Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System

Trevor Pering, Tom Burd, and Robert Brodersen
University of California Berkeley, Electronics Research Laboratory
{pering, burd, rb}@eecs.berkeley.edu
http://infopad.eecs.berkeley.edu/~pering/lpsw

Abstract

This paper describes the design of a low-power microprocessor system that can run between 8Mhz at 1.1V and 100MHz at 3.3V. The ramifications of Dynamic Voltage Scaling, which allows the processor to dynamically alter its operating voltage at run-time, will be presented along with a description of the system design and an approach to benchmarking. In addition, a more in-depth discussion of the cache memory system will be given.

1. Introduction

Our design goal is the implementation of a low-power microprocessor for embedded systems. It is estimated that the processor will consume 1.8mW at 1.1V/ 8MHz and 220mW at 3.3V/100MHz using a 0.6 µm CMOS process. This paper discusses the system design, cache optimization, and the processor's Dynamic Voltage Scaling (DVS) ability.

In CMOS design, the energy-per-operation is given by the equation

$$E_{op} \propto CV^2$$

where C is the switched capacitance and V is the operating voltage [2]. To minimize E_{op}, we use aggressive low-power design techniques to reduce C and DVS to optimize V.

Our system design, which addresses the complete microprocessor system and not just the processor core, is presented in Section 2. Our benchmark suite, which is designed for a DVS embedded system, is presented in Section 3. Section 4 discusses the issues involved with the implementation of DVS, while Section 5 presents an in-depth discussion of our cache design.

The basic goal of DVS is to quickly (~10µs) adjust the processor's operating voltage at run-time to the minimum level of performance required by the application. By continually adapting to the varying performance demands of the application energy efficiency is maximized.

The main difference between our design and that of the StrongARM is the power/performance target: our system targets ultra-low power consumption with moderate performance while the StrongARM targets moderate power consumption with high performance. Our processor core is based on the ARM8 architecture [1], which is virtually identical to that of the StrongARM. The similarities and differences between the two designs are highlighted throughout this paper.

2. System Overview

To effectively optimize system energy, it is necessary to consider all of the critical components: there is little benefit in optimizing the microprocessor core if other required elements dominate the energy consumption. For this reason, we have included the microprocessor core, data cache, processor bus, and external SRAM in our design, as seen in Figure 1. The energy consumed by the I/O system (not shown) is completely application and device dependent and is therefore beyond the scope of our work. The expected power distribution of our system is given in Figure 2.

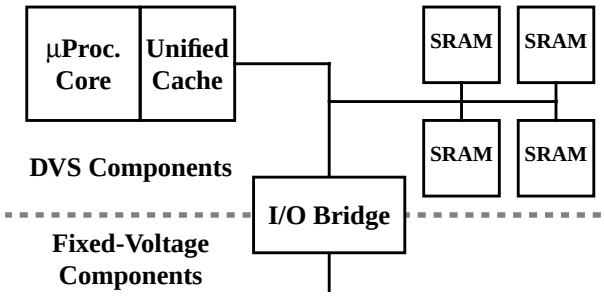

Figure 1: System Block Diagram

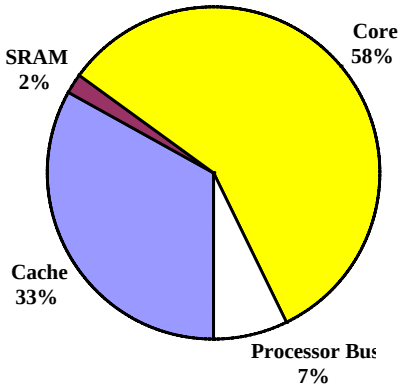

Figure 2: System Energy Breakdown

To reduce the energy consumption of the memory system, we use a highly optimized SRAM design [3] which is 32 data-bits wide, requiring only one device be

activated for each access. Schemes that use multiple narrow-width SRAMs require multiple devices to be activated for each access, resulting in a significant increases in the energy consumption. To alleviate the high pin count problem of 32-bit memory devices, we multiplex the data address onto the same bit-lines as the data words.

We use a custom designed high-efficiency non-linear switching voltage regulator [14] to generate dynamic supply voltages between 1.1V and 3.3V. An efficient regulator is crucial to an efficient system because all energy consumed is channeled through the regulator. When switching from 3.3V to 1.1V, a linear regulator would only realize a 3x energy savings, instead of the 12x reduction afforded by our design.

