An Adaptive Algorithm for Low-Power Streaming Multimedia Processing

Andrea Acquaviva Luca Benini Bruno Riccó

DEIS - Universitá di Bologna
Bologna, ITALY 40136

Abstract

This paper addresses the problem of power consumption in multimedia system architectures and presents an algorithmic optimization technique to achieve the goal of power reduction in the context of real time processing. The technique is based on a mixed speed-setting and shutdown policy. We address the problem from both a theoretical and practical point of view, by presenting a power efficient implementation of a MPEG-layer3 real-time decoder algorithm designed for wearable devices as a case study. The target system is the Hewlett-Packard's SmartBadgeIII prototype of wearable system based on the StrongARM1100 processor. Theoretical analysis as well as quantitative results of power measurements are provided to show the effectiveness of this technique. The experimental set-up is also described.

1. Introduction

One of the most critical constraints on wearable appliance design is power consumption, that is directly linked to battery size, weight and time, which are key design constraints for portable applications. Furthermore, power consumption impacts system cost and reliability. In the design of portable devices, a microprocessor-based architecture is often a forced choice because of its flexibility and fast time-to-market. In these architectures, the CPU must handle a large fraction of the computational load imposed by applications and it is the largest contributor to overall power budget. In general, CPU energy consumption depends on the type of workload imposed by applications. We focus on ultra-portable devices targeted to streaming multimedia applications, such as audio and video decoding.

As mentioned above, CPU power depends on the algorithm to be executed. Indeed, an algorithm can dynamically reconfigure the system to provide the required services and performance levels in a power-efficient way. 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 run time. Hence, it is possible to reduce power consumption by adjusting system speed and supply voltage at the minimum level necessary to match the real-time constraints.

In such a context, the contribution of this paper can be summarized in three points. First we analyze the power consumption of multimedia algorithms which manage power through speed-setting and shutdown, and develop an algorithm which configures the underlaying hardware in order to consume the minimum amount of power required to work correctly with real-time performance. Second, we show from both the theoretical and experimental point of view that variable frequency techniques are useful to save power in the context of streaming multimedia workload, even in the absence of a variable voltage supply. Third, we provide a complete and working implementation of MPEG-layerIII stand-alone decompression algorithm running on the hand-held device HP's SmartBadgeIII. Power consumption measurements of the system when running the decompression algorithm are performed by means of a customized experimental set-up also described in this paper.

2. Related Work

Our work is related with two research areas: (i) analysis and development of adaptive algorithms that exploit workload variability to save power; (ii) power optimization techniques from the point of view of operating system (i.e. involving task scheduling).

Algorithmic power optimization is not a new concept. In [5], Chandrakasan et al. explore dynamic voltage setting in DSP with a variable workload. Differently from custom DSP, general purpose SOCs are not targeted to a particular application, then an adaptation is necessary even with a fixed workload, in order to reconfigure the hardware resources in a power-efficient way. The work by Chandrakasan et al. demonstrate the effectiveness of decreasing together speed and voltage with respect to simply shut down the system in idle periods. Voltage regulation is seen as a generalization of shutdown where the voltage levels are quantized in more than two values (*on* and *off*); from this point of view, variable voltage allows better adaptation to different workloads, hence it is more effective.

Recently, Sinha in [19] has 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 [6], 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.

As for operating-system level power optimization, several techniques have been developed to dynamically control the power of a system. These approaches can be grouped in two categories: based on shutdown and variable-voltage. Shutdown consists in selectively turning on and off components when are or are not to be utilized, while variable voltage techniques are based on dynamic regulation of the supply voltage depending on the level of performance required. Naturally, additional hardware is needed to support this capability (like DC-DC converters). Reducing supply voltage increases circuit delay so that the clock speed must be accordingly reduced [4], thus voltage and speed must be regulated together and a reduction of leads to a cubic reduction of power and quadratic reduction of energy consumption.

