Energy-Aware Design of Embedded Memories: A Survey of Technologies, Architectures, and Optimization Techniques

LUCA BENINI
DEIS Universitá di Bologna
ALBERTO MACII
Politecnico di Torino
and
MASSIMO PONCINO
Universitá di Verona

Embedded systems are often designed under stringent energy consumption budgets, to limit heat generation and battery size. Since memory systems consume a significant amount of energy to store and to forward data, it is then imperative to balance power consumption and performance in memory system design. Contemporary system design focuses on the trade-off between performance and energy consumption in processing and storage units, as well as in their interconnections. Although memory design is as important as processor design in achieving the desired design objectives, the former topic has received less attention than the latter in the literature. This article centers on one of the most outstanding problems in chip design for embedded applications. It guides the reader through different memory technologies and architectures, and it reviews the most successful strategies for optimizing them in the power/performance plane.

Categories and Subject Descriptors: B.3.1 [**Hardware**]: Memory Structures—*Semiconductor Memories*; C.0 [**Hardware**]: General

General Terms: Design, Performance

Additional Key Words and Phrases: Embedded systems, system-on-a-chip, embedded memories, memories, volatile, nonvolatile, integration

1. INTRODUCTION

In the deep submicron era, hundreds of millions of transistors can be integrated onto a single silicon chip. The most natural question to ask then is: "What do we do with hundreds of millions of transistors?" The correct answer is probably: "It depends on the application," but if forced to give a more direct answer, we should

Authors' addresses: L. Benini, DEIS Universitá di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy; email: lbenini@deis.unibo.it; A. Macii, Politecnico di Torino, Corso Duca degli Abruzzi, 10129 Torino, Italy; email: amacii@athena.polito.it; M. Poncino, Dip. Informatica, Universitá di Verona, Strada le Grazie 15—Cá Vignal 2, 37134 Verona, Italy; email: poncino@sci.univr.it.

respond: “Mostly memory.” In current systems-on-chip (SoCs), memory takes, on a average, more than half of the silicon real-estate. Thus, the term “embedded memories,” commonly used to define memory blocks integrated within logic chips, is probably a misnomer in current technologies, and we should talk about “embedded logic” instead. The primary purpose of this article is to survey the evolution of embedded memories both from a technology and a design standpoint.

Integration of memories and logic onto the same silicon substrate in CMOS technology is a challenging task. For a long time, memories and logic chips have followed different evolutionary paths, and their fabrication technologies have diverged. We should then ask ourselves another question: “Why reverse this trend?” The development of embedded memories has imposed a significant reconvergence effort which is motivated by a fundamental rationale: embedded memories increase performance and reduce power consumption [Watanabe et al. 1997], that is, they greatly improve the power-delay product, an energy-efficiency metric. Logic and memory integration and energy optimization are therefore inextricably linked. Our work attempts to shed some light on this complex relationship.

The survey is organized in three parts. The first provides an up-to-date picture of the current status of embedded memory technologies, and it gives the reader a first grasp on the fundamental characteristics of various types of embedded memories and their design constraints. The second part focuses on energy optimization techniques for the embedded memory classes introduced in the first section. We survey approaches at various levels of abstraction: circuit, logic, architecture, and system level. Finally, the last part looks into the crystal ball and outlines future trends, evolutions and challenges of embedded memories, and their technologies. The main purpose of this section is to give a glimpse of the design problems that will become critical in the near future.

2. EMBEDDED MEMORIES TODAY

In the early days of digital computing, researchers focused on memory size optimization. Memory was expensive, and memory space was a scarce resource. The fast growth of semiconductor technology and the consequent increase of the level of integration have completely changed this picture. Nowadays, the cost per memory bit is extremely low (both for off-chip and embedded memories), and memory size is rarely the main issue. Memory performance and power consumption are now the key challenges in system design.

In this section, we focus on technologies, circuit techniques, and architectures for embedded memories. We first outline the key technological issues raised by the integration of memories and logic. We continue by describing some classes of circuit techniques that can be utilized to design fast and low-energy memories. Finally, we conclude by highlighting a few memory architectural templates which are commonly adopted in modern embedded computing systems.

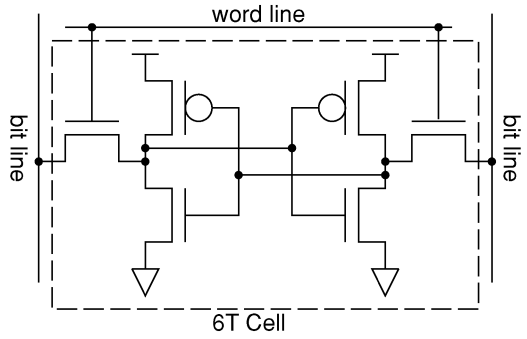

Fig. 1. SRAM cell structure.

2.1 Technologies

The traditional taxonomy for digital memories [Nachtergaele et al. 2001] distinguishes between *read-only* and *read-write* memories. Within the class of read-write memories, two major subclasses are defined, namely *volatile* and *non-volatile*. The classification tree is much deeper, introducing distinctions among different types of access (e.g., sequential, block-based) and storage persistence (e.g., static vs. dynamic). For our purposes, we adopt a coarser, technology-centric taxonomy. We distinguish between the two broad classes of *process-compatible* and *dedicated-process* memories.

Process-compatible memories do not require significant technology enhancements with respect to standard processes for digital logic. Integration of these memories with functional units on a single chip is straightforward from a technology viewpoint, even though they may require design flow modifications.

Dedicated-process memories do require process changes and technology enhancements to be integrated with logic. The significance of these enhancements may vary widely, depending on the type of memory. In some cases (e.g., low-leakage SRAMs), just a few process steps need to be modified. Other types of embedded memories require an almost complete overhaul of the process (e.g., embedded DRAMs). The impact on design flow is significant for all dedicated-memory options.

2.1.1 *Process-Compatible Embedded Memories.* As pointed out before, the fabrication processes for memories and logic have diverged substantially. Among the most common memory types, static random access memories (SRAMs) can be implemented effectively in processes optimized for logic. The elementary cell, shown in Figure 1, contains six CMOS transistors, and it is therefore called *6T cell*. The bit stored in the cell (thanks to the cross-coupled inverters, forming a bistable circuit) is maintained indefinitely as long as the cell is powered (i.e., the memory is volatile). Note that the cell contains both NMOS and PMOS transistors. Needless to say, many variations of the basic cell that use special processes to increase integration density exist [Keitel-Schulz and Wehn 2001].

The characteristic features of embedded SRAMs are low density and high speed. Hence, they are the preferred choice for frequently-accessed, time-critical storage, such as caches and register files.

Example 1. The Emotion Engine [Suzuoki et al. 1999; Kunimatsu et al. 2000] by Sony and Toshiba is a high-performance, computation-intensive SoC that integrates three independent processing cores, a few I/O controllers, specialized coprocessors and many fast SRAM memories for instruction (16 KB) and data (8 KB) caching. Local data storage is also supported by a 16 KB scratchpad SRAM.

Speed is primarily determined by the cross-coupled, regenerative structure of the elementary cell, which actively drives the output during reads. Techniques for speeding up the read operation have been extensively studied in the literature (see, for example, Chandrakasan et al. [2001]), mainly because in most high-performance processors the critical path goes through one or more SRAMs.

Power is also a serious concern for large embedded SRAMs [Chandrakasan et al. 2001], even though speed remains the primary objective. It is important to note that power minimization for SRAMs has a different focus than that for logic. First, SRAMs have a relatively low dynamic power density, because only a very small part of a complex large memory array switches at any given time. Dynamic power in SRAMs is mainly due to the switching of long and heavily loaded bit and word lines, spanning a large number of cells, and readout circuits containing regenerative amplifiers (called *sense amplifiers*) whose gain (and speed) is positively correlated with power. Static power is much more relevant for SRAMs than for standard logic circuits because a very large fraction of the memory is quiescent most of the time. In CMOS technology, static power is primarily due to leakage through OFF-transistors, an undesirable phenomenon that worsens with technology scaling. Recent studies [Frank et al. 2001] show that SRAM leakage power is becoming a serious problem, and several researchers have optimized SRAM circuit components (memory cells, the cell matrix, read and write blocks) for reduced leakage power, with minimal impact on speed [Powell et al. 2001].

Besides SRAMs, read-only memories (ROM) can also be easily implemented in a process-compatible fashion. ROM cells are much smaller than RAM cells (they require only one transistor); hence, they are extremely compact even in standard CMOS technology. Compared to read/write memories, however, ROMs have a limited range of applications, and usually they do not take a large fraction of silicon area.

2.1.2 *Dedicated-Process Embedded Memories.* The main disadvantage of SRAMs is low density, which ultimately limits the amount of fast memory that can be instantiated on chip.

Example 2. The storage requirements of the MPEG4 video codec of Takahashi et al. [2000] are significant (16 Mb). The chip has thus been implemented in an embedded-DRAM process, which allows the integration of three processing cores and DRAM on the same substrate.

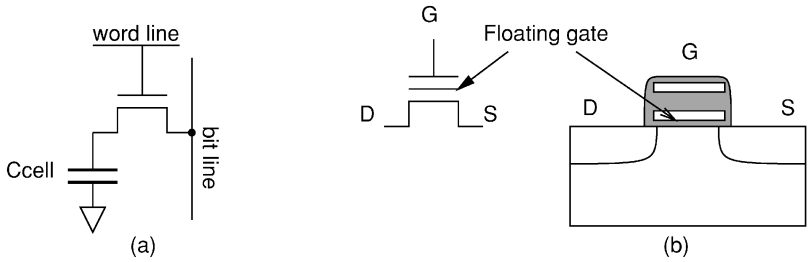

Fig. 2. Dedicated-process memory cells: (a) DRAM; (b) FGMOS.

