♦ Functional-Level Energy Characterization of μC/OS-II and Cache Locking for Energy Saving

Kyungtae Kang, Kyung-Joon Park, and Hongseok Kim

We show how to characterize the energy consumption of individual operating system (OS) functions in the μC/OS-II real time kernel running on an ARM7TDMI-based embedded system. We then derive a strategy for saving energy based on locking more energy-consuming kernel routines of μC/OS-II into the cache and reassigning cache locations to reduce cache contention between frequently invoked kernel functions. The proposed method saves about 37 percent of the energy otherwise consumed by the μC/OS-II kernel, leading to reductions of up to 5.9 percent in the total energy consumption, which includes the energy consumed by the application.
© 2012 Alcatel-Lucent.

Introduction

Embedded computing systems are often based on a single-chip microcomputer which includes a processor and peripheral devices. The design of such a system, especially for mobile applications, must satisfy power and performance requirements as well as aggressive time-to-market constraints. At the same time, the growing complexity of applications makes it increasingly important to manage the software correctly and effectively. This drives application developers to demand run-time support software, in the form of a real time kernel, a run-time library, and device drivers [6, 13]. A real time kernel is particularly important because it provides an effective means of guaranteeing timing constraints, as well as a convenient and efficient development environment [7].

Energy consumption is a critical issue in battery-operated embedded systems, such as sensors and handheld devices. Tan et al. [16] and Dick et al. [6] have shown that the real time operating system (RTOS) accounts for a significant fraction of a system's energy consumption, and has considerable influence on the energy consumption of the other parts of the application software. However, the energy consumption of the RTOS has not been fully investigated because of its high operational complexity. Instead, efforts have largely been focused on dynamic power management (DPM) [3, 17] techniques such as dynamic voltage scaling (DVS) and input/output (I/O) power management [14]. For example, Pillai and Shin [10] proposed a scheduling technique that allocates an interval of central processing unit (CPU) time and a supply of voltage to each task, in order to satisfy its real time constraints while minimizing the total energy consumption. Swaminathan et al. [15] have shown that DVS can be implemented on RTLinux.

More recently, Lu et al. [9] presented a new approach to power reduction using task-level power

Bell Labs Technical Journal 17(1), 219–228 (2012) © 2012 Alcatel-Lucent. Published by Wiley Periodicals, Inc.
Published online in Wiley Online Library (wileyonlinelibrary.com) • DOI: 10.1002/bltj.21532

Panel 1. Abbreviations, Acronyms, and Terms

ADPCM—Adaptive differential pulse code modulation
API—Application programming interface
CPU—Central processing unit
DES—Data Encryption Standard
DPM—Dynamic power management
DVS—Dynamic voltage scaling
EA—Energy aware
HAL—Hardware abstraction layer
I/O—Input/output
NOS—Network Operating System
OS—Operating system
PC—Personal computer
RTLinux—Real Time Linux
RTOS—Real time operating system
SES—Seoul National University energy scanner
SRAM—Static random access memory

management techniques. They attribute power consumption to tasks that require services from hardware components, and then allow the operating system (OS) to control their power states on the basis of the detailed information about the tasks in the OS. They showed that this approach achieves a considerable improvement over existing device-level power management schemes.

However, in most previous research, little attention was focused on the energy consumed by the RTOS itself, which is known to be appreciable [16]. Dick et al. [6] characterized the power consumption of the μC/OS-II [8] real time kernel by running several applications on a Fujitsu SPARClite* processor. They demonstrated that the manner in which the RTOS is used has a significant impact on the power consumption of this system. They also analyzed the effects of RTOS policies on power consumption. The project used an energy analysis framework based on a simulation.

Baynes et al. [2] analyzed the pattern of energy consumption in various operating systems, including μC/OS-II, Ecnidna, and NOS, and demonstrated that a large amount of energy is consumed by idle tasks. Acquaviva et al. [1] characterized the power consumption of the RTOS as independent of the application that is running. They describe the effects of context switching frequency on energy overhead, and relate them to the consequences of thread switching on the cache.

