# The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms

Trevor Pering, Tom Burd, and Robert Brodersen
University of California Berkeley Electronics Research Laboratory
211-109 Cory #1770, Berkeley, CA 94720-1770
Tel: (510) 642-9350 Fax: (510) 642-2739
{pering, burd, rb}@eecs.berkeley.edu
http://infopad.eecs.berkeley.edu/~lpsw

## 1. ABSTRACT

**The reduction of energy consumption in microprocessors can be accomplished without impacting the peak performance through the use of dynamic voltage scaling (DVS). This approach varies the processor voltage under software control to meet dynamically varying performance requirements. This paper presents a foundation for the simulation and analysis of DVS algorithms. These algorithms are applied to a benchmark suite specifically targeted for PDA devices.**

## 2. INTRODUCTION

Dynamic Voltage Scaling (DVS) allows devices to dynamically change their speed and voltage, increasing the energy efficiency of their operation [2]. Implementing DVS for a general-purpose microprocessor requires substantial software support and new metrics to fully realize and understand the advantages of this capability.

In order to reduce the energy/operation (E) of our system we can increase the delay (D), allowing an associated reduction in our current operating voltage (V) [2]. The approximate relationship between these variables for CMOS is given by:

$$E \propto V^2 \qquad \text{and} \qquad D \propto \frac{V}{V-c}$$

Dynamic Voltage Scaling (DVS) allows a device to dynamically change its voltage while in operation and thus trade-off energy for delay. This allows the processor to provide the minimum *necessary* clock frequency with the maximum possible energy efficiency.



ISLPED 98, August 10-12, 1998, Monterey, CA USA
© 2000 ACM ISBN 1-58113-059-7/98/08…$5.00

Taking advantage of DVS requires algorithms, termed *voltage schedulers*, to determine the operating speed of the processor at run-time. This paper evaluates some previously proposed algorithms in a simulation environment consisting of an energy-accurate cycle-level simulator.

Designing a microprocessor for DVS also requires substantial optimization and redesign at the circuit level [2]. However, in this paper we focus on the evaluation of the algorithms for choosing the appropriate performance and voltage level only.

We specifically target a PDA-class device with respect to system configuration, workloads, and metrics [9]. Our target platform is described in Section 7; our results, however, are applicable to other computing devices, such as laptop computers and embedded microprocessor systems. For our analysis, we use a benchmark suite consisting of applications appropriate for a portable microprocessor system. We have developed a clipped-delay metric which allows us to effectively interpret the results from our simulations.

## 3. PREVIOUS WORK

Voltage scaling has been previously investigated in the context of UNIX workstation systems [10][6]. System traces of UNIX workloads were analyzed and evaluated with a simple energy/delay model. They considered the effectiveness of a number of DVS algorithms on system energy consumption. We will use these algorithms to demonstrate our evaluation approach.

Voltage scheduling has also been applied to DSP applications (e.g. an IDCT algorithm required for MPEG decompression) [4]. DSP applications typically have predictable computational loads with a pre-determined upper limit on computation required to be performed in a given time interval. These characteristics allow relatively easy estimates of the potential benefits of DVS. We use MPEG decompression as a benchmark, along with other, less structured, computation.

## 4. ALGORITHMS

This section describes the algorithms analyzed in this paper. We restrict analysis to *interval-based* voltage schedulers, which rely solely on the global state of the system and do

not have knowledge of individual task threads in the system. If the previous interval is excessively idle, for example, an increase in the processor speed might be effected.

We compare the algorithms against the theoretical optimum, calculated using post-simulation trace analysis. We have found that restricting the voltage schedulers to interval-based activity results in unacceptably sub-optimal operation. This deficiency motivates our future work, which will attempt to exploit information from a lower level.

## 4.1 Framework

In this paper, all algorithms are executed within the same general framework: an interval-based repeating loop. At each iteration, a calculation of the system activity over the proceeding interval is made and passed to the algorithm core. This data is used by the algorithm core which makes a call to the operating system to set the target operating speed for the processor. For our simulations, the interval length parameter varies from 5 ms to 100 ms.

Algorithms in [6][10] make use of the quantity *excess_cycles* which is defined as the number of uncompleted execution cycles from a given interval due to a reduced speed setting. For example, if a given trace interval was found to have 70% activity when running at full speed, and their processor was scheduled to be running at 50% speed for that interval, they assign the value of excess_cycles to be 20%. This value (20%) would then be used by the scheduling algorithms to calculate the speed setting for the next interval.

