Software-Controlled Processor Speed Setting for Low-Power Streaming Multimedia

Andrea Acquaviva, Luca Benini, *Member, IEEE*, and Bruno Riccó, *Senior Member, IEEE*

***Abstract*—In this paper, we describe a software-controlled approach for adaptively minimizing energy in embedded systems for real-time multimedia processing. Energy is optimized by clock speed setting: the software controller dynamically adjusts processor clock speed to the frame rate (FR) requirements of the incoming multimedia stream. The speed-setting policy is based on a system model that correlates clock speed with best case, average case, and worst case sustainable FRs, accounting for data dependency in multimedia streams. The technique has been implemented in a energy-efficient MPEG3 real-time decoder algorithm designed for wearable devices as a case study. The target system is the Hewlett-Packard SmartBadgeIII prototype system based on the StrongARM1100 processor. Hardware measurements show that computational energy can be drastically reduced (up to 40%) with respect to fixed-frequency operation.**

***Index Terms*—Energy optimization, low-power, multimedia systems, real-time systems.**

I. Introduction

ONE OF the most critical constraints on portable embedded systems is power consumption, which impacts battery size, weight, and lifetime, as well as system cost and reliability. In the design of embedded systems, a microprocessor-based architecture is often a forced choice because of its flexibility and fast time-to-market. In these architectures, the central processing unit (CPU) must handle a large fraction of the computational load imposed by applications and it is a major contributor to the power budget. In general, CPU energy consumption depends on the type of workload imposed by applications. We focus on ultraportable embedded devices targeted to streaming multimedia applications, such as audio and video decoding.

To improve energy efficiency, an application can dynamically reconfigure the system to provide the required services and performance levels without wasting power. Modern hardware components provide a large freedom in dynamically adjusting important power-correlated parameters such as clock frequency and supply voltage allowing quick adaptation also at runtime. Hence, it is possible to reduce power consumption by reducing system speed and supply voltage to the minimum level necessary to match real-time constraints.

It has often been observed [5] that frequency downscaling reduces power, but does not reduce energy, unless it is coupled with supply voltage scaling. Unfortunately, many components can be operated at different frequencies, but their supply voltage cannot be changed. Our paper demonstrates that energy can be saved by frequency scaling, even at a constant supply voltage.

To achieve this target, we first obtain a set of curves that relate performance with processor frequency for a given streaming multimedia application (namely, MP3 decoding). These curves are generally nonlinear because of memory latency and input–output (I/O) synchronization requirements. We formulate a policy that controls processor speed to save power while satisfying real-time constraints. The policy is application-driven in the sense that the application provides the information needed to set processor speed based on characteristics of the incoming stream and on performance-frequency curves. As opposed to traditional speed-setting policies [11], which set the clock frequency based on predicted processor utilization, application-driven speed setting is very robust and it does not suffer from stability and poor real-time performance problems.

Application-driven speed setting has been implemented and applied on MP3 audio decoding running on a StrongARM-based wearable computer. Power consumption measurements show that our technique saves significant power (up to 40%) with respect to fixed-frequency operation, while at the same time satisfying real-time playback constraints.

The remainder of this paper is organized as follows. In Section II, related work in the research area of system-level power optimization is surveyed. A description of multimedia system architecture is introduced in Section III. This is the context in which we adopt a variable frequency approach to optimize the energy consumption, as detailed in Section IV. Section V describes the algorithm that performs frequency setting, while in Section VI, we analyze how the algorithm has been implemented in a real multimedia application like the MPEG3 decoder. Experimental results are reported in Section VII.

II. Related Work

Several researchers have faced the problem of system-level power optimization [4]. The opportunities for system-level power optimization have been enhanced in the last few years by the availability of power-manageable processors [7], [8], offering low-power idle states and runtime clock/voltage settings.

The design of cores with speed- and voltage-setting capabilities has been addressed, among others, by Ishihara and Yasuura in [17], which presents a model of dynamically variable voltage

Manuscript received April 10, 2001; revised June 28, 2001. This work was supported by the Hewlett-Packard Laboratories. This paper was recommended by Guest Editor P. Marwedel.

The authors are with the Department of Electronics and Information Science, University of Bologna, 40136 Bologna, Italy.

Publisher Item Identifier S 0278-0070(01)09996-1.

processor. In the same work, they show how this capability can be exploited in order to minimize energy consumption by a static voltage scheduling algorithm working under timing constraints. This work is representative of a class of variable-voltage techniques for system-level power optimization. The basic idea in these approaches is to schedule the processor voltage in order to spend the minimum amount of computational power required to perform the given tasks. Additional hardware is needed to support this capability (like DC–DC converters). In addition, reducing supply voltage increases circuit delay so that the clock speed must be accordingly reduced [5], thus, voltage and speed must be regulated together leading, in the ideal case, to a cubic reduction of power and quadratic reduction of energy consumption.

Researchers addressed the voltage scheduling problem mainly from two different perspectives: application-driven [6], [9], [23] and operating-system level [10], [12], [14], [16], [17], [19], [21], [25].

Chandrakasan *et al.* [6] explores dynamic voltage setting in digital signal processors (DSPs) with variable workloads. Different from custom DSPs, general-purpose system-on-chips (SOCs) are not targeted to a particular application; an adaptation is necessary even with a fixed workload in order to reconfigure the hardware resources in a power-efficient way. Chandrakasan *et al.* demonstrates the effectiveness of decreasing together speed and voltage with respect to simply shut down the system in idle periods.

Recently, Sinha *et al.* [23] have investigated the idea of applications that are aware of their power requirements and help the operating system in taking decisions about resources to be allocated, clock frequency, and supply voltage. The energy model by which the actual values of voltage and frequency are derived, however, assumes a linear relationship between clock frequency of the processor core and execution time of a certain task. This model is not suitable in the context of real-time processing and, in general, does not take into account real-life systems bottlenecks like memory latency. In [9], the authors take a dual approach: here, the algorithm adapts its requirements to resource availability. The experimental results show how characteristics of data to be processed affect power consumption and, hence, how an adaptive approach can be effective in reducing dissipation. The same work also shows that a large amount of power is spent by the processor when in idle state because of limited network bandwidth or real-time synchronization requirements.

As for operating-system-level power optimization, several techniques have been developed to dynamically control the voltage of a system. In this context, variable voltage poses the problem of scheduling and, from this point of view, we can distinguish between *best effort* scheduling and *real-time* scheduling. In the former, the CPU voltage is lowered if a decrease of the future amount of computation is predicted. The target is the average reduction of power. Therefore, this method does not guarantee that all tasks meet their deadlines [10], [25]. In the latter case, the knowledge of the deadline of each task is exploited to set voltage and speed so that they just meet their time constraints (just in time computation) [12], [14], [16], [19], [21]. Some of these algorithms assume static scheduling [10], [16], [17], i.e., the workload for each task is characterized at design time, while others are dynamic [12], [14], [19], [21].