Another limitation of SRAMs is that they are volatile, and they lose information when power is switched off, whereas many applications require read/write nonvolatile storage. To address these limitations of embedded SRAMs, embedded DRAMs and nonvolatile memories (EEPROM, FLASH) have been developed. Embedded DRAMs feature storage densities up to ten times larger than those of SRAMs, whereas embedded EEPROMs and FLASH memories provide nonvolatile and high-density storage (only slightly less dense than DRAMs). As far as speed is concerned, embedded DRAMs are generally significantly slower than SRAMs, whereas EEPROMs and FLASH memories are only marginally slower. DRAMs and nonvolatile memories greatly enhance the flexibility of SoCs, but unfortunately they are not compatible with standard CMOS technology and they require dedicated processes.

The area-optimized one-transistor DRAM cell (shown in Figure 2(a)) can be in principle realized in standard CMOS technology. However, such a cell would not be usable in practice for two main reasons: (i) logic CMOS transistors are optimized for speed, and they have relatively high leakage currents, which would compromise the lifetime of the information stored as charge in the capacitor; and (ii) the storage capacitor would occupy a very large area, thereby compromising density. On the other hand, the fabrication process for commodity DRAMs is poorly suited to implement logic circuits for several reasons: (i) transistors are optimized for low leakage and have reduced switching speed; (ii) the substrate is optimized for a single transistor type (NMOS) and the dual transistor (PMOS) has poor performance; and (iii) DRAMs are characterized by very regular connectivity and their processes support a limited number of level of metal (two to three) that would be insufficient for logic applications.

To support embedded DRAMs, compromises must be accepted in defining a process suitable for both DRAM cells and logic circuits. Usually, such a process is defined starting either from a logic-optimized process or from a DRAM-optimized process, by adding masks and process steps. Although “hybrid” processes cannot achieve the level of optimization allowed by specialized logic or memory processes, embedded DRAMs are still four to six times more dense than SRAMs.

Embedded nonvolatile memories are assuming an increasingly important role in modern SoCs. Their main function is to provide on-chip nonvolatile storage for configuration information that is frequently read but rarely written over the lifetime of the component. For instance, in processor-based templates, the

executable code that runs on the core processors is often stored in an embedded, nonvolatile memory. This solution has several advantages with respect to external nonvolatile storage. First, the read bandwidth is greatly increased (because the I/O bottleneck is bypassed) and pinout requirements are relaxed. Second, bit-width and memory size can be tailored to application-specific requirements. Finally, embedded memories provide increased security when nonvolatile data should be inaccessible to users (this is the case for proprietary algorithms running on embedded core processors).

Example 3. The single-chip voice recorder and player developed by Borgatti and coauthors [Borgatti et al. 2001] stores recorded audio samples on embedded FLASH memory. The main building blocks are a microcontroller unit (MCU), a speech coder and decoder, and an embedded FLASH memory. A distinguishing feature of the system is the use of a multilevel storage scheme to increase the speech recording capacity of the FLASH. Speech samples are first digitized, then compressed with a simple waveform coding technique (adaptive-differential pulse-code modulation) and finally stored in FLASH memory. The embedded FLASH macro, given its size and the presence of power-hungry mixed-signal components, dominates both area and power dissipation.

The dominant nonvolatile memories today are EEPROM and FLASH [Cappelletti et al. 1999]. Both EEPROM and FLASH are electrically erasable and writable, and they are based on a device known as *floating-gate MOS transistor* (FGMOS), shown in Figure 2(b). The key feature of the FGMOS is that its threshold can be adjusted by changing in a controlled fashion the amount of charge trapped in the floating-gate. Depending on the programmed threshold value, a read to the cell results in a zero or a one. In current technologies, charge trapping is extremely reliable: Current FGMOS-based memories can withstand more than 10^6 rewrites and feature retention times exceeding ten years. Nonvolatile memories differentiate mainly on the write mechanism. EEPROM can be erased and rewritten one cell at a time, whereas FLASH memories can be erased only in blocks (called sectors). On the other hand, EEPROMs trade off finer cell erase granularity with significantly larger cell size (approximately twice that of FLASH). Moreover, FLASH memories can "emulate" single-cell erase at the price of a marginal waste of memory space [Cappelletti et al. 1999].

3. ENERGY-AWARE EMBEDDED MEMORY DESIGN

As stated in the introduction, one of the key motivations for the wide diffusion of embedded memories in SoC design is their inherent energy advantage with respect to external commodity memories. In the previous section, we have described the most common memory technologies in today's design practice, and we have pointed out their pros and cons. Such analysis sets the stage for the energy-efficient design techniques surveyed in this section.

3.1 Circuits

The power consumed by CMOS memories is of two kinds: active power, which is burnt when cells are accessed from outside, and retention power, which is dissipated when the data are maintained in the circuit.

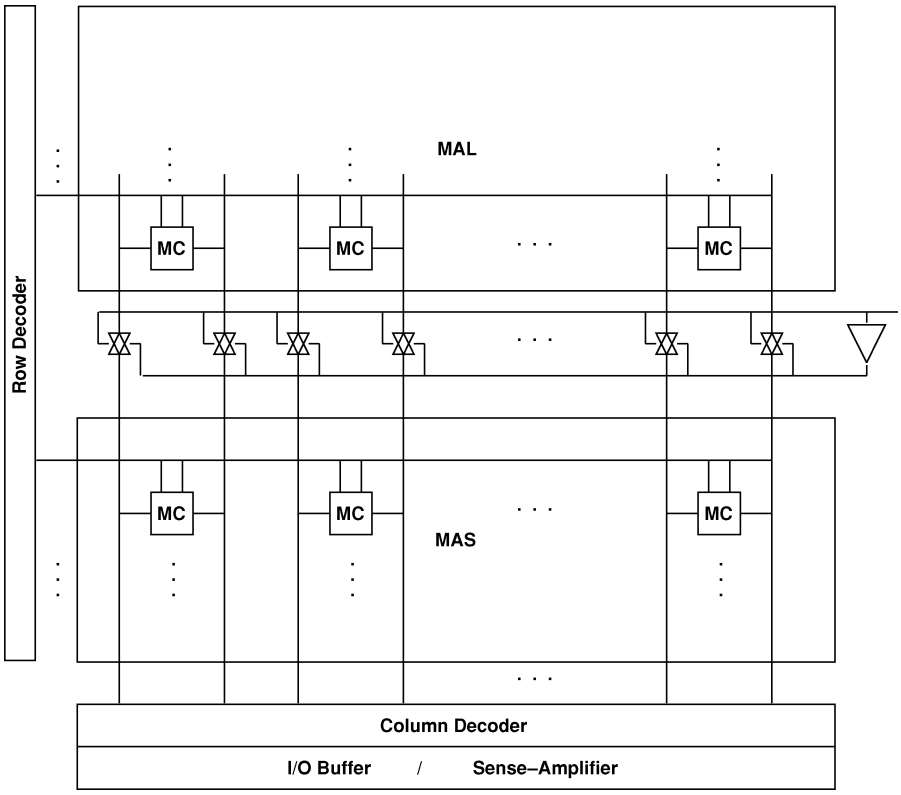

Fig. 3. Bit-line division.

Power consumption of a memory array can be synthetically expressed by equation: $P = V_{dd} \times I_{dd}$, where V_{dd} is the external power supply voltage and I_{dd} is the current of V_{dd}. I_{dd} comprises two main components: (i) the effective current of active or selected cells; and (ii) the effective data retention current of an inactive or nonselected cell. In this section, we describe some circuit techniques for the implementation of SRAM and DRAM modules aiming at the reduction of both active and retention power.

3.1.1 *Active Power Reduction.* Active power for a fixed cycle time can be minimized by reducing the charging capacitance and lowering signal swings on the heavily loaded bit- and word lines in the memory matrix. As memory capacity keeps increasing, so does the number of memory cells connected to each bit- and word line, and the line capacitance grows proportionally [Amrutur and Horowitz 2000].

A practical solution to reduce load capacitance is to divide bit-lines and word lines into several blocks and to activate only one block at a time [Usami and Kawabe 2000].

Example 4. The memory array of Figure 3 is split into two subarrays of different sizes by inserting transfer-gates into each bit-line. The smaller array,

MAS, is connected directly to the sense amplifiers, while the larger array, MAL, is connected to the sense amplifiers via the transfer-gates. When the transfer-gates are turned off, the bit-lines of array MAL are electrically separated from those of array MAS, resulting in discharging only in MAS. Power consumption is then reduced when MAS is accessed.

In DRAMs, the reduction of the number of cells connected to a data line should be accompanied by a decrease of the refresh frequency. In fact, increasing the maximum refresh time yields a reduction of the time the memory is not accessible from outside, due to refresh operations, and consequently the conflicts between normal and refresh operations are reduced [Itoh 1990]. Other techniques that aim at the reduction of the charging capacitance, especially suited for SRAM circuits, exploit I/O line division and predecoding schemes [Mai et al. 1998]. Inserting a predecoding stage between an address buffer and a final decoder optimizes both speed and power.

Regarding operating voltage reductions, many techniques have been proposed in the literature. For DRAM circuits, an effective circuit for reducing the array operating current is based on a half-V_{dd} data line precharging [Kimura et al. 1986; Lu and Chao 1984]. Similar approaches have been proposed for SRAMs [Mai et al. 1998]. Half-V_{dd} precharging cuts in half the data line power of full-V_{dd} precharging, with halved data line voltage swing.

Another solution consists of applying voltage-down conversion (VDC) [Tanaka et al. 1992] in combination with scaled-down devices. This technique provides power reduction, smaller chip area and speed increase, since the VDC has negligibly low current and negligibly small area. Notice that low-voltage operations must be accompanied by high signal-to-noise ratio design [Itoh 1990], because the latter reduces the cell margin with reduced signal charge.