In this paper, we aim to make two contributions to this developing area of work. First, we propose an accurate technique for estimating the energy consumption of the μC/OS-II kernel on ARM7TDMI*-based embedded platforms. To this end, we use the Seoul National University energy scanner (SES) [12] energy measurement tool, which determines the energy used by a CPU core in a real hardware device, and augments this with energy profiles of the cache, memory, and bus obtained by simulations [5, 11]. Second, we show how cache locking can be used to improve the utilization of the cache memory and therefore reduce the energy consumed by μC/OS-II. Cache locking has been shown to be effective in increasing the predictability of task execution in real time systems [18]. However, to the best of our knowledge, there have been no previous attempts to save energy by taking advantage of cache locking.

The rest of this paper is organized as follows. Directly below, we describe an experiment to measure the energy used by μC/OS-II kernel functions. Next, we propose a mechanism to save energy. Following that, we present experimental results that assess the efficiency of the proposed mechanism. Finally, we offer our conclusions.

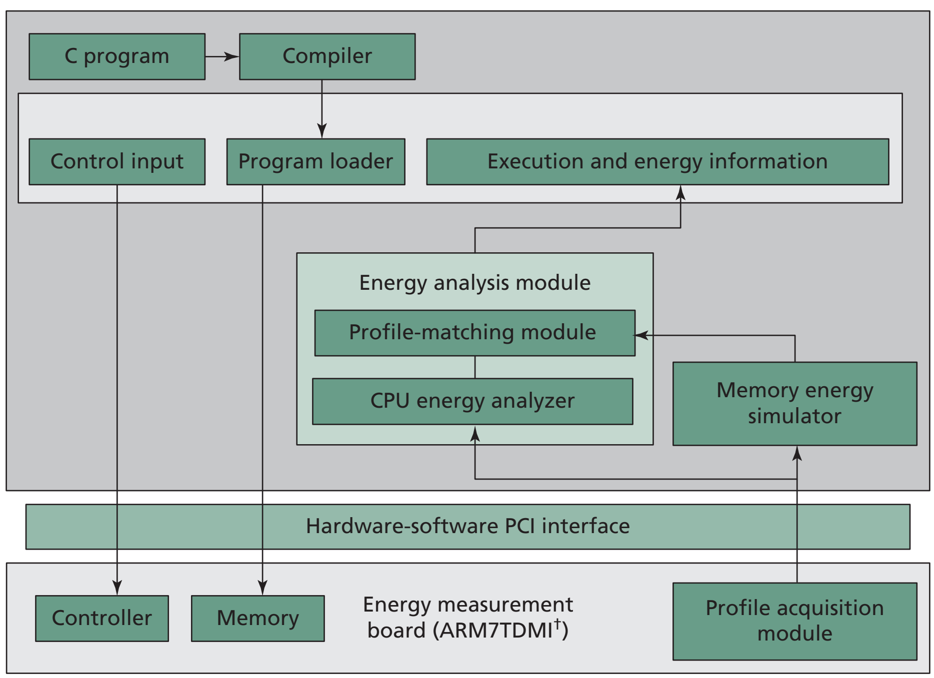
Energy Consumption of μC/OS-II

μC/OS-II is a portable and fully preemptive RTOS which provides priority scheduling, inter-process communication, memory management, interrupt handling, and timer-related services. It was designed for embedded systems and can be easily ported to many different processor architectures. It is also modular, allowing developers to reduce memory consumption by including only the services they need. In addition, μC/OS-II provides a number of system services such as mailboxes, queues, semaphores, fixed-sized memory partitions, and time-related functions. A particular feature of μC/OS-II is that all functions and services are deterministic, allowing energy consumption to be estimated accurately.

Energy Measurement Method

The SES [12] can make cycle-accurate measurements of the energy used by a processor installed on an energy measurement board, which consists of an ARM7TDMI processor core, together with memory, controllers, and a profile acquisition module [5, 11]. The program to be executed on the processor under test is compiled on a host computer, and then transferred to the energy measurement board, where the program is run. A profile of the energy used by the CPU core during program execution is created and transferred to the host personal computer (PC), as shown in **Figure 1**. This profile drives an energy simulation of the cache, memory, and bus to determine the energy consumption of these components. The final estimate of total energy consumption combines the real measurements of the processor with simulation results for the cache, memory, and bus [11].