Restrictions on the effectiveness of variable voltage arise because the regulation range is discrete. Moreover, regulation freedom must deals with technology trends toward lower voltages. In addition, as already mentioned, variable voltage requires special hardware and the physical realization of the required circuitry is quite complex [17]. Most research work on variable-voltage processing is based on simulations; therefore, it does not fully take into account hardware limitations arising when the the proposed energy management policies are implemented on a real-life core. Recent experimental work on variable-voltage techniques for real-time multimedia processing has demonstrated that simulation results often lead to optimistic conclusions and that many practical issues still need to be resolved [11].

An alternative approach to variable-voltage processing is based on shutdown. In its simplest flavor, this method consists of a binary decision: whether or not turn off the power supply of the processor. This can be exploited to save power when it is not performing useful work.

Power shutdown of a component is a radical solution that eliminates all sources of power dissipation (including leakage) [4]. Another advantage is that it is easier to implement with respect to variable voltage because shutdown capabilities are available in almost any modern low-power chip; therefore, no dedicated hardware is required. The drawback with respect to variable voltage is that shutdown allows only two quantization level of voltage (*on* and *off*); from this point of view, variable voltage allows better adaptation to different workloads; hence, it is more effective [6]. Another drawback is that transition between inactive and functional state requires time and power; therefore, it is not effective in most cases to turn off processor as soon as it becomes idle. For this reason, predictive shutdown techniques have been proposed based on the assumption that some knowledge of future input events is available [4]. Paleologo *et al.* [21] utilize a finite-state stochastic model for a power-managed system. Shutdown decisions are statistically made on the basis of the past activity of the system. Other predictive techniques are presented in [15] and [24].

The approach described in this paper differentiates from both variable-voltage and shutdown-based techniques and it is based on speed setting alone with constant voltage supply. Its main advantage is to allow fine tuning of energy consumption without the need of a complex external circuitry as in variable voltage. In the past, it has been conjectured that a similar approach could not be effective and slowdown would leave energy consumption unchanged at best. This an artifact of the simplistic assumption on hardware behavior. Traditionally, it has been assumed that power consumption scales down with s^3, where s is the voltage and speed scaling factor. In addition, because the execution time is inversely proportional to s, energy depends on the square of the supply voltage and not on f, so is not useful to change only processor speed in order to save energy unless supply voltage is scaled down as well.

On the contrary, recent measurements [3], [17], [18] on real systems have demonstrated that running at less than the maximum frequency can be advantageous. In Section III, we provide

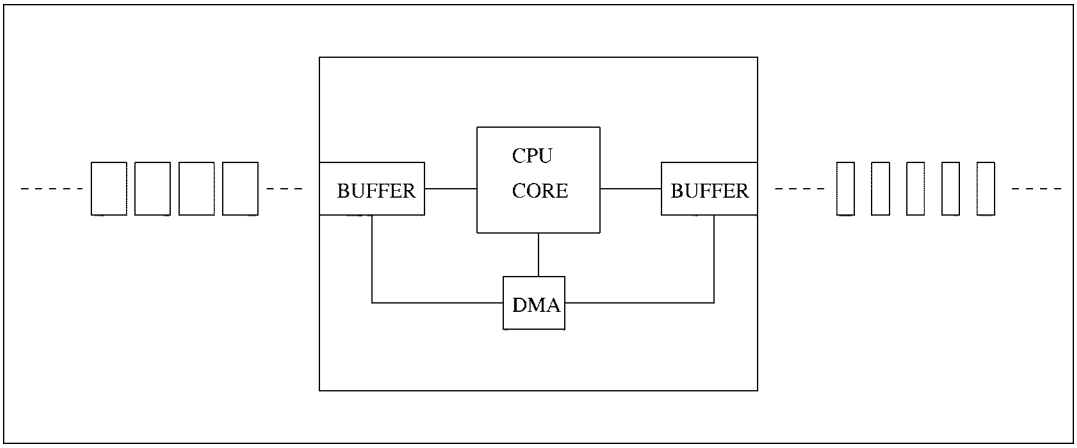

Fig. 1. Streaming multimedia architecture.

a theoretical explanation for this fact, which motivates the adaptive algorithmic power optimization strategy presented later in the paper.

III. Multimedia-Systems Architecture

In a processor-based architecture, both hardware and software organization impact power optimization. We briefly describe a multimedia system architecture used as a practical driver in our study and summarized in Fig. 1.

A. Hardware Architecture

The multimedia system consists of a wearable appliance (called *smart badge*) and a host computer or a network communicating with the device. When the wearable appliance performs streaming processing, the external system is responsible to provide input data to the device. The communication medium can be a wireless or a wired link. Since we deal with multimedia-stream processing, incoming data are in a compressed form in order to accommodate the bandwidth requirements. Because output devices like video and audio playback systems require a raw data input format, a decoding framework is needed. In most wearable devices, the unit responsible for this task is a general-purpose microprocessor. The processor core is often integrated in a an SOC together with several peripherals.

Outside the SOC, other onboard components are needed, such as external memory and audio or video codecs, which communicates with the CPU through I/O channels. Due to the different bandwidth requirement of communication protocols and between the input and output data, some level of buffering is necessary. To improve efficiency and parallelism, current SOCs contain I/O hardware units that can buffer data and manage standard communication channels (e.g., serial port, parallel port) in parallel to the CPU. High-bandwidth I/O devices often use a direct memory access (DMA) for enhanced performance.

Since this kind of architecture is targeted to handle a large range of workload levels, resource-power management is needed in order to reduce the waste of power when the workload is less than the maximum that the system is designed to sustain. Therefore, modern SOC architectures implement several power management capabilities. For example, we can adjust the frequency in a range of discrete values or we can stop the clock for some components at runtime. In most architectures, frequency can be programmed via software and a sequence of instructions is available in order to freeze the clock of a set of hardware components of the SOC. In addition, even by software, some other systems allow the gating of the power supply for the CPU core or for the integrated controllers.

B. Software Architecture

The software side of a multimedia system is composed by a server and a client part. The server functions as remote file system for the hand-held device because the hand-held has modest storage capability. When requested, the host starts sending data to the remote system, which accumulates them into an input buffer. Since multimedia-stream processing algorithms must handle variable input and output data rates, they make use of software buffering when hardware is not sufficient. Often, input and output channels are driven by a DMA controller, which transfers incoming data into the main memory buffer. The rate at which this buffer is filled depends on the bandwidth of the input channel and it is chosen in order to support the bit rate, which represents the speed of the stream's information transfer. Once the input buffer is full, the CPU starts to process the input data and sends the results to the memory output buffer. When this buffer is filled, data are ready to be transferred by the DMA toward the output channel. The destination of this data depends on the particular nature of the multimedia application. Input and output data rate are both key characteristics of the multimedia stream because they impose the speed at which the CPU must process the data to ensure that the output buffer is never empty in order to meet the real-time constraints.