Signal swings can be reduced even without down-scaling V_{dd} by exploiting self-timed pulsed operation. Embodiments of this technique are (i) pulsed bit-lines [Mai et al. 1998]; (ii) pulsed word line [Minato et al. 1984]; and (iii) the pulse operation of the column/sense circuitry [Sasaki et al. 1989]. Pulse-mode circuits are based on a self-timed feedback loop that stops signal swing on a heavily loaded line as soon as it is detected [Amrutur and Horowitz 1998].

3.1.2 *Retention Power Reduction.* In data retention mode, a DRAM chip is not accessed from outside and the data are maintained by the refresh operation. The refresh operation consists of actively reading the data on a word line and restoring them for each of the word lines in order. Hence, data retention power reduction in DRAMs entails both leakage and switching power minimization. Reducing the power of on-chip voltage converters such as VDC, voltage-up (V_{dh}) converter [Itoh et al. 1995], substrate back-bias (V_{bb}) generator, V_{ref} generator [Horiguchi et al. 1991], and half-V_{dd} generator minimize the static current component. The switching current component can be reduced by extending the refresh time and reducing the refresh charge [Itoh et al. 1995].

In low power SRAMs, the static cell leakage current is the main source of the retention power because the peripheral static current is negligible [Itoh et al.

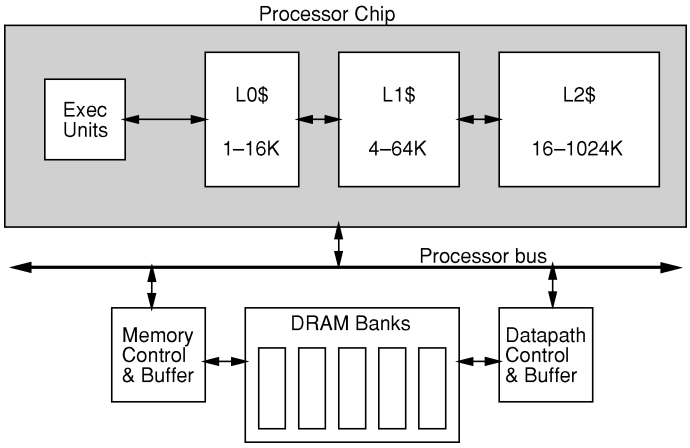

Fig. 4. An example of hierarchical memory architecture.

1995]. On-chip voltage converters have not been extensively used as SRAM cells have a wider voltage margin and a different operating principle. Thus, data retention can be sufficiently reduced solely by memory cell and technology improvements [Agawa et al. 2001; Powell et al. 2001].

3.2 Architectures

The simplest solution to implement a memory is through a flat architecture, which assumes that data are stored in a single, monolithic memory bank. Memory access time, as well as power per access, increase with increasing memory size. In modern systems, memory tends to dominate the overall chip area; thus, flat implementations are no longer applicable, and alternative architectures are adopted.

The most popular organization of the memory subsystem that is found in the majority of today's systems follows a hierarchical template. The storage space is split into different (possibly overlapping) subsets, each of which is implemented at a different hierarchical level. Lower levels in the hierarchy are made of small memories, close and tightly coupled to computation units. Higher hierarchy levels are made of increasingly large memories, far from computation units, and possibly shared. In this context, the concept of distance of memory from computational units represents the effort needed to fetch (or store) a given amount of data from (to) the memory. Effort can be expressed in units of time, or in terms of energy, depending on the cost function of interest.

Example 5. An example of a hierarchical memory architecture, taken from Ko et al. [1998], is shown in Figure 4.

The hierarchy has four levels. Three levels of cache are on the same chip of the execution units. To get high cache hit ratios, the line sizes at upper levels are bigger than those at lower levels. L0 has 16-byte lines, L1 has 32-byte and L3 has 64-byte. Similarly, cache size increases with level. L0 ranges from 1 to 16 KB, L1 from 4 to 64 KB, L2 from 16 to 1024 KB. The last level of memory

hierarchy is the off-chip DRAM, organized in banks, with up to 1024 MB for each bank, when the bank is fully populated. The DRAM access is routed through a memory control/buffer that generates row/column address strobes (RAS/CAS), addresses, and controls sequencing for burst access. Average energy for accessing the L0 cache is approximatively 1.5 nJ, energy for L1 cache access is 3 nJ, the energy for L2 access is 7 nJ. Average energy of a burst transaction to the external DRAM is 127 nJ, which is more than two orders of magnitude larger than the energy for accessing the L0 cache.

Each level of the hierarchy can be designed according to specific templates, so as to meet the given specification constraints. In the following, we introduce two broad classes of architectural options that are normally employed when the target is energy minimization: partitioned (or segmented) memories [Farrahi et al. 1995] and widened memories [Coumeri 1999]. Specialization of these architectures is possible depending on the level of hierarchy at which they need to be used (i.e., cache vs. main memory), and on the chosen process and technology (SRAM vs. DRAM vs. FLASH).

In the following, we only introduce the basic concepts of memory partitioning and widening. Details concerning methodologies and techniques for designing and optimizing memory hierarchy levels that comply with these classes are the subject of the next section.

3.2.1 *Partitioned Memories.* The principle in memory partitioning is to subdivide the address space into many blocks and to map blocks to different physical memory banks that can be enabled and disabled independently. Energy for memory access is reduced when memory banks are small. On the other hand, an excessively large number of small banks is highly area inefficient, and imposes a severe wiring overhead, which tends to increase communication power. For this reason, the number of memory banks should be constrained.

Example 6. An example of memory partitioning is provided in Coumeri [1999]. The left-hand side of Figure 5 shows a monolithic implementation of an array of 256 words of 16 bits each. The right-hand side of the figure depicts the same memory array mapped onto two memory banks of 128 words each. The chip-enable lines (`CE`) of the two banks are controlled by a 2-1 decoder which, in turn, is driven by the line of the address bus which differentiates between words in bank M1 and words in bank M2 (i.e., line `Addr[7]`). To ensure that only one bank at a time is accessed, address lines `Addr[0-6]` are buffered.

Clearly, partitioning can be applied iteratively, although overhead due to control logic and extra wiring may tend to off-set the actual savings introduced by accessing smaller memory banks.

3.2.2 *Widened Memories.* The idea on which widened memory architectures are based is that multiple words can be read during one access. Output multiplexing is then used to select the single word for which an access was actually requested. Using this architecture is advantageous when memory words are accessed in sequence: After the first word is transferred, the second can be accessed by way of the output multiplexer without further performing a read

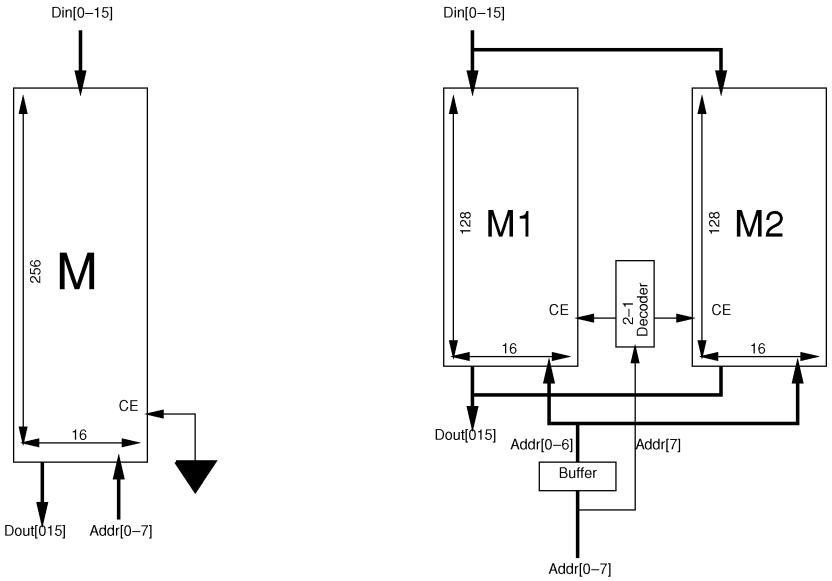

Fig. 5. An example of partitioned memory.

operation. The overhead introduced by this architecture is limited to the output multiplexer and the extra wiring.

More complex is the situation for writing information in a widened memory, as only multiple words can be written to memory per each access, and they may not all be available simultaneously. One option to solve the problem consists of resorting to a "latched-write" scheme. Assuming the memory is widened by a factor of 2, when the writing of one memory word is required, it is actually written into a latch. When the next consecutive word to be stored becomes available, the content of the latch, as well as the datum available on the data-bus, are written to memory.

Clearly, the latched-write architecture only works when data words are written consecutively in memory. If this is not the case, one may adopt a "feed-back write" scheme, whose implementation is more hardware demanding.

Example 7. Memory widening can be considered as a special case of memory paging in which size of memory pages is small [Prince 1997].

3.3 Optimization Techniques

In Section 2, we discussed technologies, circuits and architectures that are commonly adopted in energy-efficient memory implementations. In this section, we focus our attention on optimization aspects; in particular, we describe techniques and methodologies presented in the recent literature for the optimization of energy consumption of an hierarchical memory subsystem.

The objective of energy-efficient memory design is to minimize the overall energy cost for accessing memory, within performance and size constraints. As mentioned in Section 3.2, hierarchical organizations reduce memory energy

by enabling accesses to smaller banks. In addition, as most applications are characterized by nonuniform memory access patterns (i.e., a few locations are accessed with high frequency, whereas most locations are accessed a few times), mapping frequently accessed locations to low hierarchy levels helps in reducing the total energy. The dependency of memory energy on two independent variables (the access cost and the access profile) allows to model it as the product of the number of memory accesses by the cost of each individual access. Formally, if N is the number of accesses:

$$E_{mem} = \sum_{i=1}^{N} Cost(i). \tag{1}$$

In the formula, $Cost(i)$ lumps the effective cost of an access due to the memory organization and the cost of the physical access given by the technology.