The challenge of OS energy characterization is to separate the energy consumed by individual OS kernel functions. The SES was not designed to attribute energy consumption to individual functions, and therefore we modified it slightly to measure the energy consumption of each μC/OS-II kernel function. We assigned a block of memory addresses that is not required in normal operation (for example, 0×200000, as shown in **Figure 2**) to an energy measurement register. We then added code that runs when each μC/OS-II kernel function is invoked, and that writes a value specific to that function (for example, 0×02) into the register. When the simulator executes, these values index into an array and subsequent energy consumption is recorded cumulatively in the corresponding array element, also shown in Figure 2. Each array element is a



†Registered trademark of ARM Limited.

CPU—Central processing unit
PCI—Peripheral computer interface

Figure 1.
Cycle-accurate energy profiling using the SES.

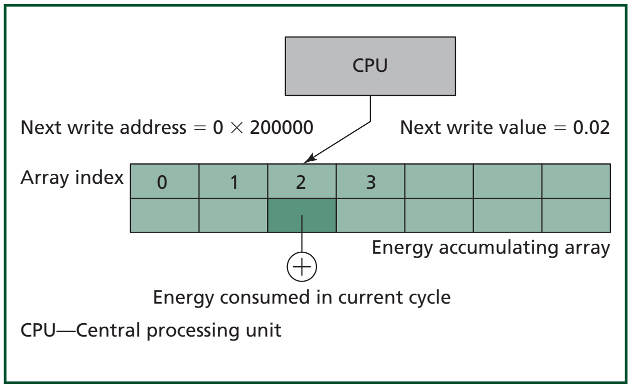


Figure 2.
Proposed method of energy information collection.

vector of values corresponding to the energy consumption of the CPU, the cache memory, and the external memory. Thus the completed array provides energy data which is attributed to every μC/OS-II kernel function and application.

Energy Consumed by μC/OS-II Kernel Functions

We used a set of three tasks for the experiment: Data Encryption Standard (DES) encoding and decoding, adaptive differential pulse code modulation (ADPCM) encoding and decoding, and matrix multiplication. These tasks are scheduled in a round-robin manner, with a timer interrupt period of 20 ms.

Running those task-sets, we measured the energy consumed by the μC/OS-II kernel functions, when 8K two-way associative data and instruction caches, and typical SDRAM main memory systems having 32-bit data width and 64 MB capacity with four Samsung K4S280832B-TC1L devices, are in use. **Figure 3** shows the results on the ratio of the energy consumption as a percentage of the total OS energy consumption and invocation frequency of μC/OS-II kernel functions, which we will use later in determining which function to lock into the cache.

In general, the energy consumption increases as the frequency of each OS function call increases. However, though OS_ENTER_CRITICAL and OS_EXIT_CRITICAL are called most frequently, they consume little energy because they only consist of a few lines of assembly language.

An Energy-Saving Strategy

We will next describe an energy-saving strategy that involves cache locking, focusing first on code