The threshold voltage (V_t) significantly effects the energy and performance of a CMOS circuit. Our design uses a V_t of 0.8V to achieve a balance between performance and energy consumption. The StrongARM [13], for comparison, uses a V_t of 0.35V, which increases performance at the expense of increased static power consumption. When idle, the StrongARM is reported to consume 20mW, which is the predicted power consumption of our processor *when running at 20MHz*. When idle, we estimate our processor will consume 200µw, an order of magnitude improvement.

3. Benchmarks

Our benchmark suite targets PDAs and embedded applications. Benchmark suites such as SPEC95 are not appropriate for our uses because they are batch-oriented and target high-performance workstations. DVS evaluation requires the benchmarking of workload idle characteristics, which is not possible with batch-oriented benchmarks. Additionally, our target device has on the order of 1MB of memory and lacks much of the system support required by heavy-weight benchmarks; running SPEC95 on our target device would simply be impractical.

We feel the following six benchmarks are needed to adequately represent the range of workloads found in embedded systems:

- AUDIO Decryption
- MPEG Decoding
- User Interfaces
- Java Interpreter
- Web Browser
- Graphics Primitive Rendering

As of this writing, we have implemented the first three of these and their characteristics are summarized in Table 3. “Idle Time” represents the portion of system idle time, used by DVS algorithms. The “Bus Activity” column reports the fraction of active cycles on the external processor bus, an important metric when optimizing the cache system. The cache architecture used to generate Table 3 is discussed in Section 5.

As an example, Figure 4 shows an *event impulse graph* [13], which is used to help characterize programs for DVS analysis. Each impulse represents one MPEG frame and indicates the amount of work necessary to process that frame. For this example, there is a fixed frame-rate which can be used to calculate the optimal processor speed for each frame, assuming only one outstanding frame at any given time.

Benchmark	Miss Rate	Idle Time	Bus Activity
AUDIO	0.23%	67%	0.35%
MPEG	1.7%	22%	14%
UI	0.62%	95%	0.52%

Table 3: Benchmark Characterization

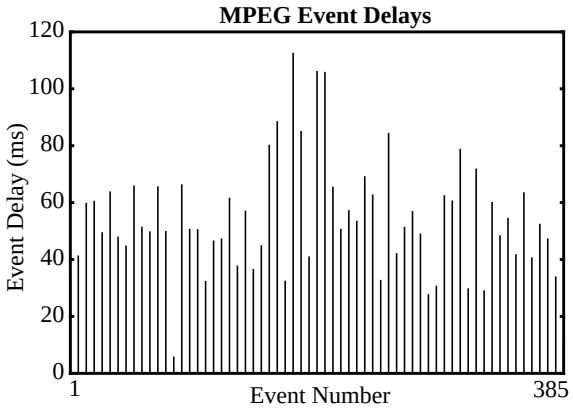

Figure 4: MPEG Event Impulse Graph

4. Dynamic Voltage Scaling

Our processor has the ability, termed Dynamic Voltage Scaling (DVS), to alter it’s execution voltage while in operation. This ability allows the processor to operate at the optimal energy/efficiency point and realize significant energy savings, which can be as much as 80% for some applications [13]. This section discusses DVS design considerations and explains how it affects architectural performance evaluations.

DVS combines two equations of sub-micron CMOS design [2]:

$$E_{op} \propto V^2 \text{ and } f_{max} \propto \frac{V - V_t}{V}$$

where E_{op} is the energy-per-operation, f_{max} is the maximum clock frequency, and V is the operating voltage. To minimize the energy consumed by a given task, we can reduce V, affecting a reduction in E_{op}. A reduction in V, as shown in the second equation, results in a corresponding decrease in f_{max}. A simple example of these effects is given below.

Reducing f_{clk}, the actual processor clock used, without reducing V does not reduce the energy consumed by a processor for a given task. The

StrongARM 1100, for example, allows f_{clk} to be dynamically altered during operation [16], affecting a linear reduction in the power consumed. However, the change in f_{clk} also causes a linear increase in task runtime, causing the energy-per-task to remain constant. Our system always runs with $f_{clk} = f_{max}$, which minimizes the energy consumed by a task.

From a software perspective, we have abstracted away the voltage parameter and specify the operating point in terms of f_{max}. The actual voltage used is determined by a feedback loop driven by a simple ring oscillator. The primary reason for this design was ease of the hardware implementation; fortunately, it also presents the most useful software interface.

