 10
	D_i (MB)		19.4	7.6	9.5	8.5

and the time for data saving is given by

$$t_{Si} = \lceil \frac{\rho_i D_i}{4\text{words}} \rceil T_{RSD} + \lceil \frac{\rho_i D_i}{512\text{bytes}} \rceil T_{WND}. \tag{14}$$

In the same way, the energy for loading and saving can be written as

$$El_i = \lceil \frac{C_i + D_i}{512\text{bytes}} \rceil E_{RND} + \lceil \frac{C_i + D_i}{4\text{words}} \rceil E_{WSD} \tag{15}$$

and

$$Es_i = \lceil \frac{\rho_i D_i}{4\text{words}} \rceil E_{RSD} + \lceil \frac{\rho_i D_i}{512\text{bytes}} \rceil E_{WND} \tag{16}$$

respectively.

We also note that, if both code and data are stored in the NAND Flash memory, we do not have to retain the data in the SDRAM, and thus $E_R(t) = 0$; otherwise $E_R(t)$ is the same as in the case where there is only a battery-backed SDRAM.

V. ENERGY-AWARE MEMORY ALLOCATION

A. Experimental environment

Our target system is equipped with a heterogeneous non-volatile memory system consisting of a 64MB mobile SDRAM memory system consisting of two SDRAM devices (K4S56163LC: 4banks $\times$ 4M $\times$ 16bit); a 16MB NOR Flash memory system consisting of

two NOR Flash memory devices (Intel StrataFlash 28F640J3A: 4M $\times$ 16bit); and a 64MB NAND Flash memory system consisting of a NAND Flash memory device (K9K1208U: 64M $\times$ 8bit).

We ran four embedded applications having distinct memory access characteristics: an MP3 decoder, an MPEG4 player, a JPEG compressor and a JPEG de-compressor. Table 6 shows the memory access characteristics of the applications. Also, We composed eight user behavior profiles and emulated 25% to 30% average system utilization (*i.e.*, the duty ratio between power-on and power-off states). Each profile represents the user's behavior by a Markovian model: $S = (s_1, s_2, ..., s_n)$ where s_i is a sub-state of S. Table 7 presents an example user behavior profile. We assume that a user uses a PDA with the four application programs. The one-day average behavior is presented in the second row: $S = (s_1, ..., s_6)$. The six sub-states, $(s_1, ..., s_6)$, that reflect characteristic of behavior change as time elapses. Where $\lambda_i t$ is a number of task invocations, ε_i is an average execution time, and D_i is a data size. The profiles used in our experiment can be classified into two groups. Profiles 1 to 4 are in first group. The users in this group are deemed to be working with a small amount of data, and thus we can avoid allocating code and data to the SDRAM if we do not want to incur the data retention energy. Profiles 1 and 2 have six sub-states whose system utilizations are similar to each other. Profiles 3 and 4 also have six sub-states, but in these profiles, each sub-state has a very different system utilization.

The profiles in the second group represent users with so much data that we must use all available types of memory devices in order to allocate all the data. The other properties of Profiles 5 to 8 are respectively similar to those of Profiles 1 to 4, respectively.

B. Memory allocation constraints

In order to obtain a feasible allocation, the total size of the tasks must not exceed the total size of the memory,

$$\sum_{i=1}^{N} (C_i + D_i) \leq S_{ND} + S_{NR} + S_{SD}. \tag{17}$$

For each memory allocation, we therefore perform acceptance tests:

$$\begin{aligned} \sum_{C_i, D_i \in ND} (C_i + D_i) &\leq S_{ND}, \\ \sum_{C_i, D_i \in NR} (C_i + D_i) &\leq S_{NR}, \\ \sum_{C_i, D_i \in SD} (C_i + D_i) &\leq S_{SD}. \end{aligned} \tag{18}$$

S_{ND}, S_{NR} and S_{SD} are the sizes of the NAND Flash memory, the NOR Flash memory, and the file area of the SDRAM, respectively.