The variable-voltage context then, 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 guarantees that all tasks meet their deadlines [21][7]. In 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) [8][10] [12][15][17]. Some of these algorithms assume static scheduling [7][12][13], that is, the workload for each task is characterized at design time, while others are dynamic [8][10][15][17].

Restrictions on the effectiveness of variable voltage arise because of the regulation range is discrete. Moreover, regulation freedom must deals with technology trend towards lower voltages. In addition, variable-voltage requires special hardware, while system shutdown does not. In its simplest flavor this method consists in 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. The 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 knowledges of future input events is available [3]. In [17] Paleologo et al., 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 [11][20].

The assumption at the basis of the variable voltage power management techniques examined above is that power consumption scales down with s^3, where s is the voltage and speed scaling factor. In addition, being the execution time 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 results [2][13][14] on real systems have demonstrated that running at less than the maximum frequency can be advantageous. In the following section we provide a theoretical explanation for this fact, which motivates the adaptive algorithmic power optimization strategy presented later in the paper.

3 Variable Frequency and Energy Optimization

In this section we discuss how energy reduction can be obtained by setting clock frequency, together with shutting down the processor, even in the absence of an associated voltage regulation. This is in contrast with the common assumption that speed-setting is effective only 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:

$$P = V_{DD}^2 \cdot C_{eff} \cdot f \qquad (1)$$

(where V_{DD}^2 is the supply voltage, C_{eff} is the average switched capacitance, and f is the CPU clock frequency). Total energy consumption can be immediately obtained as:

$$E = V_{DD}^2 \cdot C_{eff} \cdot f \cdot T \qquad (2)$$

where T is the total execution time. $T = N_{tot} \cdot t$, where N_{tot} and t are the total number of clock cycle and the cycle time, respectively. We have then:

$$E = V_{DD}^2 \cdot C_{eff} \cdot f \cdot N_{tot} \cdot t = C_{eff} \cdot V^2 \cdot N_{tot} \tag{3}$$

because $f = 1/t$. From the definition of total energy $E = P \cdot T = V_{DD} \cdot I_{avg}$, and comparing with (3), we get:

$$I_{avg} = \frac{V_{DD} \cdot C_{eff}}{T} \cdot N_{tot} \tag{4}$$

where I_{avg} is the average current absorption during execution time T. In the case of streaming multimedia processing, as shown in more details in the next sections, T is fixed (e.g., the duration of a song or a video). Therefore, I_{avg} depends on N_{tot}, i.e., on the workload. Now, if the workload is lower than the maximum one, there are times in which the CPU is idle, so let us now express N_{tot} as:

$$N_{tot} = N_{useful} + N_{idle} \tag{5}$$

We conservatively assume that N_{useful} (the number of cycles spent in execution useful operations) is fixed for a given algorithm, while N_{idle} (the number of cycles wasted with the CPU being idle) can be seen as a function $N_{idle}(f,s)$ where f is one of the available processor frequencies while s is a binary variable indicating whether or not we apply a shutdown policy. N_{idle} is a non-decreasing function of f, and it is lower if s is one (shutdown applied), than in the case $s = 0$ (shutdown not applied).

Furthermore N_{idle} is given by the sum of two contributions. The first, hereafter called *implicit idleness*, identifies CPU idleness finely 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 variation in number of wait states in a bus cycle. This happens when the CPU is not the speed limiting element. The second, hereafter referred to as *explicit idleness*, 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, that can be eliminated without affecting the algorithm effectiveness either by putting the processor in a low-power state while waiting (i.e. adjusting the variable s) or by increasing the time spent in useful operations (i.e. lowering the CPU frequency). Therefore explicit idleness depends both on f and s.