Our system applies one dynamic voltage to the entire system to realize savings from all components. It would be possible, however, to use multiple independent supply voltages to independently meet subsystem performance requirements. This was not attempted in our design. To interface with DVS-incompatible external components we use custom designed level-converting circuits.

The implementation of DVS requires the application of *voltage scheduling* algorithms. These algorithms, discussed in Section 4.2, monitor the current and expected state of the system to determine the optimal operating voltage (frequency).

4.1 Energy/Performance Evaluation Under DVS

DVS can affect the way we analyze architectural trade-offs. As an example, we explore the interaction between DVS and the ARM Thumb [4] instruction set. We apply Thumb to the MPEG benchmark from Section 3 and analyze the energy consumed. This example assumes a 32-bit memory system, which is a valid assumption for high-performance systems but not necessarily for all embedded designs.

The MPEG benchmark is 22% idle when running at 100 MHz using the 32-bit ARM instruction set. DVS allows us to minimize the operating voltage to fill unnecessary idle-time. Using a first-order approximation, this would reduce the energy consumed by 40% and slow down the processor clock to the point at which idle time is zero. From this starting point, we consider the application of the Thumb instruction set to this benchmark.

For typical programs, the 16-bit Thumb instruction-set is 30% more dense than it's 32-bit counterpart, reducing the energy consumed in the cache and memory hierarchy. However, due to reduced functionality, the number of instructions executed increases by roughly 18%, increasing the energy dissipated in the processor core as well as the task execution time.

This example will teach two important lessons. First, an increase in task delay directly relates to an increase in energy: DVS exposes the trade-off between energy and performance. Second, an increase in delay affects the *entire* system (core and cache), not just one fragment: it is vital that the associated increase in the energy-per-operation is applied to the entire system.

Figure 5 presents six metrics crossed with three configurations running the MPEG benchmark. The three configurations are:

- **Base:** 78 MHz using 32-bit instructions.
- **Thumb:** 78 MHz using Thumb instructions.
- **Adjusted:** 92 MHz using Thumb instructions.

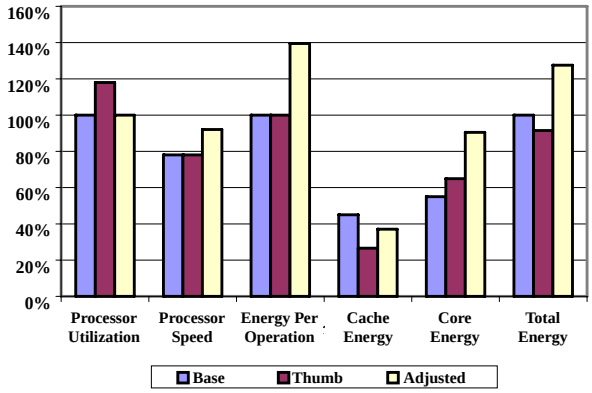

Figure 5: DVS Example

The 'Base' configuration represents the MPEG benchmark running as 32-bit code, as discussed above. 'Thumb' illustrates the intermediate effects of the 16-bit Thumb architecture without increasing the clock speed. The energy consumed in the cache (see Figure 5) decreases due to the decreased memory bandwidth caused by the smaller code size. The energy of the core, however, rises slightly due to the increased number of instructions processed. Overall, the energy decreases by approximately 10%.

The delay increase caused by the expanded instruction stream pushes the processor utilization over 100%. Because of this, the MPEG application will not be able to process its video frames fast enough. The 'Adjusted' configuration represents the increase in processor speed required to maintain performance. This change in clock frequency necessitates an increase in voltage which raises the energy-per-operation of the entire system. As can be seen from the 'Total Energy' columns, the energy savings are no longer realized: the 16-bit architecture increases overall energy consumption.

Although not energy-efficient in all situations, the Thumb instruction set may be efficient for some tasks due to the non-linearity of voltage-scaling. If the base system were initially running at a very low voltage, for example, the increase in processor speed necessary would not dramatically increase the energy-per-operation. The savings due to the reduced code-size, therefore, would affect an overall *decrease* in system energy.

4.2 Voltage Scheduling

To effectively control DVS, a *voltage scheduler* is used to dynamically adjust the processor speed and volt-

age at run-time. Voltage scheduling significantly complicates the scheduling task since it allows optimization of the processor clock rate. Voltage schedulers analyze the current and past state of the system in order to predict the future workload of the processor.