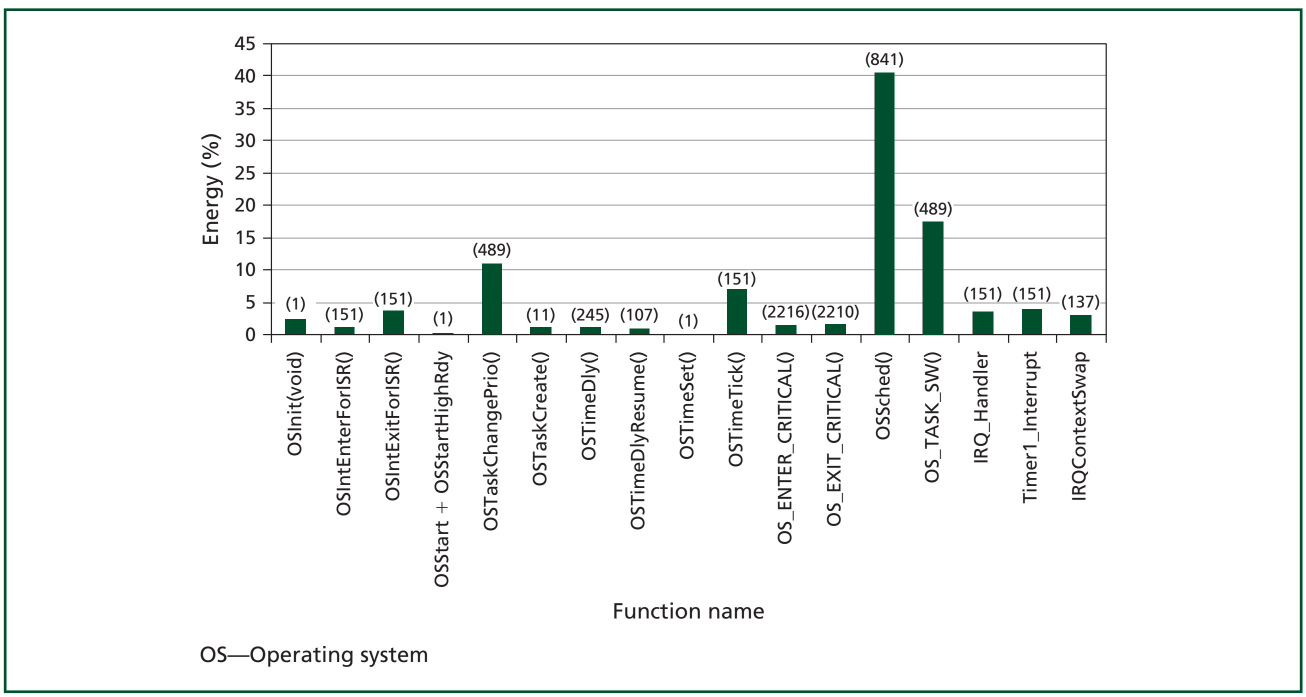


Figure 3.
Ratio of energy consumption as a percentage of total OS energy consumption and frequency of OS kernal functions.

rearrangement techniques and then on the software architecture.

Cache Locking and Code Rearrangement

When a cache miss occurs, the CPU enters a stall cycle during which it consumes energy without performing any instructions. In addition, external memory references are required to fetch the missed instruction and data from external memory, and these consume a considerable amount of energy. External memory references make a major contribution to the energy used by the memory. Thus the cache hit ratio has a significant effect on the energy consumption of the system as a whole.

Task switching is one cause of cache misses in a multitasking system. If there are many tasks to run, there will be frequent task switching and many cache misses. Sometimes a task may flush frequently used OS functions out of the cache repeatedly in an ill-considered way. By locking OS functions such as task switching and timer interrupt into the cache, these functions do not need to be fetched from memory (which increases the cache hit ratio), thus reducing energy consumption. However, this is not a significant drawback if each task is still able to access enough cache memory to exploit the entire locality that is available during the time-slot allocated within a multitasking system.

Obviously, if more than one kernel function is located at the same address when it is in the cache, then these several functions cannot be locked into a cache. To avoid this problem, we changed the binding address of colliding functions so they occupy separate locations.

Software Architecture for Cache Locking

After we ported μC/OS-II to the SES hardware, we modified the SES energy simulator to model the cache locking mechanism. We did this by adding a "locked" bit to the cache structure. If this bit is set, the cache controller does not flush that address.

To control cache locking from an application, the application programming interface (API) functions and the cache management functions were implemented in our modified "energy aware" (EA) API layer, and cache management functions that were used for locking and unlocking the instruction or data cache were implemented in the hardware abstraction layer (HAL), as shown in **Figure 4**. An application can request locking for any μC/OS-II kernel function in the cache using the EA API functions. A new EA manager layer communicates the function addresses to be locked to the memory simulator, where the requests are eventually handled.