Thus:

$$I_{avg} = \frac{V_{DD} \cdot C_{eff}}{T} \cdot (N_{useful} + N_{idle}(f,s)) \tag{6}$$

with $N_{idle}(f,s) = N_{idle,implicit}(f) + N_{idle,explicit}(f,s)$. The target of power optimization is to reduce I_{avg} acting on N_{idle}, under the constraint:

$$(N_{useful} + N_{idle})\frac{1}{f} = T \tag{7}$$

Shutdown and speed-setting pursue the same target in two different ways: shutdown by stopping the CPU clock when a sequence of idle cycles is to be executed; speed-setting by decreasing f so that in (7) N_{idle} must decrease because T and N_{useful} are fixed. The lower bound (f_{opt}) of f, namely f_{opt} is fixed by the requirement that all the useful work be executed in T (just-in-time computation).

The choice between speed-setting and shutdown depends on the workload characteristics and the system's architecture (both hardware and software). Both these policies have advantages and drawbacks. In particular, speed-setting reduces the costs of memory latency in terms of CPU wait states, while shutdown does not. Hence, in execution dominated by memory access (high miss rate), and where memory latency is higher, this technique is more effective [**?**]. 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 most cases they implement power down and gated clock strategies).

On the other side, since the frequency cannot be adjusted continuously, it is hard to completely eliminate explicit idleness with a speed setting policy. If this "adaptation mismatch" is large, system shutdown can be advantageous. As discussed later, there are cases in which a mixed approach gives best results. In some cases, we can actually set f higher than f_{opt} in order to allow the CPU to go in a low-power idle state during the mismatch interval. As an additional concern when evaluating the tradeoff between frequency setting and shutdown, the delay and the energy spent to force the processor in idle and set its speed must be considered.

4 Multimedia Systems Architecture

In a processor-based architecture, both hardware and software organizations impact power optimization. We briefly describe a specific system, which was used as a practical driver in our study.

4.1 Hardware Architecture

In multimedia stream processing, data come into the system from the external environment by a wireless network or a wired link from a host computer. In the case of interest for this work we have an audio stream of encoded data coming from a serial link connected to a host PC. The main processing unit is a general-purpose microprocessor, integrated in a SOC architecture. The CPU processes a block of data and sends the results towards an external logic when finished. In our case, output data are decoded audio samples sent by an integrated serial controller to an external audio-chip, which performs D2A conversion. To improve efficiency and parallelism, current SOCs contain input-output hardware units that can buffer data and manage standard communication channels (e.g. serial port, parallel port) in parallel with the CPU. High-bandwidth input-output devices often use DMA for enhanced performance. Our target system, the HP SmartBadge is based on the StrongARM1100 core [1], that contains several peripheral circuits that can be controlled by DMA channels.

StrongARM1100, the architecture considered in this work, implements 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 also at run-time. In most architectures, frequency can be programmed via software by writing a control word in a processor register. In StrongARM1100 twelve frequency levels are available by programming a PLL. StrongARM implements two low-power modes: idle and sleep. During idle mode the CPU clock stops, while the rest of the system continues to operate. Power consumption in idle mode depends on the number of peripherals running. Sleep mode it is not useful in our context, because DMA and serial controller are disabled.

4.2 Software Architecture

Multimedia stream processing algorithms must handle variable input and output data rates, hence they make use of software buffering when hardware is not sufficient. In our system, the serial input link is driven by a DMA input channel transferring 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 towards the output channel. The destination of this data depends on the particular nature of the multimedia application. In our case, decoded samples are sent at the appropriate sample rate towards the audio output of the SmartBadge.

5 Energy Optimization

Most of the speed setting techniques proposed in the literature [7][21] [22] are based on a prediction of processor workload in the near future. When the prediction fails, a non-optimal decision is taken, then a cost is incurred in terms of performance and power effectiveness. This "best-effort" approach is not feasible when targeting streaming multimedia. In our case, the multimedia stream must be processed under tight latency and throughput constraints: no user-perceivable interruption of the output stream is tolerated.