C. Analysis of energy consumption considering application characteristics

The energy consumption of each memory system varies with the application characteristics and the user behaviors, because these in turn affect parameters such as number of task invocations, execution time and memory access patterns. Therefore, a specific allocation may not always be optimal for a given application, and so we performed experiments that take both the application characteristics and the user behaviors into account.

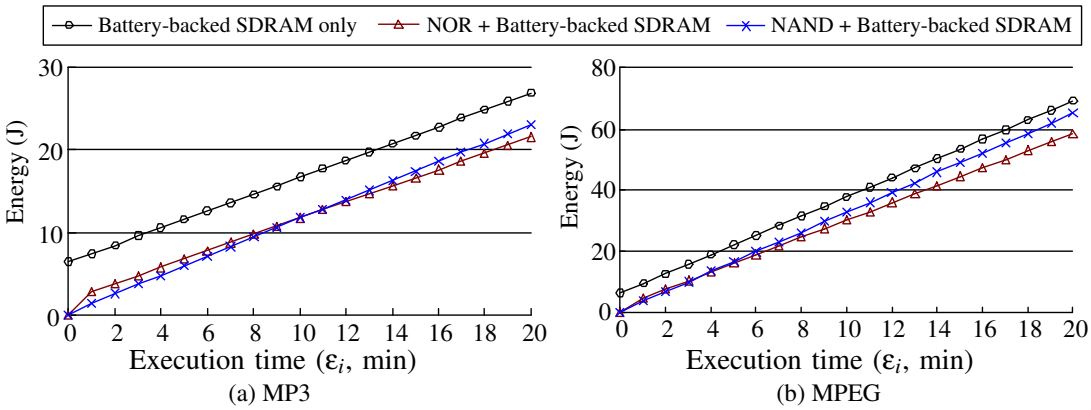


Fig. 4. Energy consumption against execution time (ε_i).

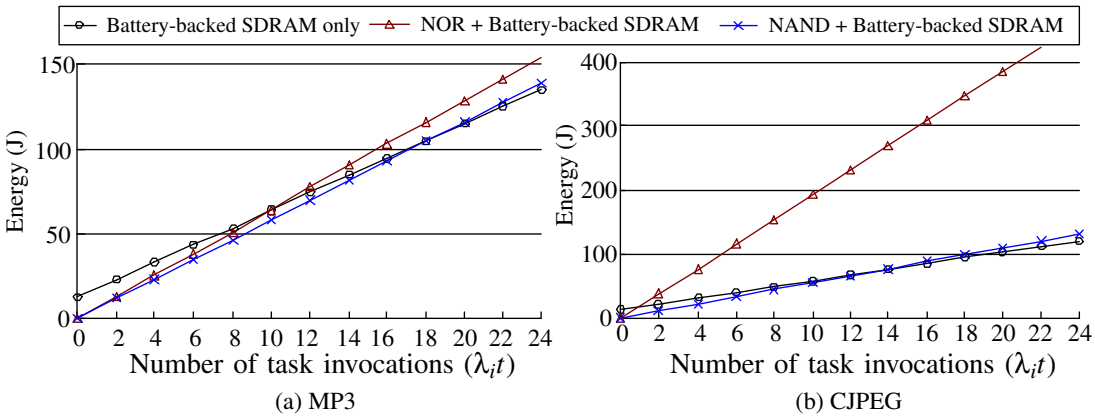


Fig. 5. Energy consumption against number of task invocations ($\lambda_i t$).

