Introduction to the Two Special Issues on Memory

Embedded systems differ from general-purpose systems in two main aspects. First, the two systems are designed for very different purposes: while general-purpose systems run a myriad of unrelated software packages, each with potentially very different performance requirements and dynamic behaviors compared to the rest, embedded systems perform a single function their entire lifetime, and thus execute the same code day in, day out, until the system is discarded or a software upgrade is performed. Second, while performance is the primary (in many instances, the only) quality by which a general-purpose system is judged, optimal embedded-system designs usually represent trade-offs between several goals, including manufacturing costs (e.g., die area, testability, etc.), energy consumption, and performance.

As a result, we see two very different design strategies. General-purpose systems are typically overbuilt; by definition, they are expected by the consumer to run all possible software applications thrown at them. Such systems are designed to handle the average case very well and the worst case at least tolerably well. Were they optimized for any particular task, they would likely become less-than-optimal for all dissimilar tasks; therefore, general-purpose systems are optimized for nothing in particular. They make up for this in raw performance and pure number-crunching. The average notebook computer is capable of performing orders of magnitude more operations per second than that required by a word processor or email client—tasks to which the average notebook is frequently relegated—but because the general-purpose system may be expected to handle virtually anything at any time, it must have the number-crunching ability of a supercomputer, just in case.

On the other hand, because embedded systems are expected to handle only one task, it is not only possible but highly beneficial to optimize an embedded design for its one task. Thus, if general-purpose systems are *overbuilt*, the goal for an embedded system is to be *appropriately built*. In addition, since effort spent at design time is amortized over the life of a product, and many embedded systems have long lifetimes (tens of years), many embedded design houses will expend significant resources up front to optimize a design, employing techniques not generally used in general-purpose systems (e.g., compiler optimizations that require many days or weeks to perform).

One of the most critical resources in embedded systems, one that receives much of the attention of embedded-system engineers, CAD tool designers, compiler writers, and researchers, is the memory resource. Memory, whether SRAM or DRAM, usually represents one of the more costly components in an embedded system, especially if the memory is located on-CPU, since once the CPU is fabricated, the memory size cannot be increased. In nearly all system-on-chip designs, and in many microcontrollers as well, memory accounts for the lion's share of available die area; moreover, memory is one of the primary consumers of energy in a system, both on-CPU and off-CPU. As an example, it was shown that, in many digital signal-processing applications, the memory system consumes more of both energy and die area than the processor datapath. Clearly, this is a resource worth optimizing.

The two special issues cover different aspects of optimizing memory usage in embedded system design. The issues span a wide range of topics, which, we hope, will portray to the reader the enormous breadth of the field of embedded systems. The two issues are comprised of a survey of embedded memories, four hardware-oriented articles, and four compiler- and synthesis-oriented articles.

The Papers

In "Energy-Aware Design of Embedded Memories: A Survey of Technologies, Architecture and Optimization Techniques" (February 2003 issue), authors **Benini**, **Macii**, and **Poncino** present a survey of hardware techniques to optimize embedded memories—those designs in which memory is integrated onto the same IC as the processor core, as is typical in systems-on-chip. They begin with a discussion of memory technologies appropriate for such integration and discuss techniques such as memory partitioning, special-purpose cache organizations, and mechanisms to improve bandwidth, such as compression. They conclude with a discussion of technologies likely to appear in future designs.

Lee, **Kim**, and **Weems** (November 02 issue) present a novel cache organization in their article, "Application-Adaptive Intelligent Cache Memory System." The scheme couples two things—a modified two-way set-associative cache that has small blocks and a fully associative cache that has large blocks—to support variable-length data transfers ranging from the smaller block size to a multiple of the larger block size. Additional hardware maintains information about sequential accesses and keeps contiguous, frequently accessed data together in large logical blocks. The experiments show that both general-purpose applications and streaming multimedia applications benefit from the ability to use multiple block sizes dynamically. The scheme improves both the average memory access time and the average energy consumption over more traditional arrangements.

In "Low-Energy Off-Chip SDRAM Memory Systems for Embedded Applications," authors **Shim**, **Joo**, **Choi**, **Lee**, and **Chang** (February 03 issue) present a comprehensive study of where energy is consumed in an SDRAM system and how to reduce that energy. They present a simulation-driven analytical model that is backed up by experimental hardware verification, and break down the energy consumed by the memory system. They investigate the effects of numerous energy-saving techniques, some well-known, some novel, including different bus encodings, various bit-mapping strategies, and DRAM model controls (e.g., idling the DRAM). They show that, by combining the best of these techniques, one can reduce the energy consumed in an external SDRAM system by roughly a factor of two.