To ensure real-time performance, the overhead of frequency setting and processor shutdown must be considered. To explore in more details the tradeoffs involved in frequency and shutdown control, we sub-divide the execution time T for processing the stream in a set of contiguous intervals T_i, $i = 1, \ldots, n$; $T = \sum_{i=1}^{n} T_i$. This partition reflects the frame-based structure of multimedia streams, where T_i is the frame processing time. Inside each T_i, we note that there are times in which the CPU is either explicitly or implicitly idle. Using the notation of Section 3, we can write:

$$\begin{aligned} E = & \sum_{i=1}^{n} V_{DD} \cdot C_{eff} \cdot (N_{i,useful} + \\ & N_{i,idle,implicit}(f) + N_{i,idle,explicit}(f,s)) \end{aligned} \tag{8}$$

We first consider shutdown overhead in energy and delay. Assume we do not change frequency, and we let the processor run at the maximum frequency $f = f_{max}$. Shutting down the processor reduces $N_{i,idle,explicit}(f,s)$ to zero, at best. Unfortunately, shutdown and wakeup have non-negligible energy cost $E_{i,s}$. Thus, we have:

$$\begin{aligned} E = & \sum_{i=1}^{n} V_{DD}^2 \cdot C_{eff} \cdot (N_{i,useful} + \\ & N_{i,implicit}(f_{max,1})) + \sum_{i=1}^{n} E_{i,sd} \end{aligned} \tag{9}$$

Furthermore, it takes a time d to transition the CPU in and out the low-power idle state. To guarantee uninterrupted streaming, we need to ensure that d is always shorter than the time the processor is explicitly idle $T_{i,idle,explicit} = N_{i,idle,explicit} \cdot t$. Hence, the performance and energy constraints for the applicability of processor shutdown are:

$$d \leq \min_i (T_{i,explicit}) \tag{10}$$

$$E_{sd} \leq E_{idle,explicit}(f_{max}) \tag{11}$$

In general, for small $N_{i,idle,explicit}$, the effectiveness of shutdown policy decreases because the costs of this policy in terms of energy and delay are non-negligible. This happens in particular when the workload increases (higher sample rate and bit rate).

We now analyze frequency-setting, assuming that no shutdown is performed. For the sake of illustration, assume that the frequency of the processor can be changed at every frame, and that we can adjust it in a continuous fashion. We call $f_{i,opt}$ the power-optimal frequency for frame i:

$$E = \sum_{i=1}^{n} V_{DD}^2 \cdot C_{eff} \cdot (N_{i,useful} + N_{i,idle}(f_{i,opt}, 0)) + E_{i,vf} \tag{12}$$

where E_{vf} is the energy cost of operations needed to adjust the frequency. Notice that $N_{i,idle}(f_{opt}, 0) < N_{i,idle}(f_{max}, 0)$. To guarantee uninterrupted streaming when applying frequency-setting policy, for each frame we must ensure the minimum frequency, namely f_{opt}, so that the CPU completes the useful work in T_i. As in the case of shutdown, we obtain the following performance and energy constraints:

$$T_i \geq T_{i,useful}(f_{i,opt}) \;\; \forall \;\; T_i \tag{13}$$

$$E_{vf} \leq E_{idle}(f_{max}, 0) - \sum_{i=1}^{n} E_{i,idle}(f_{i,opt}, 0) \tag{14}$$

Unfortunately, in real-life hardware frequency adjustment involves a large delay due to the need of re-synchronization between the CPU and other running on-chip logic. Thus, we cannot change CPU speed frame by frame, but we choose a frequency depending on the overall characteristics of the stream and we keep it constant for the entire execution time T. We name this f_{opt}. Hence the condition (13) becomes:

$$T_{i,useful}(f_{opt}) \leq \min_i(T_i) \tag{15}$$

Since the frequency is not changed every i interval, in our implementation speed-setting is less energy expensive than shutdown. In addition, applicability of speed-setting is less affected by the width of idle intervals, because it increases useful time periods by a percentage of their width. In real systems, however, the fine-adjusting capability is limited by the limited number of frequencies available.