IV. Variable Frequency and Energy Optimization

In this section, we discuss how energy reduction can be obtained by setting clock frequency even in the absence of an associated voltage regulation. This is in contrast to the common assumption that speed setting is effective only when accompanied by an adequate voltage-setting policy. Of course, if voltage is scaled with frequency, more power can be saved, but the point

here is that this is not a forced choice. To demonstrate our claim, we start by looking at the usual expression of dynamic power consumption

$$P = V_{\mathrm{DD}}^2 \cdot C_{\mathrm{eff}} \cdot f \tag{1}$$

where

V_{DD} supply voltage;
C_{eff} average switched capacitance;
f CPU clock frequency.

Because multimedia streams have a frame-based structure, it is useful to consider the frame processing time T_{frame}. The energy consumption in time T_{frame} can be immediately obtained as

$$E_{\mathrm{frame}} = V_{\mathrm{DD}}^2 \cdot C_{\mathrm{eff}} \cdot f \cdot T_{\mathrm{frame}} \tag{2}$$

where T_{frame} is the frame processing time. $T_{\mathrm{frame}} = N_{\mathrm{frame}} \cdot t$, where N_{frame} and t are the number of clock cycles necessary to elaborate a frame and the cycle time, respectively. We have then

$$E_{\mathrm{frame}} = C_{\mathrm{eff}} \cdot V_{\mathrm{DD}}^2 \cdot N_{\mathrm{frame}} \tag{3}$$

because $f = (1/t) \cdot E_{\mathrm{frame}}$ depends on N_{frame}, i.e., on the workload. Now, because the CPU must interface with external hardware, which is slower in general (e.g., off-chip memories), there are times in which the CPU is idle, so let us now express N_{frame} as

$$N_{\mathrm{frame}} = N_{\mathrm{useful}} + N_{\mathrm{idle}}. \tag{4}$$

We observe now that for a given algorithm N_{useful} (the number of cycles spent in execution of useful operations) is independent of the frequency because it is a function of the data to be processed, while N_{idle} (the number of cycles wasted with the CPU being idle) can be seen as a function $N_{\mathrm{idle}}(f)$, where f is one of the available processor frequencies. N_{idle} is a nondecreasing function of f. This is because when a certain CPU activity has a fixed time duration, e.g., a memory access or some kind of I/O device access, increasing only the frequency leads to an increasing of the idle cycles because useful operations are performed in a shorter time. This happens every time the CPU is not the speed-limiting element.

By taking into account this dependency, the energy expression can be rewritten as

$$E_{\mathrm{frame}} = V_{\mathrm{DD}}^2 \cdot C_{\mathrm{eff}} \cdot (N_{\mathrm{useful}} + N_{\mathrm{idle}}(f)). \tag{5}$$

Now, it must be considered that real-time algorithms need to provide a minimum amount of data output in a given time $T_{\max}$ depending on the output bandwidth required. For example, in audio MPEG decoding, this bandwidth depends on the sample rate of the decoded sound. The output bandwidth is directly proportional to the frame elaboration speed, which can be expressed as $1/T_{\mathrm{frame}}$. To meet the real-time constraints, this speed must be greater than $1/T_{\max}$.

The target of power optimization is to reduce E_{frame} acting on N_{idle} under the constraint

$$\frac{f}{N_{\mathrm{useful}} + N_{\mathrm{idle}}} \geq \frac{1}{T_{\max}}. \tag{6}$$

Speed setting pursues the target of power minimization by decreasing f so that in (6), N_{idle} decreases while N_{useful} and $T_{\max}$ are fixed because they affect the output data bandwidth. The lower bound of f, namely, f_{opt}, is fixed by the requirement that all the useful work be executed in $T_{\max}$ (just-in-time computation).

Speed-setting effectiveness depends on the workload characteristics and the system's architecture (both hardware and software). It reduces the costs of memory latency in terms of CPU wait states. Hence, in execution dominated by memory access (high miss rate) and where memory latency is higher, this technique is more effective [3]. In addition, from a system-energy perspective, since the CPU clock often feeds other on-chip components, additional system power can be saved by reducing useless work on these as well (even if in some cases they implement power down and gated clock strategies).

We classify these two idle contributions as *implicit idleness* and *explicit idleness*. The first identifies CPU idleness dispersed among useful operations (mainly during memory wait cycles on cache misses). This term varies with f: since memory access time is fixed, adjusting the frequency involves variations in number of wait states in a bus cycle. This happens when (as usual) the CPU is not the speed-limiting element. The second is due to coarsely clustered idle cycles. Explicit idleness is quite common in practice. When the execution time is fixed, as in the case of real-time constrained algorithms, making a computation faster involves the need of storing the results of computation in a buffer waiting for some event external at the CPU. During that time, the CPU experiences idleness, which can be eliminated without affecting the algorithm effectiveness by increasing the time spent in useful operations, i.e., by lowering the CPU frequency. Explicit idleness can be reduced also by putting the processor in a low-power state while waiting and restoring the running state when the external event arrives (i.e., an external interrupt), but in this case, we need to account for the time and energy overhead needed to shut down and wake up the CPU.

On the other hand, since the frequency cannot be adjusted continuously, it is hard to completely eliminate CPU idleness. As an additional concern when evaluating the effectiveness of a speed-setting policy, the delay and the energy spent to set the processor speed must be considered. However, in the context of the proposed algorithm, this penalty is not noticeable because the appropriate frequency value is chosen at the beginning of the stream in an application-drive fashion.

V. Frequency-Setting Framework

As previously mentioned, the effectiveness of a speed-setting policy depends on the hardware characteristics and on the workload. Therefore a characterization of the system performance as a function of clock frequency is needed in order to choose the speed that guarantees the level of performance required.