The use of excess_cycles is intractable in that a real executing algorithm could not know that 70% activity was expected, and therefore could not use the 20% value to calculate the next speed setting. We modified the implementations of these algorithms from their original specification, removing the dependence on *excess_cycles*.

## 4.2 Algorithm Description

We analyze four different algorithms in this paper. Two of the algorithms, FLAT and COPT, are used as baselines for analysis; PAST and AVG are taken from [6] and are the 'real algorithms' analyzed.

- **FLAT<speed>** is what would happen without voltage scaling. The operating voltage is fixed at a constant level and further scheduling is disabled.
- **COPT** is the theoretical optimum operating point generated by post-simulation trace analysis and is used for comparison with realizable algorithms. COPT (*clipped-optimal*) is the minimum energy at which a system can obtain a given delay. See Section 8.1 for a more detailed explanation.
- **PAST**: This simple algorithm was the baseline for the analysis in the [6] paper. The algorithm limits itself to analysis of the previous interval only: if the system was near busy it increases speed, if it was near idle it decreases speed.
- **AVG<weight>**: computes an exponentially moving average of the previous intervals. At each interval the run-percent from the previous interval is combined with the previous running average, forming a long-term prediction of system behavior. <weight> is the relative weighting of past intervals relative to the current interval (larger values mean a greater weight on the past) using the equation $(weight \times old + new)/(weight + 1)$. It was found that the results were weakly dependent on moderate values of the weight parameter and thus a value of 3 will be used throughout this paper.

# 5. BENCHMARKS

To compare the algorithms, we have created a benchmark suite targeted specifically at PDA-class devices. Standard batch-oriented benchmarks, such as compilation or simulation, are abandoned in favor of applications one might find on a PDA. We employ three benchmarks, each with a contrasting workload:

- **Address Book (UI)** User-interfaces differ from data processing [5] in that they are characterized by an alternation between idle (waiting for the user) and maximum performance processing. The processing bursts are typically small, such as drawing an activated button, but are occasionally larger, such as a spreadsheet update. Ideally, the delay for these interactive applications should be reduced to the point at which the user is no longer is aware of them or other limitations become dominant (such as communications bandwidth). We use a simple address book application with operations such as searching, update across a wireless connection, and simple editing.
- **Real-Time Audio (AUDIO)** This benchmark is an encrypted (IDEA) stream of high-quality audio running at 512Kb/s. Data is brought into the processor, decrypted, and then transferred to an external audio device. Running at full processor speed the processing consumes approximately 20% of the processor's cycles. There is every reason to expect an optimal algorithm: the workload is predictably constant with equally sized packets arriving at a constant rate.
- **MPEG Decompression (MPEG)** The core of this application is an inverse discrete cosine transform (IDCT) which is used to decompress MPEG video frames. From frame to frame, the decompression work required varies widely due to the number of IDCTs required, however, the time to available process any given frame is constant at 71 ms (14 frames/sec). This workload is similar to that found in [4].

# 6. METRICS AND DEFINITIONS

The choice of metrics is critical to obtain meaningful comparisons of algorithm. The most obvious metric for our evaluations, an energy/delay graph, was found to be misleading in that it penalizes increases in delay even when there is no real penalty. For example, if an MPEG frame is processed in a record-breaking 1 ms, there is still 70 ms of excess time waiting for the next frame (assuming a 71 ms frame rate). If instead the frame is processed in exactly 71 ms there is no

noticeable delay increase to the user. A similar effect occurs with UI events such as mouse clicks or pen movement with visual user feedback.

To overcome these limitations we have created the clipped-delay metric, described below, to best evaluate DVS. For comparison, our results are normalized to the processor running at full speed without DVS (FLAT<100>).

## 6.1 Events

We divide the processing of each benchmark into subunits called *events*. Events are benchmark specific entities that signify an occurrence that requires computation. The event definitions for each benchmark are as follows:

- **UI**: A UI event starts at the moment of a mouse action, i.e. button down, and it is completed the next time the application enters the idle state. An event, therefore, includes all application-specific processing as well as the cycles necessary for UI functions (drawing buttons, etc...). For this benchmark, the events are variable length; furthermore, the spacing between events is effectively random as it depends on human interaction.
- **AUDIO**: The event definition for AUDIO is the processing of one complete incoming network audio packet (1024 bytes). These events are fairly regular: all packets are the same size and they arrive at fixed intervals.
- **MPEG**: An MPEG event is defined as the decoding of one complete video frame. These events have a constant period (71 ms for 14 frames/sec) and a variable workload. This variable workload is primarily due to the variable number of IDCTs required to generate each frame.