To simplify the allocation problem, we first assume a constraint-free environment as shown in Equation (19). That is, we postulate a small amount of code and data that is to be allocated to a single type of memory device within a heterogeneous non-volatile memory environment. In this case,

$$\sum_{i=1}^{N}(C_i + D_i) \quad \leq \quad Min(S_{ND}, S_{NR}, S_{SD}). \tag{19}$$

Therefore, Equations (17) and (18) are always satisfied in the experiments described in this section. We also assume that we are running a single application (*i.e.*, there is no context switch overhead). We simulate eight-hour operation of each memory system by changing the application program, its execution time and the number of invocations. As far as the SDRAM contains valid data, we must back up the data for eight hours regardless of the actual execution time. This causes non-zero initial offset of the battery-backed SDRAM.

Fig. 4 illustrates the energy consumption of MP3 and MPEG applications as functions of the execution time, ε_i. It turns out that the NAND Flash memory system consumes the least energy for $\varepsilon_i \leq 10$ and $\varepsilon_i \leq 3$ in the MP3 and MPEG applications respectively. For the rest of the period, the NOR Flash memory system consumes the least energy in both applications. The two applications have similar energy-efficient memory allocation patterns, except that the MPEG application has a lower crossover point, indicating that the NOR Flash memory system

consumes less energy than the NAND Flash memory system. This is because NOR Flash memory is the most energy-efficient for read operations, and the MPEG application has larger Mc_i and Mr_i values than the MP3 application.

Fig. 5 shows the energy consumption against the number of task invocations. As we increase the number of task invocations, $\lambda_i t$, the energy consumption for backup and restore operations, E_{O_i}, increases both for the NOR and the NAND Flash memory systems, and thus battery-backed SDRAM becomes more attractive. For the MP3 application, the NAND Flash memory system consumes less energy for $\lambda_i t \leq 18$, and the battery-backed SDRAM memory system consumes less energy for the rest of period. The CJPEG application has also similar energy-efficient memory allocation pattern ($\lambda_i t \leq 14$). On the other hand, the NOR Flash memory system consumes the largest energy for most of period in the CJPEG application. This is because NOR Flash memory is the most energy-inefficient for write operations, and the CJPEG application requires heavy write-back burdens, while the MP3 application is mostly read-only.

So far, we have observed the energy consumption of non-volatile memory devices considering target applications and user behaviors, which are sufficiently restricted in scope to achieve energy-optimal allocation. However, if an energy-optimal allocation is not feasible due to the capacity limits of each device and, further, many applications are executed on a system, the complexity of obtaining the energy efficient allocation increases dramatically. For example, we cannot allocate code and data to the NOR flash memory device if it does not have sufficient memory space; and in that case, some data must be allocated to the SDRAM or the NAND Flash memory device. In other words, the problem becomes much more complex if we consider real capacity constrains such as Equations (17) and (18). The Next subsections describe energy-aware static and dynamic memory allocation schemes which consider real constraints encountered in many applications.

D. Static allocation scheme

An exhaustive search to find an optimal allocation requires M^N evaluations for N tasks under M memory configurations. As mentioned in the Introduction, we generally use the traditional memory allocation scheme. For some specific data, such as a boot image, there are no alternatives. However, we can derive the energy-optimal allocation using our proposed scheme, even though there exists several constraints. Static allocation fixes the memory allocation in advance and this initial allocation does not change for time t, the time to the next battery recharge. Given n tasks and three types of non-volatile memory system, we have 9^n possible memory allocation schemes. Exhaustive search requires 9^n trials, which implies a non-polynomial time complexity. However, since we can determine $E_{Xi}(t)$, $E_{Oi}(t)$ and $E_R(t)$ for all i in advance, allocation problem is equivalent to the well-known Knapsack Problem. From an approximate solution to this, we can determine an allocation that is near to optimal. Unfortunately, t_{Li} and t_{Si} are not known before we finish the allocation because the total execution time, t_A, is dependent on t_{Li} and t_{Si}, and $E_R(t)$ is determined by t_A. However, $t_{Li}, t_{Si} \ll \varepsilon_i$ in real cases, which means that we may derive t_A, ignoring t_{Li} and t_{Si}, without noticeable error.