Multimedia processing is carried out on a frame-by-frame basis, every iteration yielding a variable amount of output data. Since real-time applications must produce a fixed amount of output data in a given period of time, the frame-processing rate $\mathrm{FR}(f)$ is an appropriate metric to indicate the level of performance supported by the system at a given clock frequency f. The main challenge in modeling the frame rate (FR) as a function of clock frequency is that it depends on the characteristics of

TABLE I
FRAME RATE LOOKUP TABLE

	16KHz	24KHz
16KBit/s	65.36	69.67
32KBit/s	63.25	67.95
64KBit/s	60.61	66.56

Values represent $\mathrm{FR}_A^o(f_{\max})$ in frame/sec at different sample rates (row) and bit rates (column).

the multimedia stream as well. In general, we have $\mathrm{FR}(f, s, d)$, where s is a parameter representing characteristics of the entire stream, namely, the sample rate sr and the bit rate br, and d represents characteristics of a single frame in the stream (e.g., frame size). For MP3 audio, the achievable frame-procesing rate at a given clock frequency is a strong function of the stream's bit rate and sample rate.

For a fixed $\mathrm{br} = \mathrm{br}^*$ and $\mathrm{sr} = \mathrm{sr}^*$, frame-by-frame variations are quite small, but nonnegligible. To build our performance model, we analyze several streams at br^* bit rate and sr^* sample rate and we monitor the worst case frame processing time as well as the best case frame processing time at various frequencies. Then, we define three curves $\mathrm{FR}_B(f), \mathrm{FR}_A(f), \mathrm{FR}_W(f)$, representing best case FR (i.e., the FR that could be achieved if all frames in the stream could be processed at max speed), the average case FR, and the worst case FR, respectively. By construction, $\mathrm{FR}_B(f) \geq \mathrm{FR}_A(f) \geq \mathrm{FR}_W(f)$. The three curves are normalized with respect to $\mathrm{FR}_A(f_{\mathrm{MAX}})$, the average FR achieved when the processor is run at maximum speed. All three curves are monotonically increasing in frequency. The normalized $\mathrm{FR}_A(f)$ has maximum value 1.

The same process is repeated for several different values of br and sr, including all corner cases (i.e., maximum and minimum br and sr in a range of allowed values). The normalized curves are plotted on the same $\mathrm{FR} \times f$ plane. We then obtain three normalized curves: the *overall best* $\mathrm{FR}_B^o(f)$, the *overall average* $\mathrm{FR}_A^o(f)$, and the *overall worst* $\mathrm{FR}_W^o(f)$. The first one is obtained by selecting the largest FR value among all FR_B curves for each frequency point. The second one is obtained by averaging all values of FR_A curves at every frequency. The third one is obtained by selecting the smallest FR value among all FR_W curves for each frequency point. The curves $\mathrm{FR}_A^o(f), \mathrm{FR}_B^o(f)$, and $\mathrm{FR}_W^o(f)$ are the performance model for the system.

The frequency-setting algorithm exploits the knowledge of $\mathrm{FR}_A^o(f)$, $\mathrm{FR}_B^o(f)$, and $\mathrm{FR}_W^o(f)$ curves, as well as the knowledge of the $\mathrm{FR}_A(f_{\mathrm{MAX}})$ for each allowed combination of br and sr. It can be summarized as follows: when stream decoding begins, the algorithms extracts the br and sr information from the stream header and looks up the corresponding value of $\mathrm{FR}_A(f_{\mathrm{MAX}})$ shown in Table I. Furthermore, given the sr value, it is possible to determine the average FR $\mathrm{FR}_{\mathrm{req}}$ that the system must support to guarantee real-time playback of decompressed audio by the following equation:

$$\mathrm{FR}_{\mathrm{req}} = \frac{\mathrm{sr}}{N_{\mathrm{samples}}} \tag{7}$$

where N_{samples} is the number of samples per frame.[1]

Given a $\mathrm{FR}_{\mathrm{req}}$ requirement, the frequencies $f_{\min}$, f_{av}, and $f_{\max}$ are computed by intersecting the curves $\mathrm{FR}_B^o(f)$, $\mathrm{FR}_A^o(f)$, $\mathrm{FR}_W^o(f)$ with the horizontal line $\mathrm{FR}_{\mathrm{req}}^N = \mathrm{FR}_{\mathrm{req}}/\mathrm{FR}_A(f_{\mathrm{MAX}})$ representing the normalized FR required and finding the abscissas of the intersections, as shown in Fig. 3. Once the normalized FR required is obtained, the algorithm finds the optimum frequency value by looking in a lookup table, where the frequency versus FR points are stored (these points correspond to the FR curves previously described). More precisely, instead of one, we find three values, namely, $f_{\min}$, f_{av}, and $f_{\max}$, that define a range of allowed frequencies for speed setting.

Running the processor at f_{av} should be enough to provide real-time playback, but some buffering is required to accommodate frame-decoding rate jitter. Alternatively, it is possible to run the processor at $f_{\max}$. At this frequency, real-time performance is guaranteed on a frame-by-frame basis with minimal jitter compensation buffering. However, the processor consumes more energy than what is needed most of the time. Finally, $f_{\min}$ frequency can be used for short periods of time if we find out that frames are consistently processed faster than the average rate. Clearly, it is also possible to run the processor faster than $f_{\max}$ (if $f_{\max}$ is smaller than the maximum processor frequency). This is clearly suboptimal from the energy viewpoint, but it can be a forced choice in systems where processor time is not fully dedicated to MP3 decoding. In this case, the performance model described above makes it possible to quantify the fraction of processor time that is made available for other tasks by clocking the processor at $f > f_{\max}$.

It is important to stress that the $\mathrm{FR}(f)$ curves are not linear in general. This is because the memory system and interfaces do not speed up like the processor with increasing clock frequency. Increasing f leads to a decrease of the ratio $N_{\mathrm{useful}}/N_{\mathrm{idle}}$; therefore, the FR does not increase linearly with f. The slower the speed of the external hardware (e.g., memory access time) with respect to the processor, the flatter the performance curve and the greater can be the effectiveness of the speed-setting policy. Other than the hardware characteristics, the shape of the curve depends on the ratio between the computation time spent inside the CPU and that spent outside the CPU. Considering the nonideality of external memory, this can be expressed also as the ratio between the external accesses and the total memory accesses, which in turns is equal to the cache miss rate. It must be observed that the minimum FR also takes into account the decrease in bandwidth caused by the synchronization of the processor with the I/O controllers and the external peripherals as well.

VI. ALGORITHM IMPLEMENTATION

The frequency-setting algorithm and the system characterization explained in the previous section has been exploited in the implementation of an energy efficient streaming multimedia algorithm, namely, an adaptive MPEG3 audio decoder. The target system is the HP SmartBadgeIII, a prototype of wearable device based on the StrongARM1100 core [2]. The CPU is integrated in an SOC that contains several peripheral units and controllers. In particular, the DMA controller allows the management of I/O channels relieving the CPU from a great amount of synchronization work. In our case, the target system communicates with a host PC through a serial link that provides the encoded

[1]This number is fixed to 576 for MPEG1 and to 576 for MPEG1 phase2.

audio samples, which, in the case of the MPEG3 standard, are grouped in frames. The algorithm starts decoding by looking at the input buffer waiting for data to be available. When this happens, it takes a block of frames and decodes it. Then it sends the results toward an external logic when finished. In our case, output data are decoded audio samples sent by the integrated serial controller to an external audio chip, which performs digital to analog conversion and feeds the audio output connector of the SmartBadgeIII.