Because of non idealities of speed-setting policy, in some case running at higher than optimal frequency in order to allow the application of shutdown during the remaining $T_{i,idle,explicit}$, can be advantageous. We call this mixed policy. To make this policy feasible, however, the CPU speed is set higher than to the optimal value, in order to compensate for the delay introduced by entering and exiting from idleness.

In our algorithm we implements power-optimization by controlling the bit-rate and sample rate in the header of the incoming audio stream. In function of these values, we search in a look-up table for the optimal frequency and we decide whether or not applying shutdown. Frequency optimal values are calculated off-line by means of several execution traces of different audio streams.

6 Implementation and Measurements

In order to capture the power consumption of the entire system, we measured the current driven by the external power supply to the SmartBadgeIII. The following elements are needed: (i) a DAQ (Data AcQuisition) board, (ii) a I/V conversion board, (iii) a PC with Labview. In the current path from power supply to SmartBadgeIII a I/V conversion board is inserted. The current flow through a 1 ohm resistor; the voltages of the two nodes of the resistor are analog input for the acquisition board. DAQ board operations are driven by Labview software.

6.1 Experimental Results

To demonstrate the effectiveness of our approach, the total energy cost of decoding a compressed audio stream (a pop song) has been measured. 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 $Reduction = (E_{max} - E_{opt})/E_{max}$

The experimental results are summarized in Table 1, while in Figure 1 we have utilized a three dimensional plot to show overall energy behavior of the mixed-policy optimized algorithm with respect to the unoptimized one. We refer to the mixed-policy version because on average it provides the best results.

In the $X - Y$ plane are represented the points corresponding to different versions of the audio stream, while in Z axis the energy consumption when mixed policy applied. The results show how our algorithm adapt to workload, consuming less energy when computational load decrease. This behavior is in contrast with the one of the unoptimized algorithm, which spent more energy when bit rate and sample rate increase. This behavior is explained considering that in idle intervals the CPU spent a lot of power polling a synchronization variable. When the workload is higher, the CPU spend more time in decoding instructions, which are less power-expensive. A comparison between the energy consumed by the different versions corresponding to the different policies implemented in our algorithm and the unoptimized one is illustrated in Figure 2 for an audio stream with sample rate of 16KHz. We note the effectiveness of the policies.

The results of a comparison among the various policies are well explained by Figure 3, where are reported the values of energy reduction for a 16KHz sampled audio stream. Each line in the graph corresponds to the energy reduction of a particular optimization policy applied to audio streams characterized by different level of compression but fixed sample rate.

The following considerations can be derived:

- Left points in the graphic, corresponding to a bit rate of 16Kbit/sec 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.

- Points in the middle present a case in which a pure shutdown policy works better than variable frequency. This is explicable because of the imperfect adaptation caused by the impossibility of a continuos frequency tuning. This observation explain why a policy that utilize both speed-setting and shutdown often gives best results. In this case, even running at higher frequency, we compensate for the energy lost in the mismatch. In effect, 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.

- In right points we have a case of a near good adaptation, hence speed-setting is better than shut-down, while both loose the comparison with mixed, because in this particular case, we found a suboptimal frequency near to the optimal one but allowing the application of shut down.

In general, as the workload increases, the effectiveness of all three policies decreases. This is because shutdown delay and coarse frequency-adjustment are no more negligible in the case of tight real time constraints. In addition, the difference among the three policies is less evident.

Variable frequency show lower power reduction with respect to other policies at higher sample rate. This is because of the shape of the energy absorption's waveform, plotted in Figure 5, with no optimization applied. CPU-waiting intervals are narrow with respect to normal ones, therefore, as stated in section 3 and 5, 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.