Equation (1) exposes the two quantities we can consider to reduce the energy consumption of a memory system. They are also independent, thus allowing us to classify memory optimization techniques in three categories, according to which the variable is kept fixed:

(1) Given a memory access profile (N fixed), reduce the effective access cost by producing a customized memory hierarchy for that access pattern. We call this class of approaches *hardware-centric*.
(2) Given a fixed memory architecture ($Cost(i)$ fixed), modify the storage requirements and access patterns of the target computation to optimally match the given hierarchy. We call this class of approaches *software-centric*.
(3) Concurrently optimize memory access patterns and the relative access cost. We call this class of approaches *hardware–software memory codesign*.

Figure 6 shows a taxonomy of the approaches to low-power memory design existing in the literature; the first level of the taxonomy is represented by the three classes above.

Considering the scope of this article, in the following, we focus on hardware-centric optimizations, leaving aside software-centric approaches and combined hardware-software optimizations. The interested reader may refer to Benini and De Micheli [2000] and Panda et al. [2001] for more details on the latter techniques. We start by adding some insight on the relation between performance and energy in memory design; then, we provide a detailed survey of existing hardware-oriented optimizations.

3.4 Performance vs. Energy

When comparing latency and energy per access in a memory hierarchy, we observe that these two cost metrics have a similar behavior, namely, they both increase as we move from low to high hierarchy levels. Hence, we might be lead to the conclusion that a low-latency memory architecture is also a low-power one, and thus optimizing memory performance implies power optimization. Unfortunately, this conclusion is often incorrect for two main reasons.

First, even though both power and performance increase with memory size and memory hierarchy levels, they are not guaranteed to increase in the same

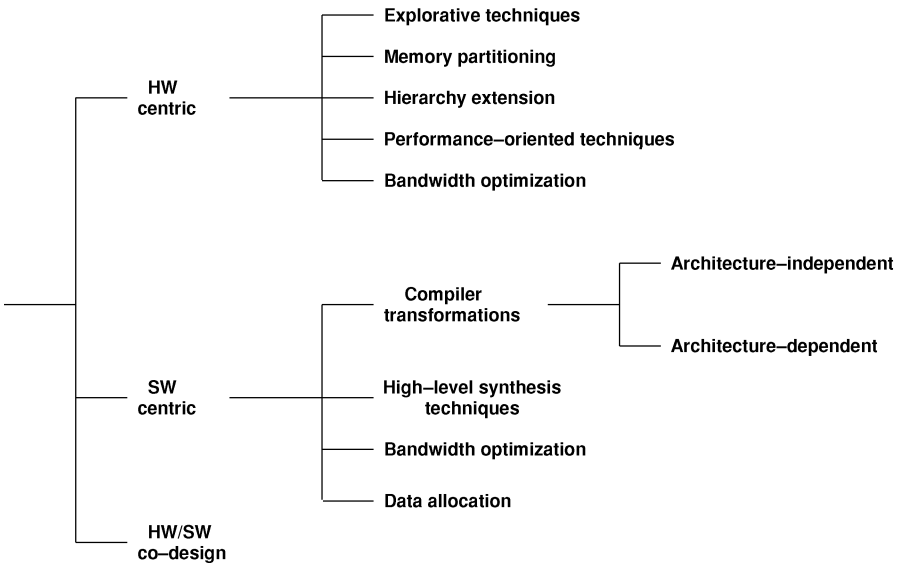

Fig. 6. A taxonomy of memory optimization techniques.

way. Second, and more important, performance is a worst-case quantity (i.e., intensive), whereas power is an average-case quantity (i.e., extensive). Thus, memory performance can be improved by removing a memory bottleneck on a critical computation, but this may be harmful for power consumption, because we need to consider the impact of a memory architecture on all memory accesses, not just the critical ones. The following example, taken from Shiue and Chakrabarti [1999], shows how energy and performance can be contrasting quantities.

Example 8. Consider a two-level memory hierarchy consisting of an on-chip cache and an off-chip main memory, characterized by the following parameters:

—Cache size, ranging from 16 bytes to 8 KB (in powers of two);
—cache line size, from 4 to 32, in powers of two;
—associativity (1, 2, 4, and 8);
—off-chip memory size, from 2 Mb SRAM, to 16 Mb SRAM.

The exhaustive exploration of the cache organization for minimum energy for an MPEG decoding application resulted in an energy-optimal cache organization with cache size 64 bytes, line size 4 bytes, eight-way set associative. For this organization, total memory energy is 293 μJ and execution time 142,000 cycles. Conversely, exploration for maximum performance yields a cache size of 512 bytes, line size 16 bytes, eight-way set associative. In this case, the execution time is reduced to 121,000 cycles, but energy becomes 1,110 μJ.

The example shows that energy cannot be simply reduced as a by-product of performance optimization. In some cases, however, solutions originally devised

for performance optimization are also beneficial in terms of energy. For example, this is the case for techniques that improve locality of access, which result in a reduction of the effective cost of a memory access. Some of the techniques that will be reviewed in this section are strictly performance-oriented solutions, as they also positively affect energy consumption.

3.5 Hardware-Centric Approaches

Solutions in this class assume systems containing one (or more) core processor(s) and an application (or an application mix), and explore the memory hierarchy design space to find the organization that best matches both processor(s) and application(s).

3.5.1 *Exploration Techniques.* One first category of techniques is intrinsically explorative. They exploit the fact that the memory design space can be parameterized and discretized to allow exhaustive or near-exhaustive search. Most efforts in this area postulate a memory hierarchy with one or more levels of caching, and in some cases an off-chip memory. A finite number of cache sizes and cache organization options are considered (e.g., degree of associativity, line size, cache replacement policy), as well as different off-chip memory alternatives (e.g., number of ports, available memory cuts). The best memory organization is obtained by simulating the workload for all possible alternative architectures.

The various contributions in the literature differ in the number of hierarchy levels that are covered by the exploration, or by the number of available dimensions in the design space. Su and Despain [1995], Kamble and Ghose [1997], Ko et al. [1998], Bahar et al. [1998], Shiue and Chakrabarti [1999] focus on cache memories. Coumeri and Thomas [1998] analyze embedded SRAMs, while Juan et al. [1997] study translation look-aside buffers.

Example 8 shows an instance of a design space and the result of the corresponding exploration. An advantage of the explorative techniques is that they allow concurrent evaluation of multiple cost functions such as performance and area, which can be used as constraints without any specific effort during exploration. On the other hand, the main shortcoming of this class of methods is that they only provide *a posteriori* insight. Furthermore, in order to limit the number of simulations, the granularity of the design space they can span is relatively coarse. Therefore, explorative techniques are more effective for general-purpose systems in which the uncertainty on the application mix imposes the adoption of robust (yet less flexible) memory architectures, which can adapt fairly well to different workloads.

3.5.2 *Memory Partitioning.* As already discussed in Section 3.2, within a given memory hierarchy level, power can be reduced by memory partitioning techniques. Memory partitioning by itself is a typical performance-oriented solution, because it reduces latency by accessing smaller memory blocks, and energy may be reduced only for some specific access patterns (see Example 8).

What actually gives this class of techniques an explicit low-power potential is the opportunity of selectively shutting down memory banks that are not

accessed. However, arbitrarily fine partitioning is prevented by the fact that introducing too many banks is area inefficient and it imposes a severe wiring overhead, which tends to increase communication power.

Partitioning-based techniques proposed in the literature differ in several aspects. First, the hierarchy level targeted for partitioning (from register files to off-chip memories). Second, the "type" of partitioning: *physical* partitioning strictly maps the address space onto different, nonoverlapping memory blocks; and *logical* partitioning allows some redundancy in the various blocks of the partition, with the possibility of having addresses that are replicated several times in the same level of hierarchy.

Physical partitioning of embedded memories has been analyzed by several authors. *Region-based* caching [Lee and Tyson 2000] proposes the use of separate caches for stack data and global data, and a main cache for all other accesses. Clearly, nonaccessed cache modules can be disabled for energy saving.

Benini et al. [2002] describe an application-driven partitioning of on-chip SRAMs, which is based on a recursive formulation. Locality is exploited to synthesize a custom memory hierarchy, consisting of independently accessible memory banks. The method explicitly accounts for the overhead induced by the partitioning. In fact, the partitioning engine is integrated inside a comprehensive framework that links the partitioning algorithm to the physical design phase.

Logical partitioning was proposed by González et al. [1995], in which the on-chip cache is split into a *spatial* and into a *temporal* cache to store data with high spatial and temporal correlation, respectively. This approach relies on a dynamic prediction mechanism that can be realized without modification to the application code by means of a prediction buffer. A similar idea is proposed by Milutinovic et al. [1996], in which a split spatial/temporal cache with different line sizes is used.

The idea behind these two approaches has been extended by Grun et al. [2001] in the context of embedded systems for energy optimization. In their approach, data are statically mapped onto either cache, thanks to the high predictability of the access profiles in embedded programs. Depending on the application, data might be duplicated and thus be mapped to both caches.

Another class of logical partitioning techniques falls into the generic scheme of Figure 7. Buffers are put along the I-cache and/or the D-cache, to realize some form of cache parallelization. Such schemes can be regarded as a partitioning solution because the buffers and the caches are actually part of the same level of hierarchy.

In this architecture, data and instructions are explicitly replicated, and redundancy is an intrinsic feature of these approaches. Energy is saved by reducing the average cost of a memory access by increasing the cache hit ratio.