Fig. 6 show a modified Greedy approach to solve the static allocation problem. First, we derive $E_{Xi}(t)$, $E_{Oi}(t)$ and $E_R(t)$ for the nine feasible allocations. The complexity of this equation is only $9N$. Furthermore, we do not have to simulate the energy consumption $9N$ times because Mc_i, Mr_i and Mw_i are independent of the configuration. Finally, we perform the energy-aware memory allocation to derive the optimal allocation for $[0,t)$, using the modified Greedy algorithm.

```
function StaticAllocation(T, S, t)
{
  /* Energy calculation for each configuration */
  for i from 1 to number of tasks
     for j from 1 to number of all possible memory allocation
        calculate E_Xi, Em_i and E_Oi for allocation j;
  /* Allocation */
  do
  {
     for i from 1 to number of non-allocated tasks
     {
        make feasible allocation lists for task i;
        if (number of feasible allocations == 1)
           allocate task i;
     }
     for i from 1 to number of non-allocated tasks
        find an allocation that has minimum E_T(t) among the feasible allocations of task i;
     for i from 1 to number of non-allocated tasks
     {
        find a task that has maximum energy saving among the non-allocated tasks;
        acceptance test for the task with given constraints;
        if (accept)
           allocate the task;
        else
           remove this allocation from the feasible allocation lists of task i;
     }
  }
  Loop until non-allocated list is empty
}
```

Fig. 6. Static allocation algorithm.

Table 8 shows the energy reduction achieved by our energy-aware memory allocation for heterogeneous non-volatile memory systems. The total energy consumption includes energy for the CPU and the memory systems. The four digits in "Alloc." Column are associated with the memory allocations (Table 5) for the MP3, MPEG4, CJPEG and DJPEG applications, in the order of appearance. For example, "6-1-6-6" denotes that the codes of the MP3, CJPEG and DJPEG are allocated in the SDRAM, the data of the MP3, CJPEG and DJPEG are allocated in the NAND Flash memory, and both of the code and data of the MPEG4 are allocated in the SDRAM. We compare the energy consumption of our scheme with a typical allocation scheme for commercial PDAs, which allocates the code and small amount of user data to the SDRAM, extensive user data to the NAND Flash memory, and default applications from the vendor to the NOR Flash memory. The energy gain is 10% to 24% depending on the characteristics of the task and of the user behavior profiles. Although the gain is variable, but it is significant.

E. Dynamic allocation scheme

In static allocation, we have assumed an exponential arrival rate for task invocations. But this is unrealistic because users' behaviors cannot be exactly matched to an exponential arrival model. The shortcomings of static allocation may be overcome by the dynamic adaptation of memory allocation, which means that the initial allocation is changed as the user behavior

```
function DynamicAllocation(T, S, t)
{
   SA = StaticAllocation(T, S, t);
   for i from 1 to number of sub-stats
   {
      DA = StaticAllocation(T, s_i, t'_i);
      calculate the energy consumption of SA and DA for time t'_i;
      calculate the migration cost from SA to DA;
      if (energy saving ≥ migration threshold cost)
      {
         chose DA for s_i;
         SA = DA;
      }
      else
         chose SA for s_i;
   }
}
```

Fig. 7. Dynamic allocation algorithm.

TABLE 8
EXPERIMENTAL RESULTS FOR STATIC AND DYNAMIC ALLOCATION (J).