## 6.2 Sufficiency, Delay, and Energy

An algorithm is considered *sufficient* if it allows benchmarks to execute "correctly." It is possible, for example, for an algorithm to run the processor too slowly, causing the loss of data due to exhausted buffers or a real-time latency bound to be exceeded.

Figure 1 shows delay impulse graphs for each of the three benchmarks. Each vertical line represents the delay for that event with the processor running at full speed. We measure *delay* as the sum of all event times for a given application. The x-axis shows only the ordering of the events, not necessarily the delay between them; not all events are shown as the complete data set would be cluttering.

The *energy* consumed is the sum of the energy all the individual event energies. The absolute (un-normalized) energy per instruction of the processor is discussed in Section 7.1.

## 6.3 Clipped-Delay Metric

We have developed the *clipped-delay* metric which takes into account the potential that some events can have their delays increased (voltage reduced) without effecting the user[1]. Any event completed before its deadline, $\alpha$, effectively completes at its deadline. We therefore define

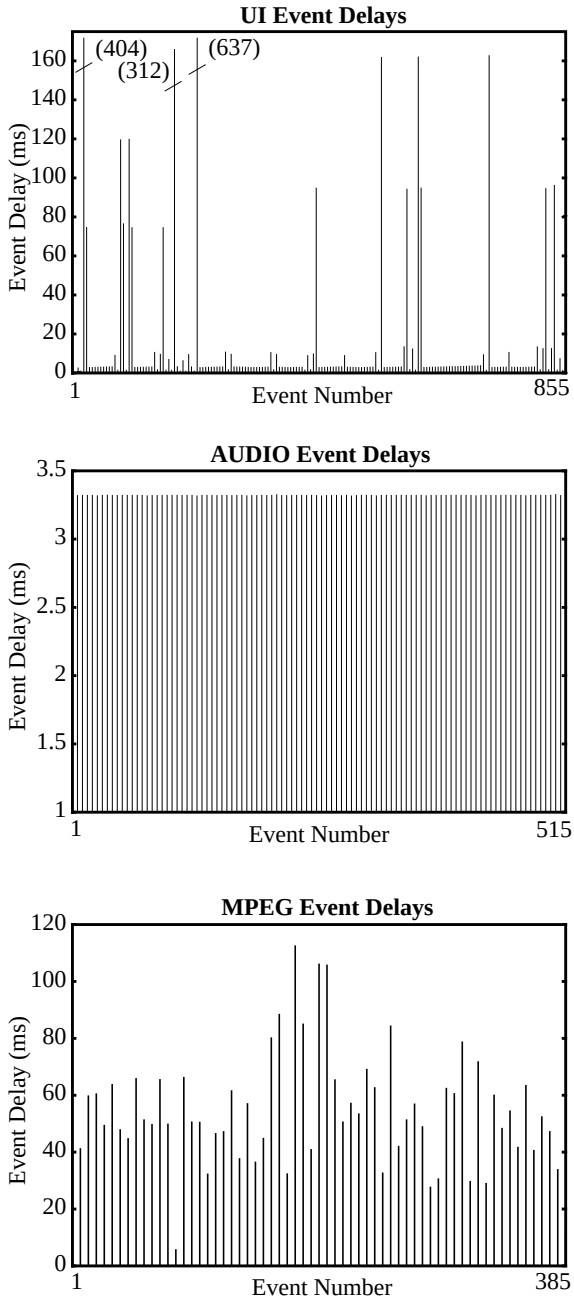


**Figure 1: Event Delay Impulse Graphs**

*clipped-delay* with deadline $\alpha$ to be:

$$C_\alpha = \sum_{events} \mathrm{MAX}[D_i, \alpha]$$

For human interactive tasks (UI), the deadline is dependent on human perception time. A lower bound of 10 ms given from the limitations of human visual [7] and audio percep-

1. [5] discusses a similar metric for the bechmarking of workstation user-interfaces.