The adaptation framework is based on the availability of the stream characteristics by reading the header of the frames. In particular, it looks at the beginning of the stream for the sample rate and bit rate values. Based on this knowledge, the appropriate frequency is chosen as described in the previous section.

In StrongARM1100, 12 frequency levels are available by programming a phase-locked loop. This can be done by writing a control word in a memory-mapped register. Other operations must be done before and after writing into this register. For example, it is necessary to change the memory wait states in order to accommodate the new frequency with the fixed memory access time.

VII. Experimental Results

The results presented in this paper were obtained for the system architecture of the HP SmartBadgeIII prototype hand-held device, based on the StrongARM SA-1100 embedded core. The embedded application is MPEG-layerIII audio decoding. Performance and power were obtained using the simulation tools described in [27], which has been validated against hardware measurements.

Experimental results in this section can be split in two parts. In the former, we illustrate the frequency-setting framework and the energy reduction, which can be obtained on a frame by frame basis. In the latter, we demonstrate the effectiveness of our approach by describing the energy consumption results of the adaptive MPEG3 decoder built using the proposed algorithm. In that case, energy consumption is related to the decoding of an entire audio stream. In this case, as we will describe later, the energy reduction in some cases is much greater because the speed-setting policy may partially reduce the energy costs of the I/O synchronization.

A. Frequency-Setting Results

The first plot, in Fig. 2, shows how energy per frame changes with clock frequency for two MP3 streams with different br (same sr). The plot is obtained by running MP3 decode at the maximum FR achieved at a given clock frequency. This can be slower or faster than the one required for real-time playback. The purpose of this plot is to show that energy per frame monotonically increases with frequency (contradicting the simplistic model where energy is constant with variable frequency) because the processor wastes energy waiting for slow memories during cache misses.

The actual energy penalty for excessive clock speed is even larger than what is shown in Fig. 2 because active decoding must be stopped when the output buffer is full to avoid frame loss. Even if we stop decoding by forcing the processor in idle state,

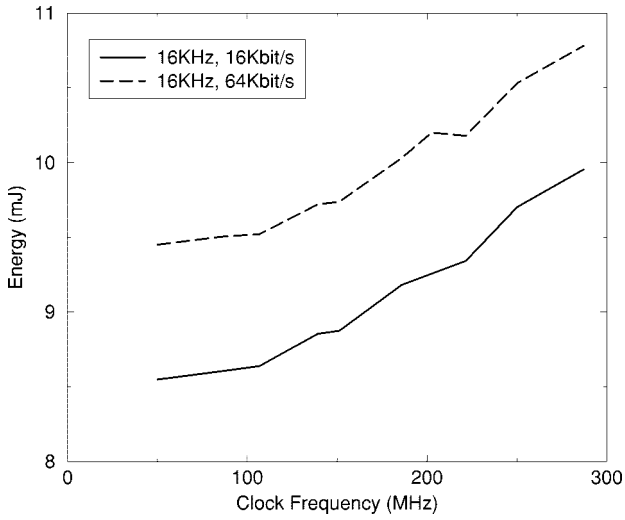

Fig. 2. Energy consumption per frame.

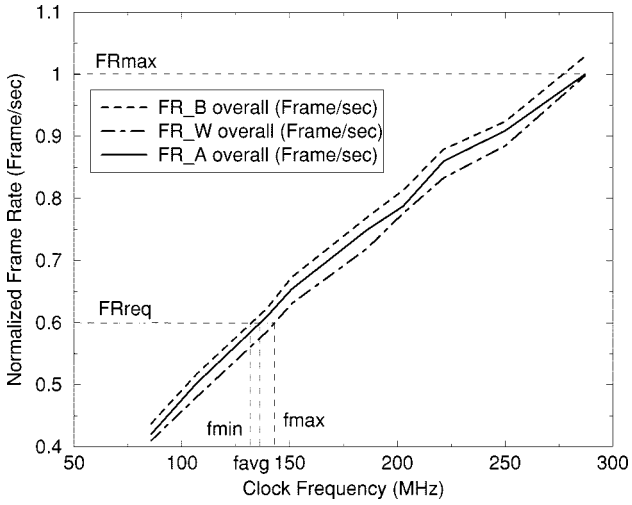

Fig. 3. Frequency setting.

power consumption in idle state is nonnull (50 mW). Hence, idle power is consumed when the processor is idle waiting for the output frame buffer to empty.

Fig. 3 shows the overall FR versus clock frequency curves, obtained with the procedure described in the previous section. All three curves are normalized on the y axis to the $\mathrm{FR}_A(f_{\mathrm{MAX}})$ value. This is all the information needed by the speed-setting algorithm. An FR specification set on the y axis implies three frequency values, shown on the x axis. Remember that the tree curves do not depend on sr and br. Hence, they are a characteristic of the MP3 decode algorithm and can be used with any MP3 stream with the only caveat that the $\mathrm{FR}_A(f_{\mathrm{MAX}})$ must be available and it must be prestored in a lookup table for every possible sr and br.

Finally, Fig. 4 shows the energy penalty paid when running the processor at a clock frequency larger than $f_{\min}$. The solid line shows the energy overhead in the assumption that when the processor is idle it consumes negligible power. The dashed curve shows the actual power penalty, which accounts for processor idle power as well. Notice how the two curves diverge at higher

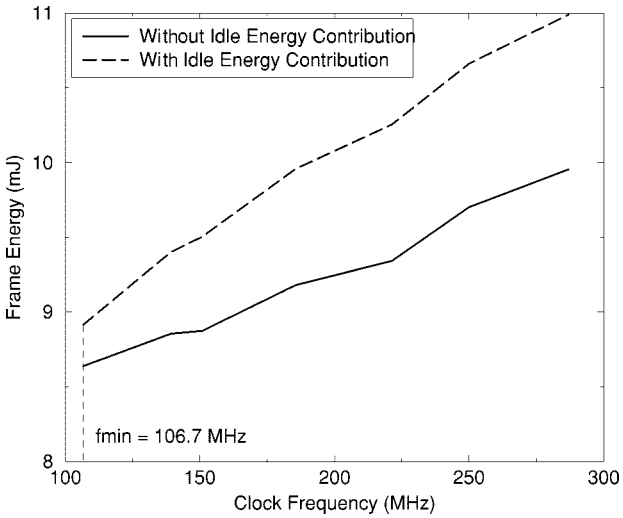

Fig. 4. Energy penalty for a 16-kHz 16-b/s audio stream.