Profile	Typical allocation			Static allocation					Dynamic allocation			
No.	Total	Mem.	Alloc.	Total	%	Mem.	%	Alloc.	Total	%	Mem.	%
1	579	316	6-6-6-6	517	89.3	241	76.2	8-8-8-9	515	88.9	239	75.6
2	591	321	6-6-6-6	560	94.8	276	86.0	8-8-8-9	552	93.2	269	83.8
3	634	332	6-6-6-6	601	94.9	283	85.2	8-8-8-9	563	88.8	246	74.1
4	648	330	6-6-6-6	584	90.1	251	76.1	8-8-8-9	578	89.2	246	74.5
5	566	285	6-1-6-6	554	97.8	259	90.2	5-8-8-2	553	97.7	259	89.3
6	571	290	6-1-6-6	555	97.2	261	90.0	5-5-8-2	553	96.8	260	89.6
7	719	389	6-1-6-6	702	97.6	354	91.0	8-8-5-8	696	96.8	350	89.9
8	735	400	6-1-6-6	713	97.0	365	91.3	8-8-5-8	702	95.5	354	88.5

changes. We use Markovian models of task execution time in dynamic allocation. The user behavior can be divided into several sub-states as shown in Table 7. Each sub-state shows different number of task invocations and different execution times, t', and thus exhibits different optimal memory allocation. We introduce dynamic allocation scheme as shown in Fig. 7 so that we may achieve better energy optimization reflecting the sub-state change. The proposed dynamic allocation migrates the allocation when the sub-state changes and provided that the total energy saving will be greater than migration threshold cost. All migration decisions have been made off-line (*i.e.*, there is no run-time overhead to decide it).

Table 8 also shows the energy gain achieved by dynamic allocation. The dynamic allocations and migration overheads of each sub-stat are showed in Table 9. If each memory device has enough capacity to store all the applications itself, then allocation become a trade-off between the data retention energy and the migration overhead. Profile 1 exhibits similar energy reductions for static and dynamic allocations, while Profile 3 shows significant further energy

TABLE 9
DYNAMIC ALLOCATION AND MIGRATION COSTS (J).

Profile No.		Sub-state					
		s_1	s_2	s_3	s_4	s_5	s_6
1	Alloc.*	8-8-3-9	8-8-8-9	8-8-3-9	8-8-8-9	8-3-8-9	8-8-8-9
	Cost	N/F**	0.28	0.012	0.28	0.044	1.11
2	Alloc.	8-8-3-9	8-8-8-9	5-8-3-5	8-8-8-3	5-3-5-8	5-8-5-8
	Cost	N/F	0	0.75	3.60	0.88	1.11
3	Alloc.	5-2-5-8	8-2-8-8	8-2-8-8	5-2-5-8	8-2-8-8	8-2-8-8
	Cost	N/F	4.61	0	0.72	4.61	0
4	Alloc.	5-3-5-9	8-8-8-9	8-8-8-9	5-8-5-2	8-8-8-9	8-3-3-9
	Cost	N/F	4.55	0	0.83	3.45	0.06
5	Alloc.	5-8-8-2	8-5-8-2	8-5-8-2	5-8-8-2	-	-
	Cost	N/F	3.49	0	3.30	-	-
6	Alloc.	5-8-5-2	8-5-5-2	8-5-5-2	5-8-5-2	-	-
	Cost	N/F	2.91	0	2.43	-	-
7	Alloc.	5-8-8-8	8-5-8-8	8-8-5-8	8-8-8-5	-	-
	Cost	N/F	2.91	2.35	2.33	-	-
8	Alloc.	5-8-8-8	8-5-8-8	8-8-5-8	8-8-8-5	-	-
	Cost	N/F	2.91	2.35	2.33	-	-

*Refer the Table 5

**N/F: Not Feasible

reduction after dynamic allocation. If the capacity of each memory is not adequate to avoid allocation to the SDRAM, we are forced to expend the data retention energy. In this case, we can save energy during active mode by allocating the tasks to the SDRAM as far as possible, while taking into account the task behaviors and their energy consumption in active mode. Profile 5 shows the reduction of active energy for static and dynamic allocation. Because task behavior is similar in each sub-state, static and dynamic allocations show similar energy reduction ratios. For Profile 8, dynamic allocation saves more energy than static allocation.