**System Summary**

| | |
|---|---|
| Processor Core | ARM8 |
| Cache Configuration | 16K unified, 4-way writeback |
| Memory | 1Meg SRAM |
| External Comm | 1Mb/s, half-duplex |
| Display Device | 640x480, 8-bit Color |
| Input Device | Pen |

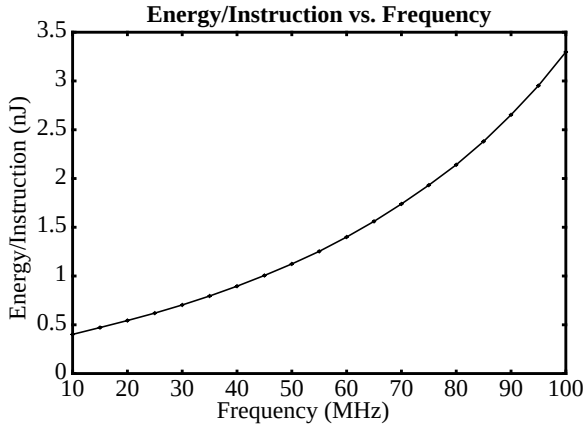


**Figure 2: Hardware Specifications**

tion, while an upper bound [8] is user and application dependent. For these simulations we use a value $\alpha = 50$ ms.

For data processing events (AUDIO and MPEG), we use the time from the start of one event until the start of the next as the event deadline. If an event is completed before the next one arrives there is nothing left for the processor to do: it must idle. Processing continuing past the deadline has the unfortunate effect of requiring additional data storage for buffering the incoming data while the current data set finishes processing.

# 7. ENVIRONMENT

Our research group is currently under development of a DVS hardware implementation. To ensure that our hardware implementation delivers all the necessary functionality, we have developed a comprehensive simulation environment. Algorithms are simulated in this environment and make use of special additions targeted for DVS. This section describes the system and highlights these modifications.

## 7.1 Hardware Model

Figure 2 is a summary of our simulated system. Our microprocessor is based on the ARM8 microprocessor core [1]. Included in our energy model is the core, cache, external processor bus, and external memory devices. Not measured in our energy simulations are external IO devices such as a communication channel, display device, and user input devices. These devices are simulated at a functional level, the delay through the communication channel is modeled, but no energy information is predicted.

In the hardware under development, the processor clock is generated by a ring oscillator driven by the operating voltage. To effect DVS, the current clock frequency is compared against the target frequency; the result is used to dynamically adjust the operating voltage. The expected worst-case settling time for this feedback loop is 10*us*.

Hardware counters, such as cumulative sleep time and active cycles, have been added and are available to the software. These counters allow DVS algorithms to accurately determine the level of system activity at run-time.

The implementation of DVS requires the redesign of some critical hardware components in a microprocessor system. The description of these modifications is beyond the scope of this paper. However, we estimate that these modifications will increase the energy/operation of the processor on the order of 10% over a system not capable of DVS.

## 7.2 Software Model

Our benchmarks use ARM's standard runtime library distribution extended with multi-threading support: the system is multi-threaded uni-process with a single address space. The thread scheduler (which thread runs when) is implemented as a preemptive round-robin scheduler without priorities. We have ported various libraries from the public domain (cryptography, MPEG, graphics, etc...) to support out applications.

Our OS also supports a "sleep until system activity" function for scheduler threads. This function allows a thread to sleep until some other system activity is detected, thereby preventing the system from wasting energy while idle.

## 7.3 Input Model

Simulated user inputs, such as mouse actions, are trace-driven. Activity traces recording the time and location of user actions were recorded in real-time and can be played back during simulation. Data for the AUDIO and MPEG benchmarks is sent through the simulated 1Mb/s communication channel.

Data flowing into the system causes the simulation of the complete input data path: an interrupt is taken and data is read from an external IO device. This level of detail is necessary to ensure comprehensive results because DVS analysis is fundamentally sensitive to timing fluctuations.

# 8. RESULTS

In this section we present graphs of the different algorithms applied to our benchmark suite. For simplicity, we show only one parameterization of the AVG algorithm (AVG<3>).

## 8.1 The COPT Curve

Using the (optimal) COPT algorithm, we can generate a plot

of the minimum energy for any given (clipped) delay (Figure 3). To generate this curve we start with the processor running at full speed and increase the delay (by decreasing the voltage) of each event so that it completes exactly on deadline. For those events already over deadline, the voltage is left unchanged. This process generates the lowest clipped--delay (highest energy) point for this curve. The remaining values on this curve corresponding to increased delay (lower energy) are generated by restricting the processors speed to be below its maximum.

We define sufficiency for this case to be that there is a maximum of two events overlapping at any given time; this is equivalent to requiring a maximum of one events worth of buffering (e.g. AUDIO packet or MPEG frame).