Jouppi [1990] proposes the use of the buffer as a *victim cache* that is accessed on a main cache miss. In the case of a buffer hit, the line is moved to the cache and returned to the CPU, while the replaced line in the cache is moved to the victim cache. In the case of a buffer miss, the lower level of hierarchy is accessed and the fetched datum is copied into the main cache as well, while the replaced

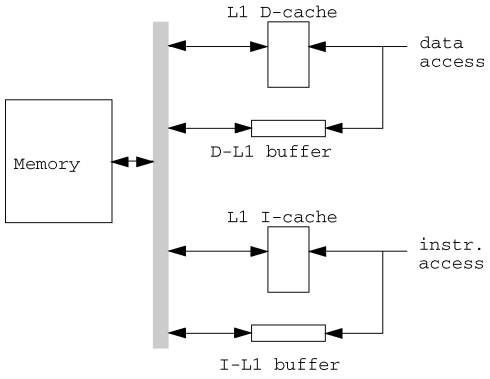

Fig. 7. Using buffers alongside caches.

line in the cache is moved to the victim cache. In practice, the victim cache serves as an over-full buffer for the main cache.

A similar approach has been used by Bahar et al. [1995], in which buffers are used as *speculative* buffers: every cache access is marked with a *confidence level*, obtained by the processor state; and the main cache contains misses with a high confidence level, whereas the buffers contain those with a low confidence level.

Fisk and Bahar [1999] proposed the use of a small associative buffer (e.g., 32 entries) in parallel to the L1-cache (called the *noncritical buffer*), used to "protect" the cache from being filled with noncritical (i.e., potentially polluting) data. Noncritical data are identified at run-time by the issue rate of the core.

A slightly different approach is used by Walsh and Board [1995] and John and Subramanian [1997], in which the data to be stored in the main cache are *filtered* through a small highly-associative cache close to the L1-cache (the *annex cache*). Unlike the victim cache (in which the data are kept before their disposal), the annex cache stores the data read from memory, which are copied into the main cache only on subsequent references to those data. This idea of filtering cache accesses to reduce writes that are very likely to cause misses has been refined in Tang et al. [2002], in which the side buffers are selectively disabled to save additional energy.

3.5.3 *Extending the Memory Hierarchy.* Memory partitioning extends the "width" of the memory hierarchy by splitting, with or without replication, a given hierarchy level. An alternative option is offered by modifying its "depth," that is, the number of hierarchy levels. This solution does not just imply the straightforward addition of extra levels of cache memories.

A first class of techniques is based on the insertion of *ad hoc* memories between existing hierarchy levels. These methods exploit the strong locality of reference of the instruction flow. In most programs, a large fraction of the execution time is spent in a very small section of the executable code, namely, a few critical loops.

Predecoded instruction buffers [Bajwa et al. 1998] store instructions in critical loops in a predecoded fashion, thereby decreasing both fetch and decode energy. The targets of the proposed optimization technique are DSP applications,

such as image or signal processing, in which cyclic execution of a few instructions dominates.

Loop caches [Kin et al. 1997] are used to store the most frequently executed instructions (typically contained in small loops) and are used as a level-0 cache. Given their very small sizes (128–256 bytes), loop caches negatively affect the miss rate, but decrease the overall energy. Compared with pre-decoded instruction buffers, loop caches are less energy efficient for programs with very high locality, but are more flexible.

Another class of solutions is based on the replacement of one or more levels of caches with more energy-efficient memory structures. Such structures are usually called *scratch-pad* buffers, and are used to *statically* store a portion of the off-chip memory (or a memory farther in the hierarchy). This is in sharp contrast with caches, that *dynamically* map a set of noncontiguous sets of addresses from a slower, larger memory. These techniques are particularly effective in application-specific systems, which run an application mix whose memory profile can be studied *a priori*, thus providing intuitive candidates for the addresses to be put into the buffer.

The work by Panda and Dutt [1999] presents a first attempt in this direction. Their architecture is concerned with data addresses, and maps a fixed set of contiguous addresses of the off-chip memory onto a on-chip SRAM buffer. Their architecture also includes a first-level cache that dynamically maps addresses of the address space of the off-chip memory that is not mapped onto the buffer. The same authors propose a solution to the problem of which subset of the data has to be mapped onto the buffer, for some specific classes of applications that are suitable for contiguous mapping (e.g., array-intensive programs) [Panda et al. 2001].

A more flexible solution, in which the buffer is named *application-specific memory* (ASM), is proposed in the literature; its major difference with respect to a conventional buffer is that an ASM appears to be distributed over many noncontiguous locations. Hence, it overcomes the limitation of standard buffers, which leave to the designer the burden of adopting a careful coding style or implementing program transformations to guarantee that the most frequently accessed data are stored in a small number of contiguous memory addresses or both. The authors also provide different architectural configurations that allow to combine the ASM with a cache or with off-chip memories or both.

3.5.4 *Performance-Oriented Techniques.* Architectural solutions explicitly for performance optimization may also help in reducing energy. Techniques that attempt to reduce the cache-miss rate by proper cache design are an example of such solutions; in fact, cache-miss reduction decreases the effective average cost of accessing memory data.

Conflict misses can be reduced by using more than a single mapping function for placing data into sets. This scheme aims at obtaining the benefits of an increased associativity without actually increasing it. The direct-mapped *column-associative* cache [Agarwal and Pudar 1993] employs two different mapping schemes: The first is used when a cache access is issued, whereas the second is applied only in case of a miss in the first attempt. In practice, this

cache behaves like a two-way associative cache with sequential search. The *skewed-associative* cache [Seznac 1993] is an alternative option. A two-way associative cache can achieve the equivalent hit performance of a four-way cache by employing different mapping functions for each way.

3.5.5 *Bandwidth Optimization.* Bandwidth is defined as the *width* of the processor-to-memory interface (PMI), as opposed to memory latency, which is a measure of the *depth* of the PMI. It has been pointed out in Burger [1998] that memory bandwidth is becoming more and more important as a metric for modern systems, due to the increased instruction-level parallelism generated by superscalar or VLIW processors, and due to the density of integration that allows shorter latencies.

Unlike latency, bandwidth is an average-case quantity, and is related to memory traffic. Therefore, not all solutions that reduce latency necessarily translate into bandwidth improvements, as for energy. For instance, hardware prefetching can be helpful in removing latency bottlenecks, but will increase the traffic along the memory hierarchy, thus actually reducing the available bandwidth. Conversely, increasing the size of a given level of memory hierarchy will typically improve both (average) latency and bandwidth, because of reduced traffic in the lower levels of the hierarchy. Therefore, due to their extensive nature, several architectural techniques that improve bandwidth translate directly into energy savings.

The works by Burger et al. [1997] and Burger [1998] present several variants of traffic-efficient caches that reduce unnecessary memory traffic by clever choice of associativity, block size or replacement policy, as well as careful strategies for memory fetches. These solutions do not necessary improve worst-case latency, but result in reduced read and writes, thus reducing energy.

Another important class of techniques is based on the compression of the information transmitted between any two levels of the hierarchy. This solution has been applied to both instruction and data memories, by implementing ad hoc instruction compression schemes.

Instruction compression aims at reducing the large amount of redundancy in instruction streams, by storing compressed instructions in main memory and decompressing them on-the-fly before execution (or when they are stored in instruction cache). The various techniques trade off the aggressiveness of the compression algorithm with the speed and power consumption of the hardware decompressor.

The approach by Yoshida et al. [1997] replaces the individual instructions used by an embedded application with binary patterns of limited width (i.e., $\lceil \log_2 N \rceil$, where N is the number of distinct instructions appearing in the code). A hardware decompression table is required for the core to be able to use the same set of instructions.

Benini et al. [1999] build upon the method of Yoshida et al. [1997] by overcoming its major limitation: The number of distinct instructions may be relatively large, with the result of increasing the size of the instruction decompression table and complicating the implementation of the controller, especially when the bit-width of the compressed instructions is not a multiple of the byte.

In their approach, only a fixed subset (namely, 256) of the used instructions is compressed, and the code stored in memory consists of a mix of compressed and uncompressed instructions.

Another approach is followed by Ishihara and Yasuura [2000], in which frequently executed basic blocks are replaced by a single instruction. The merged sequence of object codes is restored by an instruction decompressor before decoding takes place. The decompressor is implemented as a ROM.

Code compression techniques have been used to reduce the size of executables of programs; new ideas on the subject have thus been explored, especially for what concerns the domain of embedded systems; in detail, in the work by Liao et al. [1998] and by Lekatsas and Wolf [2000], code *density* is the main cost function, and power reductions are obtained as a by-product. However, the work by Benini et al. [2001] targets power minimization as the primary objective.

The principle exploited in code compression (namely, reduction of memory traffic), can be extended to the case of data. The *compressed cache* by Yang et al. [2000] is based on the *frequent value* locality, that is, the fact very few values (typically small integers) account for usually around half of the total memory accesses, for most benchmarks. This allows the storage of the selected values in a compressed form. Compression/decompression is performed on the fly on accesses from/to the next hierarchy level.

4. FUTURE PERSPECTIVES

The Semiconductor Industry Association (SIA) and most analysts predict that embedded memories will increasingly dominate the SoC content both in terms of number of transistors and silicon real-estate. According to the SIA, ITRS 2000, 52% of the area of a typical 2002 SoC design is occupied by embedded memories. This percentage is expected to increase to 71% in 2005. Clearly, this trend emphasizes the strategic role of embedded (on-chip) memories for SoC architectures, and motivates the efforts made in academia and in the business arena to improve the integration of logic and memories onto the same silicon substrate.

The purpose of this section is to outline the main challenges and development directions for embedded memories, following both an *evolutionary* and a *revolutionary* path. Whereas evolutionary approaches are based on mainstream memory technology, revolutionary approaches leverage novel devices and physical properties of materials. The risk associated with revolutionary approaches is obviously very high, and success is a long-term perspective. On the other hand, the pay-off for a memory technology capable of overturning the current mainstream is staggering. For this reason, numerous revolutionary approaches are being pursued, and surveying all of them would be impossible. Thus, after outlining the most active areas of evolution in traditional memory technologies, we focus on revolutionary technologies that are already at an advanced stage of development and commercialization level.