VI. CONCLUSIONS

Non-volatile primary storage is now a *de facto* standard for battery-operated systems that require data retention and fast boot. But current technology is still not able to avoid the complementary use of different non-volatile memory devices. Therefore, most hand-held embedded systems are equipped with more than one type of non-volatile memory device such as battery-backed SDRAM, NOR Flash memory or NAND Flash memory. We have therefore proposed the energy-aware memory allocation of code and data in heterogeneous non-volatile memory systems.

In this paper, we have discussed new energy issues raised by the use of non-volatile primary storage in the hand-held applications. Our approach differs fundamentally from traditional allocation in that we consider energy consumption during both power-on and power-off periods. We also address the data retention aiming to extend battery life in actual real-world use. To do

this, we derive analytical energy consumption models of various heterogeneous non-volatile memory systems based on the high-fidelity energy characterization of memory devices. Energy-optimal memory allocation is then achieved by taking into account the active-mode energy, the idle-mode energy and the data retention energy, all of which are in turn affected by the device characteristics as well as the code, the data and the users' behaviors. We present static and dynamic allocation schemes, and analyze their energy savings. Trace-driven simulation results demonstrate that our energy-aware allocation saves up to 26% of memory system energy reduction, compared with a traditional allocation.

REFERENCES

[1] T. Simunic, L. Benini, and G. D. Micheli, Energy-efficient design of battery-powered embedded systems, Proceedings of International Symposium on Low Power Electronics and Design, (**1999**) August 16 – 17, San Diego, USA.

[2] L. Benini, A. Macii, and M. Poncino, A recursive algorithm for low-power memory partitioning, Proceedings of International Symposium on Low Power Electronics and Design, (**2000**) July 25 – 27, Rapallo, Italy.

[3] S. L. Coumeri and D. E. Thomas, An environment for exploring low power memory configurations in system level design, Proceedings of International Conference on Computer Design, (**1999**) October 10 – 13, Austin, USA.

[4] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis, Power aware page allocation, Ninth international conference on Architectural Support for Programming Languages and Operating Systems, (**2000**) November 12 – 15, Cambridge, USA.

[5] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, Scheduler-based DRAM energy management, Proceedings of Design Automation Conference, (**2002**) June 10 - 14, New Orleans, USA.

[6] P. Marchal, D. Bruni, J. Gomez, L.Benini, L.Pinuel, and F. Catthoor, SDRAM-energy-aware memory allocation for dynamic multi-media applications on multi-processor platforms, Prodeedings of the Design, Automation and Test in Europ Conference and Exhibition, (**2003**) March 03 – 04, Munich, Germany.

[7] I. Lee, Y. Choi, Y. Cho, Y. Joo, H. Lim, H. G. Lee, H. Shim, and N. Chang, A web-based energy exploration tool for embedded systems, IEEE Design and Test of Computers, vol. 21, no. 6, pp. 572 – 586, (**2004**) November – December.

[8] M. Newman and J. Hong, A look at power consumption and performance on the 3Com Palm Pilot, UC Berkeley CS252, (**1998**) Spring.

[9] C. Park, J. Seo, S. Bae, H. Kim, S. Kim, and B. Kim, A low-cost memory architecture with NAND XIP for mobile embedded systems, Proceedings of CODES+ISSS, (**2003**) October 1–3, Newport Beach, USA.

[10] Intel Corporation, StrataFlash, Data Sheets, (**2002**) April.

[11] Y. Joo, Y. S. Choi, H. Shim, H. G. Lee, K. Kim, and N. Chang, Energy exploration and reduction of SDRAM memory systems, Proceedings of Design Automation Conference, (**2002**) June 10 - 14, New Orleans, USA.

[12] Infineon Technologies, Mobile-RAM, Application Note, V1.1, (**2002**) February.

[13] Samsung Electronics, Mobile SDRAM (K4S56163LC) and NAND Flash memory (K9K1208U), Data Sheets, (**2002**) December.