It has been known for some time that locality exists in the data stream. In "Frequent Value Locality and Its Applications," (November 02 issue) authors **Yang** and **Gupta** explore another aspect of this behavior. They find that, at any given moment, a relatively small number of distinct values are found in the memory space. Moreover, this set of values remains fairly constant throughout the application's execution. They investigate two hardware means to exploit this phenomenon. The first is a small-footprint cache to hold this "frequent value" set; the cache is small and is written only occasionally, hence it has very low energy consumption characteristics. The second mechanism is a one-hot encoding scheme to transmit these frequent values over a bus using only one bit of the data bus per value.

Java is being used in an increasing number of embedded and mobile designs, and its garbage collection function presents interesting problems to solve in these environments. Authors **Chen**, **Shetty**, **Kandemir**, **Vijaykrishnan**, **Irwin**, and **Wolczko** (in the November 02 issue), in "Tuning Garbage Collection for Reducing Memory System Energy in an Embedded Java Environment," design tuning the garbage collector to reduce energy consumption when the memory system has multiple banks that can be idled independently of each other. For example, when the garbage collector reclaims dead objects, it can compact the remaining live objects into a minimum number of banks and disable all memory banks that contain no live objects. When the application resumes execution, it will consume less energy than if its data were spread out over all banks.

In "Synthesis of Synchronous Multimedia Applications" (February 03 issue) **Qu** and **Potkonjak** examine the problem of configuring processors and memory hierarchies to minimize area in system-on-chip implementations for multimedia applications. Their approach considers multimedia quality-of-service requirements that include both latency constraints associated with each individual data stream and synchronization constraints between the processing of different streams.

In "Access Pattern-based Memory and Connectivity Architecture Exploration" (February 03 issue), **Grun**, **Dutt**, and **Nicolau** develop design space exploration techniques to customize the memory architecture for a given application. These techniques use access patterns of the targeted application to drive simultaneous configuration of memory modules and the processor/memory connectivity architecture and generate Pareto curves in spaces involving cost, power consumption, and performance.

The absence of hardware caches and use of heterogeneous memory modules (e.g., off-chip vs. on-chip, SRAM vs. DRAM, RAM vs. ROM) are common characteristics of memory systems for embedded applications. **Avissar**, **Barua**, and **Stewart**, in "An Optimal Memory Allocation Scheme for Scratch-Pad Based Embedded Systems" (Nov. 02 issue), tackle the problem of compiler-managed data partitioning for multiple banks of heterogeneous memory. Their partitioning techniques, based on an integer linear programming formulation, are optimal among static approaches and incorporate the distribution of stack data as well as globals.

The paper "System-level Exploration of Association Table Implementations in Telecom Network Applications" (Nov. 02 issue) by **Ykman-Couvreur**, **Lambrecht**, **van der Togt**, **Catthoor**, and **De Man**, addresses memory power optimization for managing association tables of records that are indexed by keys. The optimization method targets a complex design space involving multilayered table organizations with alternative configurations of primitive data structures and key transformation strategies.

Acknowledgments

We would like to thank the authors of all submitted papers, including the authors of papers that could not be included in these special issues. The tight publication schedule did not allow for any extensive revisions, so many good papers could not be included here due to the requests for such revisions by the reviewers. We hope that those papers will appear in future issues of the *ACM Transactions on Embedded Computing Systems*. We would also like to thank the reviewers of the submitted papers for their rapid turn-around and thoughtful comments. This issue would not have been possible without their hard work. We hope that the readers find this issue enlightening and enjoyable.

Bruce Jacob
Shuvra Bhattacharyya

Reviewers

O. Avissar, F. Barat, R. Barua, S. S. Bhattacharyya, E. Brockmeyer, F. Catthoor, C. Chakrabarti, N. Chang, W.-K. Chen, J. Cho, E. Daylight, V. Degalahal, V. Delaluz, A. Dominguez, K. Gallivan, C. Ghez, P. Grun, R. Gupta, T. V. K. Gupta, J. Hu, B. Jacob, M. Jayapala, S. Kambhampati, S. Kim, A. Krishnaswamy, B. Li, L. Li, X. Liu, V. Mooney, Y. Paek, M. Potkonjak, G. Qu, F. Rousseau, R. Sangireddy, E. Sha, M. A. Shalan, W.-T. Shiue, A. Somani, A. Srivastava, D. Stewart, M. Tojima, F. Vahid, D. Verkest, F. Vermeulen, D. Whalley, H. Yasuura, Z. Yi, C. Ykman-Couvreur, X. Yuan, X. Zhang, Y. Zhang.