One last observation follows from experimental results in Figure 4 which presents variable-voltage extension results. We note how further energy saving can be obtained if hardware platform has this capability. It is also important to observe that variable-voltage variable-speed policy is much more effective than others, included mixed-variable voltage policy. In effect, the costs of running at a suboptimal speed and voltage are larger than in the no voltage-setting case.

	Var. Freq		*Shut Down*		*Mixed*	
	16	24	16	24	16	24
16	0.545	0.315	0.516	0.386	0.543	0.407
32	0.482	0.313	0.499	0.369	0.518	0.376
64	0.478	0.268	0.473	0.336	0.494	0.335
Average	0.400		0.429		0.445	

Table 1. Energy Reduction: column corresponds to different sample rate in KHz, row to bit rate in Kbit/s

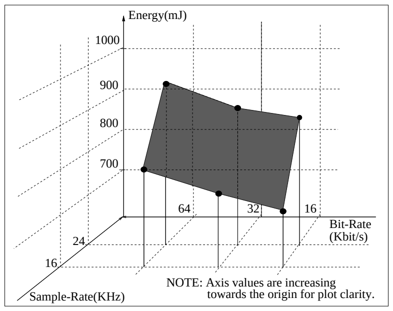

Figure 1. Energy consumption of mixed policy optimized algorithm

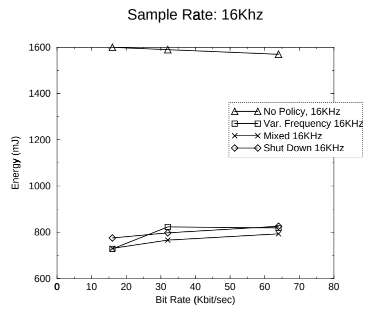

Figure 2. Energy consumption of differents algorithm's versions

7 Conclusions and Future Work

In the context of streaming real time, multimedia signal processing, standard consideration about power optimization policies must be revisited. From our work, we conclude that an adaptive-algorithmic power optimization approach is effective to reduce energy consumption in the context

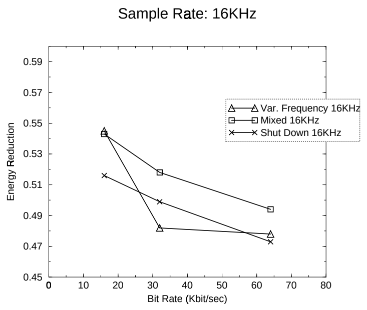

Figure 3. Energy reduction comparison

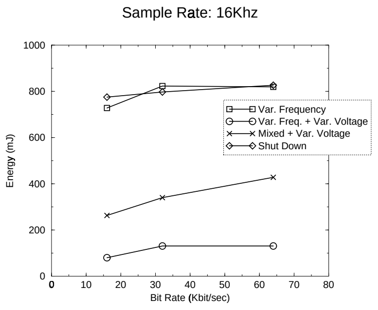

Figure 4. Energy reduction comparison with variable voltage extension

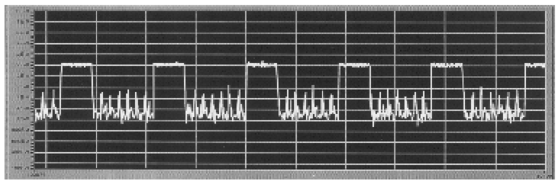

Figure 5. Energy waveform for a 24KHz sampled, 16Kbit/s audio stream.

of multimedia systems and workload. In addition, when targeting real systems, a speed-setting power optimization policy lead to large energy saving even in absence of an associated voltage regulation. Speed-setting overcomes other policies when no frequency adaptation mismatch arises. When a mismatch exists, the solution is to associate speed-setting and shutdown techniques. Future work will be in the analysis of other architectural and algorithmic parameters affecting energy consumption in order to design algorithms with more adaptation capability and improve their power-performance trade-off.