## 8.2 Energy/Clipped-Delay Graphs

Figure 3 contains the energy/clipped-delay graphs for all our benchmarks. For PAST and AVG, we generate curves by varying the interval length from 10ms to 100ms (in 10ms increments). For FLAT, we vary the constant speed setting from 10 MHz to 100 MHz (in 5 MHz increments). The lowest-delay point for the PAST and AVG curves corresponds to the 10ms parameterization, while the lowest-delay point for the FLAT curve corresponds to 100 MHz.

A brief analysis of the results is given below:

- **UI**: The performance of the algorithms depends heavily on the increase in delay allowed. For no increase (1x), only COPT offers a 20% reduction in energy. At a delay factor of 1.2x, the results for all algorithms vary between a 30% and 50% reduction in energy consumed.
- **AUDIO**: Because of its regularity, we should expect results close to optimal. Indeed, the AVG algorithm achieves an 82% decrease in energy while the optimal is 84%, allowing for a small increase in delay.
- **MPEG**: The variability of this workload causes difficulty for the DVS algorithms: the AVG algorithm achieves a maximum 24% reduction while the optimum is a 60% reduction in energy.

The vertical portions of the energy/clipped-delay graphs of Figure 3 represent areas where energy was saved due to voltage reduction that caused no effective performance degradation. For example, stretching an AUDIO event so that it completes on deadline, instead of before deadline, would have this impact.

For the two real algorithms (PAST and AVG) there is an overhead of approximately 1% (in energy) for running the scheduling algorithm with an interval of 20 ms. This number is small enough that it does not significantly affect the results.

## 8.3 Summarized Benchmark Performance

Figure 4 summarizes the simulated results. Percentages indicate the energy consumed relative to the processor running at full speed. The final column is an average of each algorithm across all the benchmarks, assuming an equal weight for each activity.

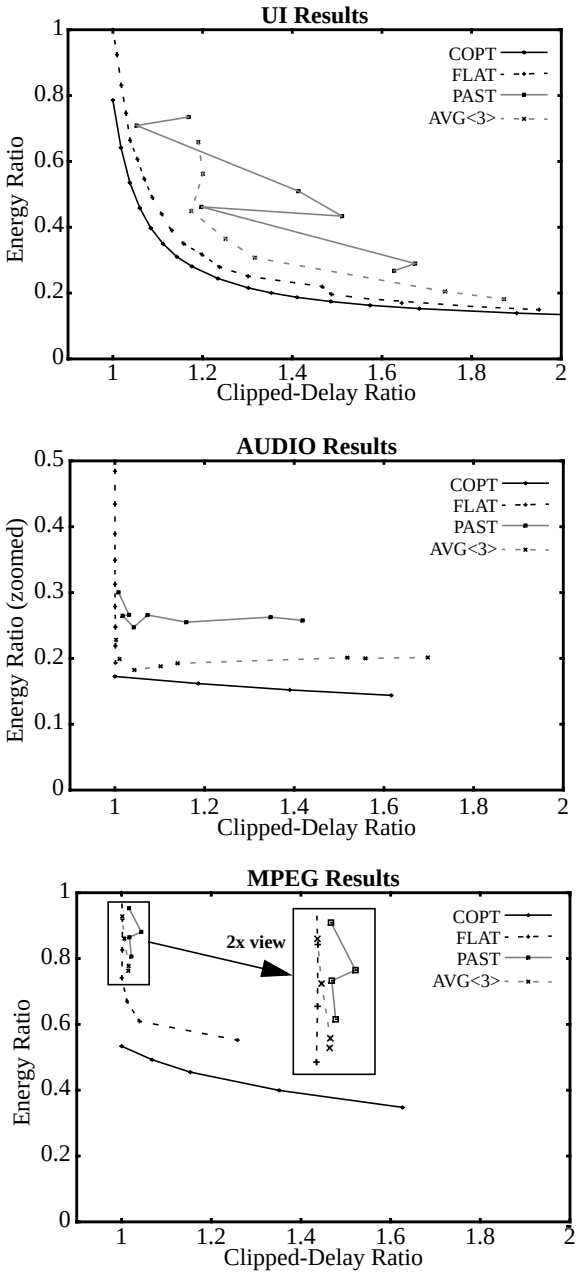


**Figure 3: Energy/Clipped-Delay Graphs**

To generate this table, we have fixed each algorithm with a specific parameterization. The PAST algorithm was fixed with an interval length of 10 ms, while the AVG<3> algorithm was fixed at interval length 20 ms. For the COPT data, which is not interval based, we chose a clipped-delay factor of 1.1x as the intercept point. Although this table is useful for generalizing results, it is dangerous to draw strong conclusions based on these parameterizations. It is unclear, for example, if these parameterizations will be effective for a benchmark with different workload characterizations.