Interval-based voltage schedulers are simple techniques that periodically analyze system utilization at a global level: no direct knowledge of individual threads or programs is needed. If the preceding time interval was greater than 50% active, for example, the algorithm might increase the processors speed and voltage for the next time interval. [5][13][17] analyze the effectiveness of this scheduling technique across a variety of workloads. Interval-based scheduling has the advantage of being easy to implement, but it often has the difficulty of incorrectly predicting future workloads.

More recently, investigation has begun into thread-based voltage schedulers, which require knowledge of individual thread deadlines and computation required [7][12]. Given such information, thread-based schedulers can calculate the optimal speed and voltage setting, resulting in minimized energy consumption. A sample deadline-based *voltage scheduling graph* is given in Figure 6; S_x and D_x represent task start-time and deadline, respectively, while the graph area, C_x, represents computational resources required.

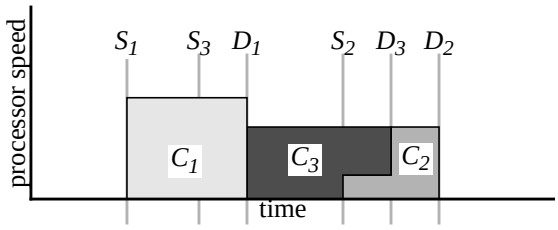

Figure 6: The Voltage Scheduling Graph

4.3 Circuit Level Considerations

At the circuit level, there are two types of components in our design adversely affected by DVS: complex logic gates and memory sense-amps. Complex logic gates, such as 8-input NAND gates, are implemented by a CMOS transistor chain which will have a different relative delay if the voltage is varied. Additionally, memory sense-amps are sensitive to voltage variations because of their analog nature, which is necessary to detect the small voltage fluctuations of the memory cells.

To the largest extent possible, these voltage sensitive circuits are avoided; however, in some situations, such as in the cache CAM design described below, it is better to redesign the required components with increased tolerance. Redesigns of these components will often be less efficient or slower than the original version when running at a fixed voltage. We estimate an increase in the average energy/instruction of the microprocessor on the order of 10%, which is justified by the overall savings afforded by DVS.

5. Cache Design

This section describes the design of our cache system, which is a 16kB unified 32-way set-associative read-allocate write-back cache with a 32-byte line size. The cache is an important component to optimize since it consumes roughly 33% of the system power and is central to system performance. Our primary design goal was to optimize for low-power while maintaining performance; our cache analysis is based on layout capacitance estimates and aggregated benchmark statistics.

Our 16kB cache is divided into 16 individual 1kB blocks. The 1kB block-size was chosen to achieve a balance between block access energy and global routing energy. Increasing the block-size would decrease the capacitance of the global routing but it would also increase the energy-per-access of the individual blocks.

Our cache geometry is very similar to that of the StrongARM, which has a split 16kB/16kB instruction/data cache. Other features, namely the 32-way associative CAM array, are similar. In the StrongARM design, the caches consume approximately 47% of the system power [13].

5.1 Basic Cache Structure

We have discovered that a CAM based cache design (our implementation is given in Figure 7) is more efficient than a traditional set-associative organization (Figure 8) in terms of both power and performance. The fundamental drawback with the traditional design is that the energy per access scales linearly with the associativity: multiple tags and data must be fetched simultaneously to maintain cycle time. A direct-mapped cache, therefore, would be extremely energy efficient; its performance, however, would be unacceptable. We estimate that the energy of our 32-way set-associative design is comparable to that of a 2-way set-associative traditional design.