4.1 Trends of Evolution

The main objectives in the evolution of embedded memories are high density, storage permanence, speed, and reliability. At the same time, a low cost,

high-yield process is desirable to successfully compete with multi-chip alternatives. No single memory technology excels in all categories: Density is maximum for DRAMs; SRAMs are better in speed; and FLASH and EEPROMs are nonvolatile. Improvements with respect to all the cost metrics listed above are pursued at the technology, circuit, and architectural levels.

Density improvements are especially critical for SRAMs, whose atomic six-transistor cell in Figure 1 is much larger than that of DRAMs and FLASH/EEPROMs. A promising approach in this direction is the development of the 1T SRAM [Glaskowsky 1999] (one-transistor SRAM). This architecture is based on an elementary cell containing one transistor and a MOS capacitor (similarly to early DRAMs), which is still much larger than modern DRAM cells, but it is fully compatible with logic CMOS technology. The cell does not contain any regenerating feedback, hence it must be refreshed as in DRAMs. The 1T SRAMs completely hide the refresh controller to the external world, and offer an input–output interface fully compliant with standard synchronous memory macros. The main issue in achieving this goal is to avoid delayed reads when the read target cells are being refreshed. The 1T SRAM architecture overcomes this problem by implementing a clever caching structure: the memory is internally multibanked, and whenever a bank needs to be refreshed, its entire content is copied into a standard 6T SRAM memory buffer, which is accessed in replacement of the 1T bank being refreshed. Additionally, the multibank architecture helps speed up access times because it reduces the bitline capacitance that must be driven by a cell during reads. Thus, access time is only marginally larger than that of standard embedded SRAMs. Needless to say, the logic CMOS compatible cell, the heavily multibanked architecture, the controller and the refresh buffer have a significant area cost, and the 1T SRAM is less dense than embedded DRAMs (15% density loss is reported), but it is fully process-compatible with logic CMOS and significantly denser than traditional 6T SRAMs (70% smaller area for the same capacity). Additionally, power consumption is significantly reduced. A factor of 4 power reduction with respect to 6T SRAMs is claimed.

Although density is probably the main concern for embedded SRAMs, dedicated-process memories have their own challenges. The most significant research and development efforts are directed toward decreasing the complexity and improving the yield of merged memory–logic processes. Excessive cost, caused by these joint factors, is the most significant obstacle to the diffusion of embedded DRAMs and embedded FLASH (EEPROM), in view of the competition from low-cost commodity memories [Rajsuman 2001; Watanabe et al. 1997]. Nonvolatile memories are also faced by limited write performance and reliability. Writes are much slower than reads and require high voltages to erase the content of floating-gate cells and store new content. These high voltages cause oxide wear-out of the floating-gate transistors under multiple rewrite cycles. Today's nonvolatile memories can withstand approximatively 10^6 write cycles [Strauss and Daud 2000], which is already one order of magnitude better than in 1999 [Cappelletti et al. 1999]. One million rewrites are unfortunately not sufficient for some applications characterized by a fast rewrite rate and long life cycle [Strauss and Daud 2000]. Fast and low-stress

techniques for nonvolatile memory erase and rewrite are being actively investigated [Cappelletti et al. 1999]. Another evolutionary trend in nonvolatile memories is the development of multibit storage cells. Floating-gate cells can be programmed with many different threshold values, enabling storage of multiple bits of information into a single cell with no area penalty [Borgatti et al. 2001; Cappelletti et al. 1999].

Another area of active research is the development of design flows for the integration of embedded memories within large SoCs. In this arena, we can distinguish two main directions, namely memory compilers (or generators) and memory design outsourcing (or memory IP development). The first approach [Clerc et al. 1999; Rajsuman 2001] is synthesis-oriented: the designer specifies a memory configuration (number of rows and columns, word size, output size, etc.) and the compiler instantiates the memory block by generating different back-end views for design integration such as HDL functional and timing simulation model, SPICE netlist, layout database (GDSII), etc. Memory compilers are very effective for small memories, but they become unreliable for very large and complex memories: predicting delay, power, and reliability is very hard for large memory cuts, because second-order effects (e.g., line inductance) become non-negligible. Furthermore, memory compilers still produce inferior results in comparison with hand-crafted designs, and the optimality loss tends to grow with memory size. For these reasons, large memories are usually hand-designed as hard macros. These macros can be either exchanged among design groups in large companies, or they can be outsourced to specialized design houses [Hall and Costakis 2001]. Generally, these companies do employ specialized generators for functional blocks inside the memory macro, but they customize the macro on demand, and they tailor their generators to different technologies and processes. In this way, SoC design time is still reduced, but the embedded memory is highly optimized. The choice between memory compilers and outsourcing is a difficult one and most SoC designs exploit both, depending on the size of the memory macro to be instantiated. In perspective, we expect memory compilers to conquer larger market share as embedded memories become commodified, but the largest memory macros will still be hand-crafted (and increasingly outsourced). From the technology viewpoint, SRAMs are definitely more amenable to automatic generation, whereas generators for dedicated-process memories, such as DRAMs and FLASH are still quite limited in flexibility and usage.

Besides memory instantiation, validation and design quality assessment methodologies are in rapid development [Rajsuman 2001]. In particular, timing and power analysis are critical topics in this area. As design sign-off moves toward higher levels of abstraction, there is a growing need for *sign-off quality* memory modeling techniques, primarily for performance, but also for power. Large memory macros are complex, mixed signal subsystems, and full-chip simulation would not have been thinkable without abstracting away their transistor-level details. This abstraction should not, however, compromise the accuracy of timing characterization, to avoid potentially fatal design failures. Memory characterization usually relies on manual analysis: the memory designer identifies the critical path, creates load models, and extracts an

equivalent circuit for SPICE simulation. Worst-case corners are specified and characterization of the process and operating condition induced variations is then carried out through iterative simulation. Unfortunately, this process is slow and error-prone, and automatic analysis and model extraction tools are strategic advances that we expect to see in this area.

The growth of on-chip memory usage has profound implications on chip yield [Dipert 2001b; Zorian 2000]. One of the primary causes of chip failure is the growth of memory as a percentage of a chip's area. Dedicated-process memories require more complex fabrication processes, but in general embedded memories imply higher cell density, thereby increasing the likelihood for a point defect to create a faulty cell. To circumvent this problem, semiconductor manufacturers started adding redundancy into large embedded memories, to enable repair after manufacturing, as is commonly done in stand-alone memories. The standard test-and-repair process for commodity memories follows three steps: (i) memory testing is performed on a dedicated tester with automatic test pattern generation capability (ATPG); (ii) test results are down-loaded and diagnosis is performed to locate defective cells; and (iii) chip repair is carried out by a fuse-blow process using laser repair equipment. For an embedded memory, these steps are much more cumbersome and expensive than for a stand-alone memory. SoC testers are much more expensive because they must test both memory and logic; furthermore, feeding test patterns to an embedded memory requires an on-chip test bus (with a silicon area cost). External test generation and testing can be bypassed with built-in self-test (BIST) techniques. BIST embeds the memory test function on-chip [Zorian 2000].

The BIST block is functionally equivalent to the on-chip test access bus and the ATPG required to test the memory. In addition, BIST allows an at-speed test, which is critical for high-speed chips. The use of basic BIST techniques does not simplify diagnosis and repair, but it can simplify the architecture of the postrepair tester, which does not have to perform ATPG. This idea can be pushed one step further, towards built-in redundancy analysis (BIRA). Instead of transferring the test results externally, on-chip diagnostic circuits analyze the BIST outcome and identify the failed memory rows/columns. In this case, only the redundancy configuration map needs to be transferred to the external repair equipment, which can be highly simplified. As a last step, even the laser repair equipment can be eliminated. The fuse-based "hard" repair then becomes "soft" repair, also called built-in self-repair (BISR). In addition to BIST and BIRA, BISR includes the storage of repair data and the control of a soft reconfiguration mechanism that bypasses the failed cells and activates available spares [Schober et al. 2001; Zorian 2000].

An example of a practical instantiation of the concepts outlined above is the STAR memory system by Virage Logic [Virage Logic]. It is a memory macro containing not only memory banks, but also a test, diagnosis, and repair controller, and spare blocks. The controller automatically tests the memory, allocates redundancy resources, and coordinates any needed repairs. Two repair methods are available: (i) *factory repair*, in which the STAR processor performs BIRA and transfers programming instruction to external laser repair equipment; and (ii) *field repair*, which tests and repairs memory instances each time the SoC

is powered up or reset. These modes of operation are not mutually exclusive. Factory repair is well suited for eliminating manufacturing defects and raising chip yield, whereas field repair is useful for any subsequent problem over the life of the end product. Clearly, the self-repair approach is viable only for fairly large memories: the STAR controller includes a few thousand gates and requires additional nonvolatile memory for storing test patterns and programs. Virage Logic commercializes STAR for memory blocks not smaller than 256 KB.

Self-diagnosis and self-repair are effective countermeasures for permanent failures, but unfortunately, memories are subject to a number of transient failure mechanisms that take place during system operation for a limited time span. Phenomena that cause transient faults are supply noise, radiation, electrostatic discharge from I/O pins, etc.

Effective techniques to test and repair these types of faults include periodic self-diagnosis and self-repair, as well as self-checking and fault-tolerant circuits and architectures. Self-checking and fault-tolerant operation exploit some amount of redundancy in storing information in memory. As a result, all these techniques impose a significant area penalty but a limited performance penalty. The research activity on error-resilient memory architectures is extremely active, and the interested reader is referred to specialized literature in the field (see, for example, Lala [2001]; Sarrazin and Malek [1984]). In perspective, the reduction in minimum feature size, the increasing complexity of silicon technology, and the trend of decreasing supply voltages, which together contribute to diminishing the signal-to-noise ratio in digital integrated circuits, are strong drivers towards the widespread adoption of these techniques in future SoCs.