From Figure 4, one can see that the results of these interval-based algorithms have the potential to offer significant

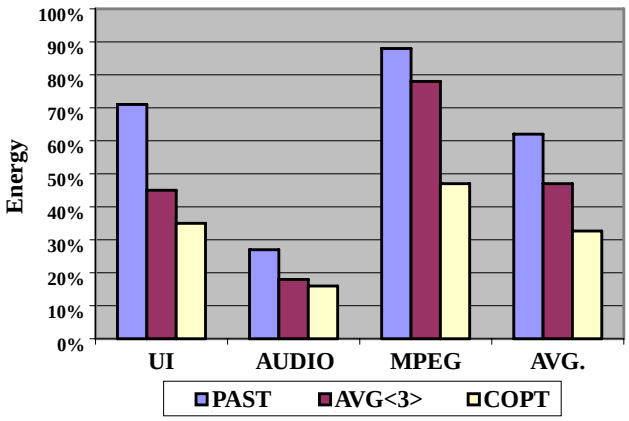


**Figure 4: Energy Summary**

improvements over a system without DVS. However, there is still much to be gained by an improved algorithm for benchmarks with complex workloads, such as MPEG and UI.

## 9. CONCLUSION AND FUTURE WORK

We have implemented and analyzed several DVS algorithms on benchmarks applied to PDA devices. These algorithms reduce system energy by about 46% while maintaining the peak performance demanded by general purpose systems.

Our detailed simulation environment incorporates several unique elements necessary for accurate run-time DVS modeling; without these elements we would be limited to artificial post-trace analysis techniques. Simulation ensures that the hardware will support all necessary features to support DVS algorithms.

The algorithms analyzed fall short of optimal, due to the restricted environment afforded by the interval approach to scheduling. Additionally, a specific choice of parameters for the algorithms is necessary, which is difficult to justify across *all* applications. An adaptive strategy which does not require such parameterization is clearly required.

There are two approaches to increasing the effectiveness of DVS algorithms. First, we can analyze the system behavior with attention placed on the behavior of individual threads. Each thread could be tagged with an expected workload to be used by a voltage scheduler when that thread is queued to run.

Alternatively, application developers could annotate their applications with hints to the OS of expected behavior. This second technique is likely to produce better results than the first; unfortunately, it requires modification of the application source.

## 10. ACKNOWLEDGEMENTS

This work was funded by DARPA and made possible by cooperation with Advanced RISC Machines Ltd (ARM). The authors would like to thank Jeff Gilbert and Jason Wu for their help with this paper. Special thanks to Eric Anderson for his insightful comments and discussions.

## 11. References

[1] *ARM 8 Data-Sheet*, Document Number ARM DDI0080C, Advanced RISC Machines Ltd, July 1996.

[2] T. Burd and R. W. Brodersen, "Energy efficient CMOS microprocessor design," *Proc. 28th Hawaii Int'l Conf. on System Sciences*, Vol.1, pp. 288-297, Jan. 1995.

[3] A. Chandrakasan, S. Sheng, R. W. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, Vol. 27, pp. 473-484, Ap. 1992

[4] A. Chandrakasan, V. Gutnik, T. Xanthopoulos, "Data Driven Signal Processing: An Approach for Energy Efficient Computing," *Proc. 1996 Int'l Symp. on Low Power Electronics and Design*, Aug 1996.

[5] Y. Endo, Z. Wang, J. B. Chen, and M. Seltzer, "Using Latency to Evaluate Interactive System Performance," *Proc. 2nd Symp. on Operating Systems Design and Implementation*, Nov. 1996.

[6] K. Govil, E. Chan, H. Wasserman, "Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU", *Proc. 1st Int'l Conference on Mobile Computing and Networking*, Nov 1995.

[7] C. J. Linbald and D. L. Tennenhouse, "The VuSystem: A Programming System for Compute-Intensive Multimedia," *IEEE Journal of Slected Areas in Communication*, 1996.

[8] B. Shneiderman, *Designing the User Interface*, Addison-Wesley, 1992.

[9] M. Weiser, "Some computer science issues in ubiquitous computing," *Communications of the ACM*, Vol. 36, pp. 74-83, July 1993.

[10] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Proc. 1st Symp. on Operating Systems Design and Implementation*, pp. 13-23, Nov. 1994.