In the experiment, we predetermined the set of functions to be locked before creating the tasks to be run. In a practical implementation, we would expect

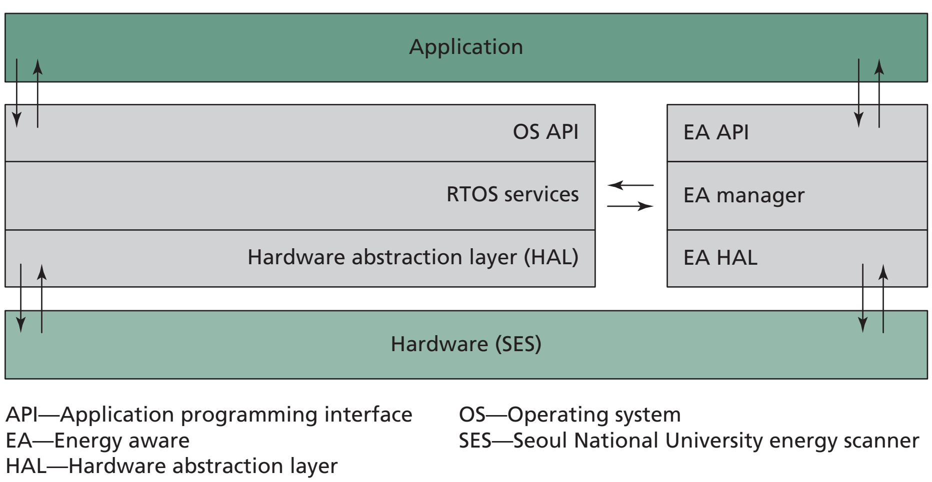


Figure 4.
Implementation of API and cache management functions.

the EA manager to receive run-time energy information from the μC/OS-II kernel, to dynamically control the cache to reduce energy consumption.

Experimental Results

We now analyze the effect of locking some OS kernel routines into the cache. Because the functions OSSched, OS_TASK_SW, OSTimeTick, and IRQContextSwap use a large amount of energy, we designated these functions as candidates for locking. Note that OSTaskChangePrio, which is used to simulate round-robin scheduling, also appears to consume a lot of energy; however we do not consider locking it into the cache because we would not expect it to be used frequently in normal real time operations.

Table I shows the sets of kernel functions which were locked into the cache in our tests. The reference locking set *Ref* contains no functions, and thus none are locked into the cache when this set is used. The locking set *Lock1* contains one function, OSSched, which is locked into the cache. *Lock2* also contains OSSched, as well as the additional function OSTimeTick. In the second reference set, *Ref2*, we changed the binding address of OS_TASK_SW so that it can be locked in the cache with OSSched. Previously, these two functions occupied the same location in the cache, meaning that neither of them could be locked. The locking sets *Lock3* through to *Lock7* retain this change and add the individual locked functions shown in the table.

Table I. Locking sets.

Locking sets	Functions	Locked bytes
Ref	No locking	0
Lock1	OSSched	208
Lock2	Lock1 + OSTimeTick	408
Ref2	Code rearrangement	0
Lock3	OSSched, OS_TASK_SW	300
Lock4	Lock3 + OSTimeTick	500
Lock5	Lock4 + IRQContextSwap	618
Lock6	Lock5 + OSTimeDly	764
Lock7	Lock6 + IRQHandler	816

Table II. Locking sets and their effects on total energy consumption and execution time, for 8K two-way associative instruction and data caches.

Locking sets	Cache hit ratio	Total energy (μ_J)	Energy saving
Ref	0.998714	29024793	0%
Lock1	0.998889	28666201	1.24%
Lock2	0.998921	28602051	1.46%
Ref2	0.999396	27837949	4.09%
Lock3	0.999403	27342815	5.79%
Lock4	0.999426	27300561	5.94%
Lock5	0.999441	27293431	5.97%
Lock6	0.999443	27290074	5.98%
Lock7	0.999433	27299343	5.94%

Table II shows the total energy saved by cache locking. Clearly, locking functions into the cache saves energy. Additionally, the results show that simply rearranging code to avoid cache contention can have a significant effect on energy consumption. However, the results for *Lock5*, *Lock6*, and *Lock7* suggest that there is a limit to how many functions should be locked. Beyond this point, locking-in additional functions will decrease the energy savings because of the reduced cache memory that remains available for the application.