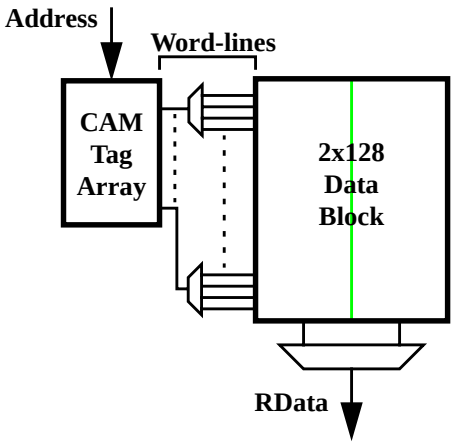

Figure 7: Implemented CAM Cache Design

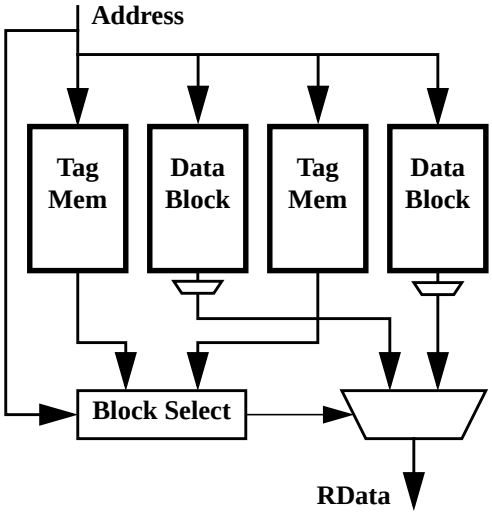

Figure 8: Traditional Set-Associative Cache Design

Our design has been modified from a vanilla CAM in two major ways:

- **Narrow memory bank:** The fundamental SRAM data block for our design is organized as a 2-word x 128-row block, instead of a 8-word x 32-row block.
- **Inhibited tag checks:** Back-to-back accesses mapping to the same cache line do not trigger multiple tag checks.

The 2-word by 128-row block organization for our cache data was chosen primarily because a large block width would increase the energy-per-access to the data block. A block width of 8 words, for example, would effectively entail fetching 8 words per access, which is wasteful since only one or two of these words would be used. The narrow block width unfortunately causes an irregular physical layout, increasing total cache area; however, we chose this design as energy was our primary concern.

There are two natural lower-bounds on the block width. First, the physical implementation of the SRAM block has an inherent minimum width of 2-words [3]. Second, the ARM8 architecture has the capability for double-bandwidth instruction fetches and data reads [1], which lends itself to a 2-word per access implementation.

Unnecessary tag checks, which would waste energy, are inhibited for temporally sequential accesses that map to the same cache line. Using the sequential-access signal provided by the processor core and a small number of access bits, this condition can be detected without a full address comparison. Our simulations indicate that about 46% of the tag checks are avoided with a 8-word cache line size, aggregated across both instruction and data accesses. For the individual instruction and data streams, 61% and 8% of tag checks are prevented, respectively.

5.2 Cache Policies and Geometry

Cache energy has a roughly logarithmic relationship with respect to its overall size, due to selective block enabling: a 16kB cache consumes little more energy than an 8kB cache. Our fundamental cache size constraint was die cost, which is determined primarily by cache area. Benchmark simulations indicate that a 16kB unified cache is sufficient; we felt the increased cost of a 32kB cache was not justified. We chose a unified cache because it is most compatible with the ARM8 architecture.

The cache line size has a wide-ranging impact on energy efficiency; our analysis (Figure 9) indicates that an 8-word line size is optimal for our workload. Given the 1kB block size, our associativity is inversely proportional to the line size: an 8-word line yields 32-way associativity (1kB / 8-words = 32-way). The energy of a CAM tag access is roughly linear with associativity. Also, smaller cache line sizes generate less external bus traffic, consuming less energy. The energy of the data memory is practically constant, although there are slight variations caused by updates due to cache misses.

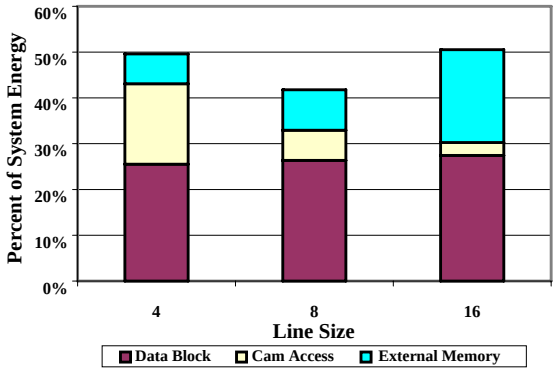

Figure 9: Line-Size Energy Breakdown

We implement a write-back cache to minimize external bus traffic. Our simulations indicate that a write-through cache would increase the external bus traffic by approximately 4x, increasing the energy of the entire system by 27%. We found no observable performance difference between the two policies.

Our simulations find no significant evidence either for or against read-allocate in terms of energy or performance; we implement read-allocate to simplify the internal implementation. Similarly, we find that round-robin replacement performance is comparable to that of both LRU and random replacement, due to the large associativity.

5.3 Related Work

Most low-power cache literature [6][9][15][8] suggests improvements to the standard set-associative cache model of Figure 8. The architectural improvements proposed center around the concepts of sub-banking and row-buffering. Sub-banking retrieves only the required portion of a cache line, saving energy by not extraneously fetching data. Row-buffering fetches and saves an entire cache line to avoid future unnecessary

tag comparisons.