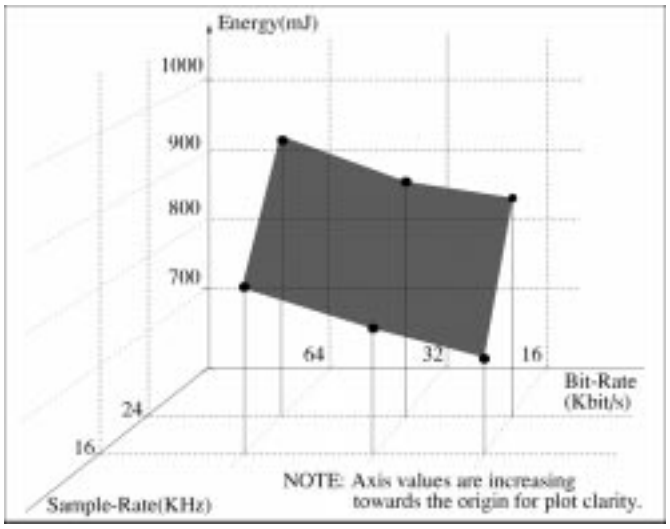

Fig. 5. Energy consumption of speed-optimized algorithm.

frequencies, as the percentage of idle time becomes larger. Energy-per-frame savings of more than 40% are obtained with respect to the trivial policy that clocks the processors always at maximum speed. As an example, consider an audio stream with $\mathrm{sr} = 16$ kHz and $\mathrm{br} = 16$ kb/s. At the maximum frequency and, hence, without optimization, the energy per frame results 10.989 mJ, as shown in Fig. 2.

In order to apply our algorithm, first the FR required $\mathrm{FR}_{\mathrm{req}}$ must be obtained directly by the knowledge of sr. In this case, from (7) it follows that $\mathrm{FR}_{\mathrm{req}} = 27.78$ frame/s. By using the appropriate values of sr and br in lookup table, the corresponding value of $\mathrm{FR}_A(f_{\max}) = 65.36$ frame/s can be obtained, with $f_{\max} = 287$ MHz. This value is used to scale the normalized FR curve shown in Fig. 3. At that point, we find $f_{\min} = 85.7$ MHz, $f_{\max} = 106.7$ MHz as abscissas corresponding to the ordinate $\mathrm{FR}_{\mathrm{req}}$ in the plot of the scaled FR curve obtained above. In this case, f_{avg} is either equal to $f_{\max}$ or $f_{\min}$ because there is not an allowed processor value between 85.7 and 106.7 MHz. Looking at the energy plot of Fig. 4, it can be found that, if we conservatively choose $f_{\max}$, the energy per frame results 8.64 mJ. Hence, we obtain 21% of energy reduction. It must be noted that in this case, $f_{\min}$ is lower than the minimum frequency indicated in the plot of Fig. 4 because the former refers to the best case FR indicated by the FR_B curve in Fig. 3.

B. Adaptive Decoder Results

In the second part of the experimental results, we demonstrate the effectiveness of our approach by evaluating the total energy cost of decoding a compressed audio stream (a pop song). To test the adaptive capability of our algorithm, the same audio stream with different level of compression and sampling rates has been provided. For each of these versions, all the three algorithmic power optimization approaches described in the previous section have been tested. Before describing the results, we must make clear that since the decoding time is fixed, we deal with energy and average power consumption with no distinctions. However, we utilize energy spent to decoding a second of sound as a metric in our numerical results. Energy reduction is computed as $\mathrm{Reduction} = (E_{\max} - E_{\mathrm{opt}})/E_{\max}$. We have utilized a three-dimensional plot to show overall energy behavior of the optimized algorithm with respect to the unoptimized one in Fig. 5, while the experimental results are summarized in Table II.

TABLE II
ENERGY REDUCTION

	Var. Freq	
	16	24
16	0.545	0.315
32	0.482	0.313
64	0.478	0.268
Average	0.400	

Column corresponds to different sample rates in kilohertz, row-to-bit rate in kilobits per second.

In the X-Y plane, we have represented the points corresponding to different versions of the audio stream, while in Z axis, the energy consumption when the policy is applied. The results show how our algorithm adapts to the workload, consuming less energy when computational load decrease. This behavior is in contrast with the one of the unoptimized algorithm, which consumes less energy when bit rate and sample rate increase. This behavior is explained considering that in idle intervals the CPU spends a lot of power polling a synchronization variable. When the workload is higher, the CPU spends more time in decoding instructions, which are less power-expensive. A comparison between the energy consumed by the adaptive algorithm and the unoptimized one is illustrated in Figs. 6 and 7 for an audio stream with sample rates of 16 kHz and 24 kHz, respectively. We note the effectiveness of our policy.

The results of energy reduction are explained by Fig. 8, where we have reported values for a 16 kHz and a 24 kHz sampled audio streams. Each line in the graph corresponds to the energy reduction of the policy applied to audio streams characterized by different level of compression, but fixed sample rate. The energy reduction waveforms are decreasing because the effectiveness of the policy decrease as the workload increases. This is easily explained by considering that the greater the workload,

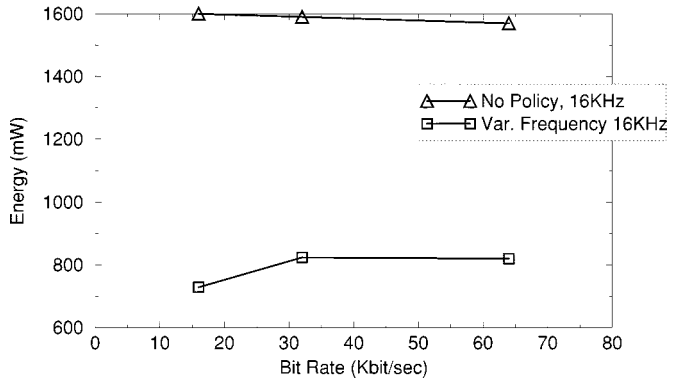

Fig. 6. Energy consumption of adaptive versus nonadaptive algorithm (sample rate: 16 kHz).