4.2 Revolutionary Technologies

As seen in the previous section, mainstream embedded memories are rapidly evolving to meet the challenges of SoC design. However, a substantial research effort is also directed towards the development of radically new solid-state storage devices, based on new materials and different physical phenomena. In this area, we briefly mention four "revolutionary" technologies that have matured beyond the level of a single-cell feasibility study and show good promise. Ferroelectric (FRAM), silicon-oxide-nitride silicon (SONOS), and giant magnetoresistance RAM (GMRAM) are nonvolatile, read-write memories that aim at overcoming the fundamental limitations of floating-gate memory write process (low speed and limited reliability). Negative-resistance device (NDR) RAMs are being investigated as high-density alternatives to SRAMs.

Ferroelectric RAM [Dipert 2001a; Takasu 2001] is probably the most mature technology. These memories are based on ferroelectric polarization effects, which can be summarized as the capability of a material to store polarization in absence of an applied electric field. Under an electric field, a ferroelectric material goes through an hysteresis cycle that can be exploited to store one bit of information (i.e., 0 or 1 depending on which branch of the hysteresis cycle the cell is polarized into), similarly to a DRAM capacitor, but in a nonvolatile fashion. Reads are destructive, and FRAMs automatically rewrite cell content after reads, analogous to a DRAM read-refresh cycle. Similarly to floating-gate

memories, FRAMs exhibit fatigue under multiple rewrites, but they can withstand up to 10^{13} rewrite cycles. Further, rewrite times are in the nanosecond range, as opposed to the microsecond to millisecond times of floating-gate memories. FRAM fabrication technology is based on thin-film deposition over a silicon substrate. Current challenges in FRAM production are increasing process yield, cell density, and ensuring compatibility with a logic-CMOS process, an essential requirement for embedded applications [Dipert 2001a].

SONOS nonvolatile memories are close relatives of traditional floating-gate memories, but they replace the floating-gate with a thin silicon nitride film. The stored charge is localized in isolated sites within the silicon nitride dielectric, as opposed to a delocalized charge storage in the conductive polysilicon of a standard floating-gate. This is a key reliability advantage for SONOS. In a floating-gate structure, if the oxide becomes permeable to charge, then the entire conductive floating-gate is discharged. In SONOS, only a few localized charge traps can be affected by a localized oxide failure, and information storage is degraded, but not destroyed. Integration of SONOS with standard CMOS is simple: only a few (as little as two) additional masking steps are required. Cell size is very small, and multibit storage is possible. Even though SONOS memories seem very strong competitors for both EEPROM/FLASH and DRAMs, their technology is still in a maturing stage, and the basic failure mechanisms of these memories needs to be better understood [White et al. 2000].

Giant magnetoresistance [Dipert 2001a; Strauss and Daud 2000] and negative-resistance [Nemati and Plummer 1998] memories are less mature than FRAMs and SONOS memories. GMRAM is based on the resistance change of a multilayered thin-film material when subject to a magnetic field (induced by an electric current in the underlying substrate). The resistance variation is enduring, but it can be reversed. Hence, GMRAMs are inherently nonvolatile. They also show much higher rewrite reliability and speed than traditional floating-gate memories. Integration with standard CMOS processes, albeit possible, has still to be fully demonstrated at high levels of integration. Finally, NRD memories are under investigation as high-density SRAM replacement. They leverage a vertically stacked PNPN structure (a thyristor) in contact with an addressable MOS transistor. Vertically stacking the device, similar to DRAMs capacitors, is highly area-efficient, but at the same time, the high current drive capability of the PNPN thyristor has the potential for read times comparable, if not superior, to those of current SRAMs. Furthermore, in contrast with current DRAMs, the information stored in the cell does not need to be periodically refreshed. Even though simple NRD memories have been fabricated, their fabrication process is fairly complex and not yet mature for mass production in submicron processes.

Concluding this section, we observe that current mainstream embedded memory technologies are still maturing, and they can partly leverage the huge investments and design experiences made in the commodity memory field. Thus, we do not expect CMOS SRAM, DRAM, and FLASH/EEPROM to run out of steam anytime soon. However, our brief survey of "revolutionary" approaches shows that viable alternatives do exist, and they are under very active development.

5. CONCLUSION

Embedded systems are now becoming ubiquitous; in particular, they have large applicability in mobile, battery-operated personal communication systems, for which energy consumption is a major constraint.

Among the various contributors to the system's energy budget, memory plays a prominent role; in fact, reading and writing data to the memory hierarchy is an operation which takes place very often, especially in data-dominated applications (e.g., video and audio playing). As such, memory energy optimization has shown itself to be one of the most promising and successful approaches to system power minimization.

In this article, we have surveyed the current state-of-the-art in energy efficient embedded memory design. We have addressed technological, architectural, and methodological issues by providing theoretical illustrations and practical application examples of the most promising memory design approaches. Although broad, the survey does not cover the whole spectrum of available solutions proposed in the literature; our focus has been on hardware-centric techniques. Equally important energy optimization opportunities can be provided by software-centric and HW–SW codesign approaches. We have not considered them in this manuscript for space reasons, but the interested reader can refer to [Power Aware Design Methodologies 2002] and [Panda et al. 2001] for excellent reviews of these two aspects of energy-aware memory design.

REFERENCES

AGARWAL, A. AND PUDAR, S. D. 1993. Column-associative caches: A technique for reducing the miss rate of direct-mapped caches. In *ISCA-93: ACM/IEEE International Symposium on Computer Architecture* (San Diego, CA, May), 179–180.

AGAWA, K., HARA, H., TAKAYANAGI, T., AND KURODA, T. 2001. A bitline leakage compensation scheme for low-voltage SRAMs. *IEEE J. Solid-State Circuits 36*, 5, 726–734.

AMRUTUR, B. AND HOROWITZ, M. 1998. A replica technique for wordline and sense control in low-power SRAMs. *IEEE J. Solid-State Circuits 32*, 8, 1208–1219.

AMRUTUR, B. AND HOROWITZ, M. 2000. Speed and power scaling of SRAMs. *IEEE J. Solid State Circuits 34*, 2, 175–185.

BAHAR, R. I., ALBERA, G., AND MANNE, S. 1998. Power and performance tradeoffs using various caching strategies. In *ISLPED-98: ACM/IEEE International Symposium on Low Power Electronics and Design* (Monterey, CA, Aug.), 64–69.

BAJWA, R. S., HIRAKI, M., KOJIMA, H., GORNY, D. J., NITTA, K., SHRIDHAR, A., SEKI, K., AND SASAKI, K. 1998. Instruction buffering to reduce power in processors for signal processing. *IEEE Trans. VLSI Syst. 5*, 4 (Dec.), 417–424.

BENINI, L., MACII, A., MACII, E., AND PONCINO, M. 1999. Selective instruction compression for memory energy reduction in embedded systems. In *ISLPED-99: ACM/IEEE International Symposium on Low Power Electronics and Design* (San Diego, CA, Aug.), 206–211.

BENINI, L., MACII, A., AND PONCINO, M. 2002. *Memory Design Techniques for Low-Energy Embedded Systems*, Kluwer, Dordrecht.

BORGATTI, M., ET AL. 2001. A 64-Min single-chip voice recorder/player using embedded 4-b/cell FLASH memory. *IEEE J. Solid-State Circuits 36*, 3, 516–521.

BURGER, D. C. 1998. Hardware techniques to improve the performance of the processor/memory interface, Ph.D. dissertation, Univ. of Wisconsin-Madison.

BURGER, D. C., GOODMAN, J. R., AND KAGLE, A. 1997. Limited bandwidth to affect processor design. *IEEE Micro 17*, 6, 55–62.

CAPPELLETTI, P., GOLLA, C., OLIVO, P., AND ZANONI, E. 1999. *Flash Memories*, Kluwer, Dordrecht.

CHANDRAKASAN, A., BOWHILL, W., AND FOX, F. 2001. *Design of High-Performance Microprocessor Circuits*, IEEE Press, New York.

CLERC, S., DUFOURT, D., AND ZANGARA, L. 1999. High flexibility CMOS SRAM generator using multiplan architecture. In *IEEE ASIC/SOC Conference*, 414–417.

COUMERI, S. L. 1999. Modeling memory organizations for the synthesis of low power systems, Ph.D. dissertation, EE and CS Dept., Carnegie Mellon Univ.

COUMERI, S. L. AND THOMAS, D. E. 1998. Memory modeling for system synthesis. In *ISLPED-98: ACM/IEEE International Symposium on Low Power Electronics and Design* (Monterey, CA, Aug.), 179–184.

DIPERT, B. 2001a. Exotic memories. *EDN Mag.* (Apr.).

DIPERT, B. 2001b. Banish bad memories. *EDN Mag.* (Nov.).

FARRAHI, A., TELLEZ, G., AND SARRAFZADEH, M. 1995. Memory segmentation to exploit sleep mode operation. In *DAC-32: ACM/IEEE Design Automation Conference* (San Francisco, CA, June), 36–41.

FISK, B. R. AND BAHAR, R. I. 1999. The non-critical buffer: using load latency tolerance to improve data cache efficiency. In *ICCD-99: IEEE International Conference on Computer Design* (Austin, TX, Oct.), 538–545.

FRANK, D., DENNARD, R., NOVAK, E., SOLOMON, P., TAUR, Y., AND WONG, H. S. 2001. Device scaling limits of Si MOSFETs and their application dependencies. *Proc. IEEE 89*, 3, 259–288.

GLASKOWSKY, P. 1999. MoSys explains 1T-SRAM technology, *Microprocess. Rep. 13*, 12.

GONZÀLEZ, A., ALIAGAS, C., AND VALERO, M. 1995. A data-cache with multiple caching strategies tuned to different types of locality. In *ICS-95: ACM International Conference on Supercomputing* (Barcelona, Spain, July), 338–347.

GRUN, P., DUTT, N., AND NICOLAU, A. 2001. Access pattern based local memory customization for low-power embedded systems. In *DATE-01: IEEE Design Automation and Test in Europe* (Munich, Germany, March), 778–784.