References

[1] Advanced RISC Machines Ltd., Advanced RISC Machines Architectural Reference Manual, *Prentice Hall, New York*, July 1996

[2] F. Bellosa, “Endurix: OS-Direct Throttling of Processor Activity for Dynamic Power Management,” *Technical Report TR-I4-99-03, University of Erlangen*, June 1999.

[3] L. Benini, G. De Micheli, “System-Level Power Optimization: Techniques and Tools,” *ACM, TODAES*, Vol. 5, No. 2, pp. 115–192, April 2000.

[4] A. P. Chandrakasan, S. Sheng, R. W. Brodersen, “Low-Power CMOS Digital Design,” *Journal of Solid State Circuits, IEEE*, Vol. 27 No. 4, pp. 473–484, April 1992.

[5] A. P. Chandrakasan, V. Gutnik, T. Xanthopoulos, “Data Driven Signal Processing: An Approach for Energy Efficient Computing,” *ISLPED*, pp. 347–352, August 1996.

[6] M. Flinn, M. Satyanarayanan, “Energy-Aware Adaptation for Mobile Application” *ACM SOSP*, pp. 48–63, December 1996.

[7] K. Govil, E. Chan, H. Wasserman, “Comparing Algorithm for Dynamic Speed-Setting of a Low-Power CPU,” *International Conference on Mobile Computing and Networking*, pp. 13–25, November 1995.

[8] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, M. Srivastava, “Power Optimization of Variable Voltage Core-Based Systems,” *DAC*, pp. 176–181, June 1998.

[9] I. Hong, M. Potkonjak, M. B. Srivastava, “On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processors,” *ICCAD*, pp. 653–656, November 1998.

[10] I. Hong, G. Qu, M. Potkonjak, M. B. Srivastava, “Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processor,” *DAC*, pp. 176–181, June 1998.

[11] C. H. Hwang, A. C. H. Wu, “A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation,” *ICCAD*, pp. 28–32, November 1997.

[12] C. M. Krishna, Y. H. Lee, “Voltage-Clock-Scaling Adaptive Scheduling Techniques for Low-Power in Hard Real-Time Systems,” *RTAS*, May 2000.

[13] T. Ishihara, H. Yasuura, “Voltage Scheduling Problem for Dynamically Variable Voltage Processor,” *ISLPED*, pp. 197–202, August 1998.

[14] T. L. Martin, D. P. Siewiorek, “The Impact of Battery Capacity and Memory Bandwidth on CPU Speed-Setting: A Case Study,” *ISLPED*, pp. 200–205, August 1998.

[15] T. L. Martin, “Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing,” *Ph.D. Thesis, Carnegie Mellon University*, August 1999.

[16] T. Okuma, T. Ishihara, H. Yasuura, “Real-Time Task Scheduling for a Variable Voltage Processor,” *DAC*, pp. 176–181, June 1998.

[17] G. Paleologo, L. Benini, A. Bogliolo, G. De Micheli, “Policy Optimization for Dynamic Power Mangement,” *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, Vol.18, No.6 pp. 813–833, June 1999.

[18] Y. Shin, K. Choi, “Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems,” *DAC*, pp. 134–139, June 1999.

[19] A. Sinha, A. P. Chandrakasan, “Energy Aware Software” *IEEE Int. Conference on VLSI Design*, pp. 50–55, January 2000.

[20] M. Srivastava, A. P. Chandrakasan, R. W. Brodersen, “Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation,” *IEEE Transaction on VLSI Systems*, Vol.4, No.1, pp. 42–55, March 1996.

[21] I. Weiser, B. Welch, A. Demers, S. Shenker, “Scheduling for Reduced CPU Energy,” *SOSDI*, pp. 13–23, November 1994.

[22] F. Yao, A. Demers, S. Shenker, “A Scheduling Model for Reduced CPU Energy,” *Annual Foundation of Computer Science*, pp. 374–382, October 1995.