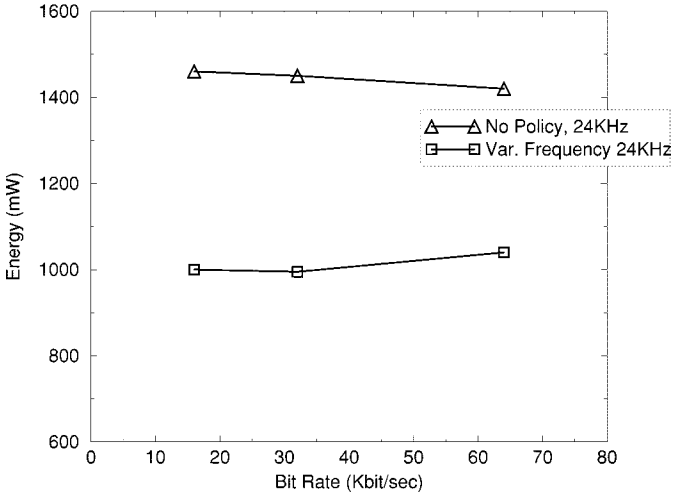

Fig. 7. Energy consumption of adaptive versus nonadaptive algorithm (sample rate: 24 kHz).

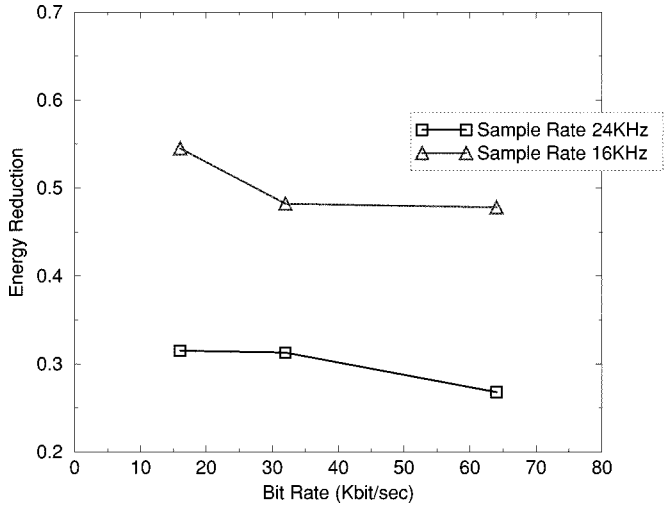

Fig. 8. Energy reduction.

the lesser is the chance of performing an adaptation. Furthermore, coarse frequency adjustments are no more negligible in the case of tight real-time constraints. In addition, the following considerations can be derived.

1) Left points in the graphic, corresponding to a bit rate of 16 kb/s, show the case of a perfect frequency match. Hence, variable frequency policy provides the greatest reduction. This is because, as stated previously in this paper, running at a lower frequency leads to energy reduction not only because of the elimination of useless CPU time, but also because the reduction of the memory latency costs.
2) Points in the middle present a case of imperfect adaptation caused by the impossibility of a continuous frequency tuning. The effect of frequency mismatch is twofold. First, useless energy expensive polling intervals are not completely eliminated. Second, in these intervals, the CPU runs at frequency higher than optimal, further increasing the energy consumption.
3) In right points, we have a case of a nearly optimal adaptation. Hence, speed-setting energy reduction does not decrease sensitively. In effect, it must be recalled that as the workload increases, the effectiveness of the speed-setting policy and, in general, all of the energy management policies based on the workload adaptation decreases.

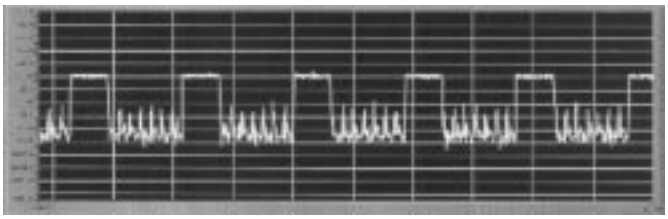

Fig. 9. Energy waveform for a 24-kHz sampled 16-kb/s audio stream.

From these two plots, it can be seen how variable frequency shows lower energy reduction at higher sample rate. This is because of the shape of the energy absorption's waveform, plotted in Fig. 9, with no optimization applied. CPU-waiting intervals are narrow with respect to normal ones and, therefore, little frequency adjustments involve large increase of the their width. As a consequence, we are not able to make a fine regulation of speed-setting effects in order to get a just in time computation.

C. Speed Setting and Shutdown—Experimental Results

In some cases, it is possible to overcome the problem of adaptation mismatch by the application of a joined speed-setting shutdown policy. In effect, when the cost of the mismatch is comparable to the energy that saved by running at the optimal frequency, it can be effective to set the processor clock to an higher than optimal value in order to perform the useful operation in a lesser time and the idleness time interval increases. We increase this interval of a time necessary to allow the processor shutdown.

Results of the application of this policy, which we called mixed, are shown in Fig. 10. Notice that the mixed policy performs better than the pure variable frequency policy when a large adaptation mismatch arises, as in the case of 32 and 64 kb/s bit rate. On the contrary, when the adaptation is good, such as in the 16 kHz bit rate case, variable frequency is more effective because shutdown does not reduce the cost of memory latency, acting only on the explicit idleness contribution, which we have described in Section IV.

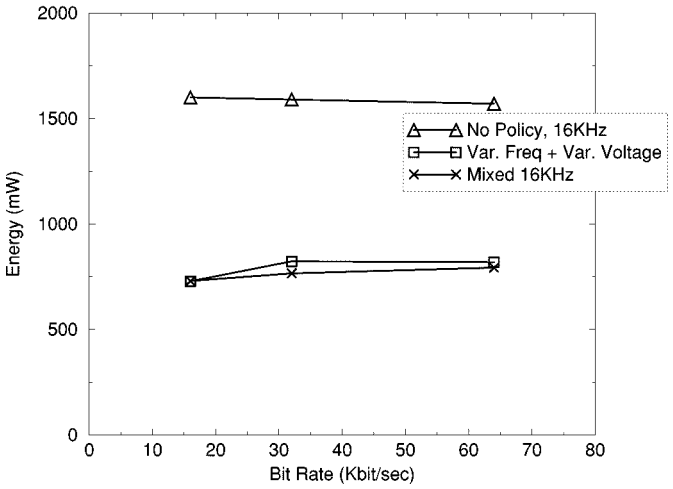

Fig. 10. Energy reduction comparison (sample rate: 16 kHz).

VIII. Conclusion and Future Work

In this paper, we have introduced an approach for automatic runtime setting of the optimum processor frequency that minimizes energy for streaming MP3 audio decoding. The technique has been applied on an embedded portable appliance based on the StrongARM SA-1100 core, obtaining sizable energy-per-frame reduction. Adaptive speed setting is based on a performance versus clock frequency model that is obtained by precharacterization once and for all for a given application. At runtime, FR requirements are obtained by analyzing the MP3 steam header and then a range of acceptable processor clock frequencies is automatically determined based on the performance model.