Figure 5 attributes the energy savings to the OS and the application individually. This figure shows that up to 37 percent of the energy used by the OS can be saved by locking OS functions into the cache. However, as we lock more functions, the application increases its energy usage.

Our next step involved changing the cache size for both the data and instruction caches at the same time. **Table III** shows the energy saved by locking set *Lock5* for different cache sizes as compared to the reference locking set, *Ref*. As expected, a larger cache reduces the energy consumption because of improved performance in task execution. For example, the 2K cache is so small that contention occurs frequently, even if there is only one task, and so the energy consumption in the 2K cache is three or four times greater than that of a 4K cache. Locking OS functions into this cache naturally worsens the situation. As a result,

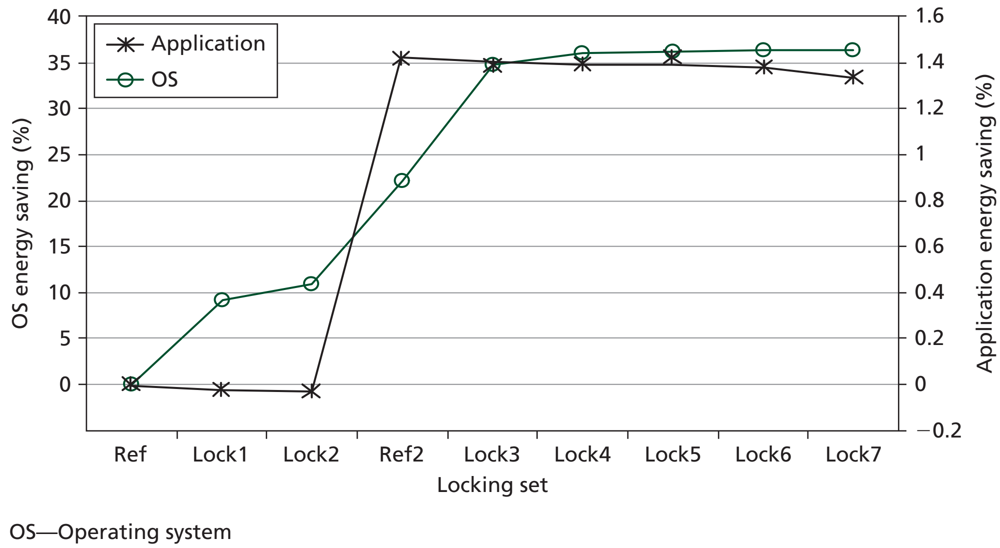


Figure 5.
Energy savings in the OS versus energy savings in the application.

Table III. Total energy consumption using various cache sizes.

Setting	64K	32K	16K	8K	4K	2K
Energy (μJ) – *Ref*	33027	32432	31289	29622	33745	82401
Energy (μJ) – *Lock5*	33028	32400	31079	27846	31608	112164
Energy saving (%)	–0.003	0.099	0.67	5.99	6.33	–36.12

the energy savings driven by using the proposed locking scheme becomes negative when using 2K caches.

When the cache size is increased to 4K or 8K, locking the OS functions produces an improvement of 6.33 percent and 5.99 percent, respectively. These improvements in energy usage are from improved cache performance. However, with a 16K cache, the energy savings is reduced to 0.7 percent. The energy saved by locking the μC/OS-II kernel functions into the cache is minimal with the 32K cache, and with the 64K cache, energy use actually increases. This is because each cache is static random access memory (SRAM)-based and thus the larger the cache, the more energy the cache will consume. Further increases in cache size nullify the advantage of locking but increase the base energy costs.

For the last test, we added two more complex tasks in the task set to increase the workload of the system: a Reed-Solomon encoder/decoder pair that uses the Berlekamp [4] algorithm and a turbo encoder/decoder pair. The energy usages for locking set *Lock5* with different cache sizes and the reference locking set *Ref* are shown in **Table IV**. In general, the cache hit ratio decreases with the higher workload on the system. This suggests that the maximum energy savings achieved by using the proposed cache locking mechanism is higher than originally estimated when the workload on the system is high, as shown in Table IV. We can also see that energy savings begins to reach a saturation point with larger-size caches (here at 32K, unlike 16K in the original workload). Based on these results, we suggest that a more detailed study should be done to correlate the cache hit ratio with energy consumption, thereby gaining a better understanding of the system.