HALL, E. AND COSTAKIS, G. 2001. Developing a design methodology for embedded memories. *ISD Mag.* (Jan.)

HORIGUCHI, M., ET AL. 1991. Dual-regulator dual-decoding-trimmer DRAM voltage limiter for burn-in tests. *IEEE J. Solid-State Circuits 26*, 11, 1544–1549.

ISHIHARA, T. AND YASUURA, H. 2000. A power reduction technique with object code merging for application specific embedded processors. In *DATE'00: Design Automation and Test in Europe* (Paris, France, Mar.), 617–623.

ITOH, K. 1990. Trends in megabit DRAM circuit design. *IEEE J. Solid-State Circuits 25*, 778–789.

ITOH, K., SASAKI, K., AND NAKAGOME, Y. 1995. Trends in low-power RAM circuit technologies. *Proc. IEEE 83*, 4, 524–543.

JOHN, L. K. AND SUBRAMANIAN, A. 1997. Design and performance evaluation of a cache assist to implement selective caching. In *ICCD-97: IEEE International Conference on Computer Design* (Austin, TX, Oct.), 510–518.

JOUPPI, N. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and pre-fetch buffer. In *ISCA-90: ACM/IEEE International Symposium on Computer Architecture* (Seattle, WA, May), 364–373.

JUAN, T., LANG, T., NAVARRO, J. J. 1997. Reducing TLB power requirements. In *ISLPED-97: ACM/IEEE International Symposium on Low Power Electronics and Design* (Monterey, CA, Aug.) 196–201.

KAMBLE, M. B. AND GHOSE, K. 1997. Analytical energy dissipation models for low-power caches. In *ISLPED-97: ACM/IEEE International Symposium on Low Power Electronics and Design* (Monterey, CA, Aug.), 143–148.

KEITEL-SCHULZ, D. AND WEHN, N. 2001. Embedded DRAM development: Technology, physical design and application issues. *IEEE Des. Test Comput. 18*, 3, 7–15.

KIMURA, K., ET AL. 1986. Power reduction technique in megabit DRAM's. *IEEE J. Solid-State Circuits 21*, 381–389.

KIN, J., GUPTA, M., AND MANGIONE-SMITH, W. 1997. The filter cache: An energy efficient memory structure. In *MICRO-30: Annual IEEE/ACM International Symposium on Microarchitecture* (Research Triangle Park, NC, Dec.) 184–193.

KO, U., BALSARA, P. T., AND NANDA, A. K. 1998. Energy optimization of multilevel cache architectures for RISC and CISC processors. *IEEE Trans. VLSI Syst. 6*, 2 (June), 299–308.

KUNIMATSU, A., ET AL. 2000. Vector unit architecture for emotion synthesis. *IEEE Micro 20*, 2, 40–47.

LALA, P. 2001. *Self-Checking and Fault-Tolerant Digital Design*, Morgan-Kaufmann, San Mateo, CA.

LEE, H. S. AND TYSON, G. S. 2000. Region-based baching: An energy-delay efficient memory architecture for embedded processors. In *IEEE International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (Nov.), 120–127.

LEKATSAS, H. AND WOLF, W. 2000. Code compression for low power embedded systems. In *DAC-37: ACM/IEEE Design Automation Conference* (Los Angeles, CA, June), 294–299.

LIAO, S. Y., DEVADAS, S., AND KEUTZER, K. 1998. Code density optimization for embedded DSP processors using data compression techniques. *IEEE Trans. CAD/ICAS 17*, 7 (July), 601–608.

LU, N. C. C. AND CHAO, H. 1984. Half-V_{DD} bit line sensing scheme in CMOS DRAM. *IEEE J. Solid-State Circuits 19*, 451–454.

MAI, K., MORI, T., AMRUTUR, B., HO, R., WILBURN, B., AND HOROWITZ, M. 1998. Low-power SRAM design usig half-swing pulse-mode techniques. *IEEE J. Solid-State Circuits 33*, 1659–1671.

MILUTINOVIC, V., MARKOVIC, B., TOMASEVIC, M., AND TREMBLAY, M. 1996. The split temporal/spatial cache: A complexity analysis. In *SCIzzL-6 Workshop* (Santa Clara, CA, Sept.), 89–96.

MINATO, O., ET AL. 1984. A 20ns 64K CMOS RAM. *ISSCC Dig. Tech. Papers* (Feb.), 222–223.

NACHTERGAELE, L., CATTHOOR, F., AND KULKARNI, C. 2001. Random-access data storage components in customized architectures. *IEEE Des. Test Comput. 18*, 3, 40–54.

NEMATI, F. AND PLUMMER, J. 1998. A novel, high density, low voltage SRAM cell with a vertical NDR device. In *IEEE Symposium on VLSI Technology*, 66–67.

PANDA, P. R., CATTHOR, F., DUTT, N. D., DANCKAERT, K., BROCKMEYER, E., KULKARNI, C., VANDERCAPPELE, A., AND KJELDSBERG, P. G. 2001. Data and memory optimization techniques for embedded systems. *ACM Trans. Des. Autom. Electron. Syst. 6*, 2 (April), 149–206.

PANDA, P. AND DUTT, N. 1999. *Memory Issues in Embedded Systems-on-Chip Optimization and Exploration*, Kluwer, Dordrecht.

PANDA, P., DUTT, N., AND NICOLAU, A. 2001. On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems. *ACM Trans. Des. Autom. Electron. Syst. 5*, 3, (July), 682–704.

PEDRAM, M. AND RABAEY, J. *Power Aware Design Methodologies*, Kluwer, Dordrecht.

POWELL, M., YANG, S. H., FALSAFI, B., ROU, K., AND VIJAYKUMAR, N. 2001. Reducing leakage in a high-performance deep-submicron instruction cache. *IEEE Trans. VLSI Syst. 9*, 1 (Feb.), 77–89.

PRINCE, B. 1997. *Semiconductor Memories*, 2d ed., Wiley, New York.

RAJSUMAN, R. 2001. Design and test of large embedded memories: An overview. *IEEE Des. Test Comput. 18*, 3, 16–27.

SARRAZIN, D. AND MALEK, M. 1984. Fault-tolerant semiconductor memories. *IEEE Comput. 17*, 8, 49–56.

SASAKI, K., ET AL. 1989. A 9-ns 1-Mbit CMOS RAM. *IEEE J. Solid-State Circuits 24*, 1219–1225.

SCHOBER, V., PAUL, S., AND PICOT, O. 2001. Memory built-in self-repair using redundant words. In *International Test Conference*, 995–1001.

SEZNEC, A. 1993. A case for two-way skewed-associative caches. In *ISCA-93: ACM/IEEE International Symposium on Computer Architecture* (San Diego, CA, May), 169–178.

SHIUE, W. AND CHAKRABARTI, C. 1999. Memory exploration for low power, embedded systems. In *DAC-36: ACM/IEEE Design Automation Conference* (New Orleans, LA, June), 140–145.

STRAUSS, K. AND DAUD, T. 2000. Overview of radiation tolerant unlimited write cycle non-volatile memory. In *IEEE Aereospace Conference* (Mar), 399-408.

SU, C. L. AND DESPAIN, A. M. 1995. Cache design trade-offs for power and performance optimization: A case study. In *ISLPD-95: ACM/IEEE International Symposium on Low Power Design* (Dana Point, CA, Apr.), 63–68.

SUZUOKI, M., ET AL. 1999. A microprocessor with a 128-bit CPU, ten floating-point MACs, four floating-point dividers and an MPEG-2 decoder. *IEEE J. Solid-State Circuits 34*, 11, 1608–1618.

TAKAHASHI, M., ET AL. 2000. A 60-MHz 240-mW MPEG-4 videophone LSI with 16-Mb embedded DRAM. *IEEE J. Solid-State Circuits 35*, 11, 1713–1721.

TAKASU, H. 2001. Ferroelectric memories and their applications. *Microelectron. Eng. 59*, 237–246.

TANAKA, H., ET AL. 1992. Stabilization of voltage limiter circuit for high-density DRAM's using pole-zero compensation. *IEICE Trans. Electron. E75-C*, 1 (Nov.), 1333–1343.

TANG, W., GUPTA, R., AND NICOLAU, A. 2002. Power savings in embedded processors through decode file cache. In *DATE'02: Design and Test in Europe* (Paris, France, Mar.), 443–448.

USAMI, K. AND KAWABE, N. 2000. Low-power technique for on-chip memory using biased partitioning and access concentration. In *IEEE Custom Integrated Circuits Conference* (May), 214–220.

Virage Logic *The STAR Memory System*, www.viragelogic.com.

WALSH, S. J. AND BOARD, J. A. 1995. Pollution control caching. In *ICCD-95: IEEE International Conference on Computer Design* (Austin, TX, Oct.), 300–306.

WATANABE, T., FUJITA, R., AND YANAGISAWA, K. 1997. Low-power and high-speed advantages of DRAM-logic integration for multimedia systems. *IECE Trans. Electron. E80-C*, 12, 1523–1531.

WHITE, M., ADAMS, D., AND BU, J. 2000. On the go with SONOS. *IEEE Circuits Devices 16*, 4, 22–31.

YANG, J., ZHANG, Y., AND GUPTA, R. 2000. Frequent value compression in data caches. In *MICRO-33: IEEE/ACM 33d International Symposium on Microarchitecture* (Monterey, CA, Dec.), 258–265.

YOSHIDA, Y., SONG, B., OKUHATA, H., ONOYE, T., AND SHIRAKAWA, I. 1997. An object code compression approach to embedded processors. In *ISLPED-97: ACM/IEEE International Symposium on Low Power Electronics and Design.* (Monterey, CA, Aug.), 265–268.

ZORIAN, Y. 2000. Yield improvement and repair trade-off for large embedded memories. *IEEE Des. Test Eur.* 69–70.

Received January 2002; accepted July 2002