Our CAM-based cache design indirectly implements the concepts of sub-banking and row-buffering. The 2-word block size of our memory bank is similar to 2-word sub-banking. Tag checks inhibition is similar to row-buffering: only one tag-check is required for each cache-line access.

[8] presents a technique for reducing the energy of CAM-based TLBs by restricting the effective associativity of the parallel tag compare and modifying the internal CAM block. Due to time constraints, these modifications were not considered for our design.

6. Conclusion

This paper describes the implementation of a low-power Dynamic Voltage Scaling (DVS) microprocessor. Our analysis encompasses the entire microprocessor system, including the memory hierarchy and processor core. We use a custom benchmark suite appropriate for our target application: a portable embedded system.

Dynamic Voltage Scaling allows our processor to operate at maximal efficiency without limiting peak performance. Understanding the fluid relationship between energy and performance is crucial when making architectural design decisions. A new class of algorithms, termed voltage schedulers, are required to effectively control DVS.

A description of our cache design was given which presents the architectural and circuit trade-offs with energy and performance for our application domain. For minimized energy consumption, we found that a CAM-based cache design is more energy efficient than a traditional set-associative configuration.

Acknowledgments

This work was funded by DARPA and made possible by cooperation with Advanced RISC Machines Ltd (ARM). The authors would like to thank Eric Anderson, Kim Keeton, and Tom Truman for their proofreading help.

7. References

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

[2] T. Burd and R. Brodersen, "Energy Efficient CMOS Microprocessor Design," *Proc. 28th Hawaii Int'l Conf. on System Sciences*, 1995.

[3] A. Chandrakasan, A. Burstein, R. Brodersen, "A Low-Power Chipset for a Portable Multimedia I/O Terminal," *IEEE Journal of Solid-State Circuits, Vol. 29, No. 12, December 1994.*

[4] L. Goudge, S. Segars, "Thumb: reducing the cost of 32-bit RISC performance in portable and consumer applications," *Forty-First IEEE Computer Society International Conference, 1996.*

[5] K. Govil, E. Chan, H. Wasserman, "Comparing algorithms for dynamic speed-setting of a low-power CPU," *Proc. of the First Annual Int'l Conf. on Mobile Computing and Networking, 1995*.

[6] P. Hicks, M. Walnock, and R. Owens, "Analysis of Power Consumption in Memory Hierarchies," *Proc. 1997 Int'l Symp. on Low Power Electronics and Design.*

[7] T. Ishihara, H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," *Proc. 1998 Int'l Symp. on Low Power Electronics and Design.*

[8] T. Juan, T. Lang, J. Navarro, "Reducing TLB Power Requirements," *Proc. 1997 Int'l Symp. on Low Power Electronics and Design*.

[9] M. Kamble and K. Ghose, "Analytical Energy Dissipation Models For Low Power Caches," *Proc. 1997 Int'l Symp. on Low Power Electronics and Design.*

[10] T. Kuroda, et. al., "Variable Supply-Voltage Scheme for Low-Power High-Speed CMOS Digital Design," *IEEE Journal of Solid-State Circuits, Vol. 33, No. 3, March 1998.*

[11] J. Montanaro, et. al., "A 160Mhz 32b 0.5W CMOS RISC Microprocessor," *1996 IEEE Int'l Solid-State Circuits Conf.*

[12] T. Pering and R. Brodersen, "Energy Efficient Voltage Scheduling for Real-Time Operating Systems," *4th IEEE Real-Time Technology and Applications Symposium, 1998, Works In Progress Session*.

[13] T. Pering, T. Burd, and R. W. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms," *Proc. 1998 Int'l Symp. on Low Power Electronics Design*.

[14] A. Stratakos, S. Sanders, R. Brodersen, "A low-voltage CMOS DC-DC converter for a portable battery-operated system," *25th Annual IEEE Power Electronics Specialists Conference, 1994.*

[15] C. Su and A. Despain, "Cache Designs for Energy Efficiency," *Proc. 28th Hawaii Int'l Conf. on System Sciences*, 1995.

[16] M. Viredaz, "Itsy: An Open Platform for Pocket Computing," presentation from Digital Equipment Corporation's Western Research Laboratory.

[17] M. Weiser, "Some computer science issues in ubiquitous computing," *Communications of the ACM*, Vol. 36, July 1993.