Table IV. Total energy consumption with increased workload.

Setting	64K	32K	16K	8K	4K	2K
Energy (μJ) – *Ref*	35002	34128	32035	35759	41126	91032
Energy (μJ) – *Lock5*	34712	33105	29863	33164	40812	127004
Energy saving (%)	0.828	2.997	6.78	7.25	0.76	–39.52

Conclusion

We have introduced new measurement techniques to characterize the energy consumption of the μC/OS-II operating system. We showed that knowledge of the energy consumption pattern of a μC/OS-II kernel is important for the effectiveness of energy management polices based on cache-locking mechanisms. By locking frequently used OS routines into the cache and rearranging the code to avoid cache contention between these routines, we can achieve a 5.9 percent increase in total energy savings using 8K two-way associative data and instruction caches in the experiment.

μC/OS-II has a small kernel and the cache hit ratio is usually greater than 0.98. We plan to analyze the energy consumption of more extensive real time operating systems such as μClinux and Windows* CE. With these larger systems we expect the cache-locking effect to be more significant.

Acknowledgements

This research was supported in part by the research fund of Hanyang University (HY-2011-N), Ministry of Education, Science and Technology (MEST) and Daegu Gyeongbuk Institute of Science and Technology (DGIST) Convergence Science Center, and Sogang University Research Grant of 2011.

*Trademarks

ARM7TDMI is a registered trademark of ARM Limited.
SPARClite is a registered trademark of SPARC International, Inc.
Windows is a registered trademark of Microsoft Corporation.

References

[1] A. Acquaviva, L. Benini, and B. Riccò, “Energy Characterization of Embedded Real-Time Operating Systems,” ACM SIGARCH Comput. Architecture News, 29:5 (2001), 13–18.

[2] K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, and B. Jacob, “The Performance and Energy Consumption of Embedded Real-Time Operating Systems,” IEEE Trans. Comput., 52:11 (2003), 1454–1469.

[3] L. Benini, A. Bogliolo, and G. De Micheli, “A Survey of Design Techniques for System-Level Dynamic Power Management,” IEEE Trans. Very Large Scale Integration (VLSI) Syst., 8:3 (2000), 299–316.

[4] E. R. Berlekamp, Algebraic Coding Theory, McGraw-Hill, New York, 1968.

[5] N. Chang, K. Kim, and H. G. Lee, “Cycle-Accurate Energy Measurement and Characterization with a Case Study of the ARM7TDMI,” IEEE Trans. Very Large Scale Integration (VLSI) Syst., 10:2 (2002), 146–154.

[6] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha, “Power Analysis of Embedded Operating Systems,” Proc. 37th Design Automation Conf. (DAC ‘00) (Los Angeles, CA, 2000), pp. 312–315.

[7] J. G. Ganssle, “The Challenges of Real-Time Programming,” Embedded Syst. Programming, 11:7 (1997), 20–26.

[8] J. J. Labrosse, MicroC/OS-II: The Real-Time Kernel, CMP Books, Lawrence, KS, 2002.

[9] Y.-H. Lu, L. Benini, and G. De Micheli, “Operating-System Directed Power Reduction,” Proc. Internat. Symp. on Low Power Electron. and Design (ISLPED ‘00) (Rapallo, Ita., 2000), pp. 37–42.

[10] P. Pillai and K. G. Shin, “Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems,” Proc. 18th ACM Symp. on Operating Syst. Principles (SOSP ‘01) (Banff, AB, Can., 2001), pp. 89–102.

[11] H. Shim, Y. Joo, Y. Choi, H. G. Lee, and N. Chang, “Low-Energy Off-Chip SDRAM Memory Systems for Embedded Applications,” ACM Trans. Embedded Comput. Syst., 2:1 (2003), 98–130.

[12] D. Shin, H. Shim, Y. Joo, H.-S. Yun, J. Kim, and N. Chang, “Energy-Monitoring Tool for Low-Power Embedded Programs,” IEEE Design Test Comput., 19:4 (2002), 7–17.