Future work in this area will focus on speed-setting policies for embedded applications (such as MPEG2 video), where clock speed requirements change rapidly even within a single stream. Variable speed-setting policies that take into account the impact of input and output buffering also warrants further investigation. Speed-setting policies that take into account the impact of I/O buffering in a multitasking environment also warrant further investigation.

References

[1] A. Acquaviva, L. Benini, and B. Riccó, "An adaptive algorithm for low-power streaming multimedia processing," in *Proc. Conf. Design Automation and Test Eur.*, Mar. 2001, pp. 273–279.

[2] D. Jaggar, *Advanced RISC Machines Architectural Reference Manual*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[3] F. Bellosa, "Endurix: OS-direct throttling of processor activity for dynamic power management," Univ. Erlangen, Erlangen, Germany, Tech. Rep. TR-I4-99-03, June 1999.

[4] L. Benini and G. De Micheli, "System-level power optimization: Techniques and tools," *ACM Trans. Design Automat. Electron. Syst.*, vol. 5, no. 2, pp. 115–192, Apr. 2000.

[5] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, pp. 473–484, Apr. 1992.

[6] A. P. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: An approach for energy efficient computing," in *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 1996, pp. 347–352.

[7] D. Dobberpuhl, "The design of a high performance low power microprocessor," in *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 1996, pp. 11–16.

[8] L. Geppert and T. S. Perry, "Transmeta's magic show [microprocessor chips]," *IEEE Spectrum*, pp. 26–33, May 2000.

[9] M. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile application," in *Proc. ACM Symp. Operating System Principles*, Dec. 1996, pp. 48–63.

[10] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithm for dynamic speed-setting of a low-power CPU," in *Proc. Int. Conf. Mobile Computing and Networking*, Nov. 1995, pp. 13–25.

[11] D. Grunwald, C. Morrey III, M. Neufeld, P. Levis, and K. Farkas, "Policies for dynamic clock scheduling," in *Proc. Symp. Operating Systems Design and Implementation*, Oct. 2000, pp. 13–25.

[12] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava, "Power optimization of variable voltage core-based systems," in *Proc. Design Automation Conf.*, June 1998, pp. 176–181.

[13] I. Hong, M. Potkonjak, and M. B. Srivastava, "On-line scheduling of hard real-time tasks on variable voltage processors," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1998, pp. 653–656.

[14] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processor," in *Proc. Design Automation Conf.*, June 1998, pp. 176–181.

[15] C. H. Hwang and A. C. H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1997, pp. 28–32.

[16] C. M. Krishna and Y. H. Lee, "Voltage-clock-scaling adaptive scheduling techniques for low-power in hard real-time systems," in *Proc. IEEE Real-Time Technology and Applications Symp.*, May 2000, pp. 156–165.

[17] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processor," in *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 1998, pp. 197–202.

[18] T. L. Martin and D. P. Siewiorek, "The impact of battery capacity and memory bandwidth on CPU speed-setting: A case study," in *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 1998, pp. 200–205.

[19] T. L. Martin, "Balancing batteries, power and performance: System issues in CPU speed-setting for mobile computing," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, Aug. 1999.

[20] T. Okuma, T. Ishihara, and H. Yasuura, "Real-time task scheduling for a variable voltage processor," in *Proc. Design Automation Conf.*, June 1998, pp. 176–181.

[21] G. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli, "Policy optimization for dynamic power mangement," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 813–833, June 1999.

[22] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proc. Design Automation Conf.*, June 1999, pp. 134–139.

[23] A. Sinha and A. P. Chandrakasan, "Energy Aware Software," in *IEEE Int. Conf. VLSI Design*, Jan. 2000, pp. 50–55.

[24] M. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. VLSI Syst.*, vol. 4, pp. 42–55, Mar. 1996.

[25] I. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. Symp. Operating Systems Design and Implementation*, Nov. 1994, pp. 13–23.

[26] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. 36th IEEE Symp. Foundations of Computer Science*, Oct. 1995, pp. 374–382.

[27] T. Simunic, L. Benini, and G. De Micheli, "Cycle-accurate simulation of energy consumption in embedded systems," in *Proc. IEEE Design Automation Conf.*, June 1999, pp. 867–872.

Andrea Acquaviva received the Laurea degree (*summa cum laude*) in electrical engineering from the University of Ferrara, Ferrara, Italy, in 1999. He is currently working toward the Ph.D. degree in electrical engineering at the University of Bologna, Bologna, Italy.

His current research interests include the design of energy-efficient systems, power-efficient operating systems, and programming environments.

Luca Benini (S'94–M'97) received the B.S. degree (*summa cum laude*) in electrical engineering from the University of Bologna, Bologna, Italy, in 1991 and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1994 and 1997, respectively.

Since 1998, he has been an Assistant Professor with the Department of Electronics and Computer Science, University of Bologna. He is also a Visiting Researcher with Stanford Univesrity and the Hewlett-Packard Laboratories, Palo Alto, CA. He is a member of the organizing committee of the International Symposium on Low Power Design and a member of the technical program committee for several conferences, including Design Automation and Test in Europe and the Symposium on Hardware–Software Codesign. He has authored or coauthored more than 120 papers in journals and conferences, a book, and several book chapters. His current research interests include all aspects of computer-aided design of digital circutis, with special emphasis on low-power applications, and the design of portable systems.

Bruno Riccó (M'85–SM'91) received the Ph.D. degree from the University of Cambridge, Cambridge, U.K., in 1975.

In 1980, he became a Full Professor of Applied Electronics with the University of Padova, Padova, Italy. In 1983, he joined the University of Bologna, Bologna, Italy. He has been a Visiting Professor with Stanford University, the IBM T. J. Watson Research Center, and the University of Washington. He has also consulted for major companies interested in integrated circuit (IC) fabrication and evaluation and for the Commission of the European Union in the definition, evaluation, and review of research projects in microelectronics. In 1996, he became the President of the Italian Group of Electronics Engineerings. In 1999, he was appointed the European representative for the International Electron Devices Meeting (IEDM). He has authored or coauthored over ten publications, three books, and six patents in the field of nonvolatile memories. His previous research interests include solid-state devices and ICs, particulary the understanding and modeling of electron transport, tunnelling in heterostructures and thin insulating films, silicon dioxide physics, MOSFET physics, latchup in CMOS structures, device modeling, and stimulation. His current research interests include IC design, evaluation, and testing.

Dr. Riccó received the G. Marconi Award from the Italian Association of Electrical and Electronics Engineerings (AEI) in 1995. From 1986 to 1996, he was a European Editor of the IEEE TRANSACTIONS ON ELECTRON DEVICES. In 2000, he became Vice-President of the North Italy Section of the IEEE.