[13] D. Stepner, N. Rajan, and D. Hui, “Embedded Application Design Using a Real-Time OS,” Proc. 36th Design Automation Conf. (DAC ‘99) (New Orleans, LA, 1999), pp. 151–156.

[14] V. Swaminathan, K. Chakrabarty, and S. S. Iyengar, “Dynamic I/O Power Management for Hard Real-Time Systems,” Proc. 9th Internat. Symp. on Hardware/Software Codesign (CODES ‘01) (Copenhagen, Dnk., 2001), pp. 237–242.

[15] V. Swaminathan, C. B. Schweizer, K. Chakrabarty, and A. A. Patel, “Experiences in Implementing an Energy-Driven Task Scheduler in RT-Linux,” Proc. 8th IEEE Real-Time and Embedded Technol. and Applications Symp. (RTAS ‘02) (San Jose, CA, 2002), pp. 229–238.

[16] T. K. Tan, A. Raghunathan, and N. K. Jha, “Energy Macromodeling of Embedded Operating Systems,” ACM Trans. Embedded Comput. Syst., 4:1 (2005), 231–254.

[17] O. S. Unsal and I. Koren, “System-Level Power-Aware Design Techniques in Real-Time Systems,” Proc. IEEE, 91:7 (2003), 1055–1069.

[18] X. Vera, B. Lisper, and J. Xue, “Data Cache Locking for Higher Program Predictability,” ACM SIGMETRICS Perform. Eval. Rev., 31:1 (2003), 272–282.

(Manuscript approved January 2012)

KYUNGTAE KANG is an assistant professor in the Department of Computer Science and Engineering at Hanyang University, Seoul, Korea. He received a B.S. degree in computer science and engineering, followed by M.S. and Ph.D. degrees in electrical engineering and computer science, from Seoul National University, Seoul, Korea. Prior to his tenure at Hanyang University, he spent four years as a postdoctoral research associate at the University of Illinois at Urbana-Champaign. His research on wireless mobile systems has included the development of techniques for saving energy and for improving the quality of service provided by real time embedded applications. His most recent research interest is in the interdisciplinary area of cyber-physical systems. He is a member of the IEEE and the ACM.

KYUNG-JOON PARK is an assistant professor in the Department of Information and Communication Engineering at Daegu Gyeongbuk Institute of Science and Technology (DGIST), Daegu, Korea. He graduated from Seoul Science High School, and received his B.S., M.S., and Ph.D. degrees all from the School of Electrical Engineering and Computer Science (EECS), Seoul National University (SNU), Seoul, Korea. Prior to his tenure at DGIST, he spent four years a postdoctoral research associate in the Department of Computer Science, University of Illinois at Urbana-Champaign (UIUC), Illinois, USA. He also spent a year as a senior engineer at Samsung Electronics, Suwon, Korea. His current research interests include design of medical-grade protocols for wireless healthcare systems, design and analysis of self-adjusting protocols for wireless environments, and modeling and analysis of cyber-physical systems.

HONGSEOK KIM was a member of technical staff in the Alcatel-Lucent Bell Labs Network Performance and Reliability Department in Murray Hill, New Jersey, USA, when this research was submitted for publication in the Bell Labs Technical Journal. He is currently an assistant professor in the Department of Electronic Engineering at Sogang University, Seoul, Korea. He received his B.S. and M.S. degree in electrical engineering from Seoul National University, Seoul, Korea, and his Ph.D. degree in electrical and computer engineering at the University of Texas at Austin. He was a post-doctoral research associate in the Department of Electrical Engineering at Princeton University, Princeton, New Jersey. For five years prior to his tenure at Alcatel-Lucent, he was also a member of technical staff at the Korea Telecom Network Laboratory, where he participated in full service access network (FSAN) and ITU-T SG16 FS-VDSL standardization efforts. His research interests include smart grid, demand response, information middleware, network resource allocation, optimization, network economics, and cross layer design of wireless communication systems. Dr. Kim has authored more than 30 papers in IEEE and ACM journals and conferences. He is also a recipient of the Korea Government Overseas Scholarship.◆