Energy Aware Kernel for Hard Real-Time Systems

A. Goel, C. M. Krishna, I. Koren
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003, USA
{agoel,krishna,koren}@ecs.umass.edu *

ABSTRACT

Embedded systems often have severe power and energy constraints. Dynamic voltage scaling (DVS) is a mechanism by which energy consumption may be reduced. In this paper, we implement a dynamic voltage scaling based scheduler in the eCos operating system running on Voltage Scalable Intel Xscale Board and show how energy usage can be reduced while still meeting hard real-time deadlines.

Keywords

Embedded System, dynamic voltage scaling, eCos, real-time operating system, earliest deadline first.

1. INTRODUCTION

Power is becoming a key constraint in embedded systems, with Dynamic Voltage Scaling (DVS) emerging as a powerful mechanism for reducing the energy consumption during system operation [1]. Many current embedded processors such as the Intel XScale [14] and Transmeta Crusoe [15] allow their operating voltage to be changed dynamically. DVS tries to address the tradeoff between performance and power consumption by taking into account the fact that the majority of computer systems need high performance for only a small fraction of the time, while most of the time, a lower performance, lower-power processor would suffice.

Most modern microprocessors use CMOS circuits whose power consumption has been modeled accurately. CMOS circuits have both dynamic and static power consumption. Dynamic power is consumed due to the switching of gates and is still responsible for a large percentage of the total power dissipated in CMOS circuits, although static power consumption, which is due to the leakage current between the power supply and ground, is expected to increase in the future. Power consumption in present-day circuits is dominated by its dynamic component, which can be expressed as [1].

$$P_{cmos} = C_L N_{SW} V_{DD}^2 f \quad (1)$$

where V_{DD} is the CPU core supply voltage, C_L is the circuit output load capacitance, N_{SW} is the number of switches per clock cycle, and f is the clock frequency.

The circuit delay, denoted by δ, obeys the following equation [15]

$$\delta = \frac{C_L V_{DD}}{K(V_{DD} - V_T)^\alpha} \quad (2)$$

where K is a constant depending on the process and gate size, V_T is the threshold voltage, and α varies between 1 and 2; $\alpha = 2$ for long-channel devices which have no velocity saturation.

From (1), reducing the clock frequency linearly decreases the power consumption while reducing the voltage reduces it quadratically.

Equation (2) shows that δ is inversely proportional to V_{DD} (for devices with α=2) and as a result, decreasing the supply voltage increases the circuit delay. Clearly there is a fundamental tradeoff between operating speed (or performance) and supply voltage (or power consumption). For hard real-time systems, where meeting deadlines is critical, this tradeoff is particularly important.

The idea of trading off CPU energy consumption with performance has gained attention in recent years. Unsal and Koren present a comprehensive survey of system-level power-aware design techniques for real-time systems [2]. A study of periodic real-time tasks, which can consume energy at possibly varying rates, is described in [9]. A benchmark suite and simulation environment for voltage scaling is presented in [10]. Non-preemptive scheduling is studied in [18] in a system where voltage scaling overheads were assumed to be negligible. Dynamic voltage-scheduling algorithms for fixed-priority task systems were presented in [5]. A DVS based scheduling algorithm for hard real-time systems using stochastic workload information is described in [27].

Most of the DVS research is based on simulations and not much work has been done on implementing DVS in real-time operating systems. A speculative scheme has been proposed in [11] and implemented in the eCos operating system [23] running on an Intel Xscale-based board. It allows processor speed to be reduced in anticipation of lower execution time, and cannot therefore guarantee that deadlines will be satisfied. Ma and Shin describe energy-adaptive scheduling in mobile applications using their Emerald operating system [13]. In [4], Earliest Deadline First (EDF) and Rate

*This work was supported in part by NSF grant EIA-0102696.

CASES'05, September 24–27, 2005, San Francisco, California, USA.

Monotonic (RM) based DVS algorithms have been developed. These techniques are implemented as Linux kernel modules, and do not guarantee real-time behavior. Energy savings from asynchronous and synchronous clock scaling on IBM PowerPC have been shown in [28], and [29]. DVS has also been implemented in RTlinux [3] based on the EDF algorithm. This work assumes task sets where all tasks have identical periods, are non-preemptible, and run to their worst case execution time.

In this paper, we present the implementation details of an EDF-based DVS algorithm that guarantees meeting all real-time deadlines with tasks which have arbitrary periods, are preemptible, and do not necessarily run to their worst case execution time (WCET). We demonstrate the validity of our algorithm with measurements on an unmodified off-the shelf processor. To the best of our knowledge, this is one of the first working implementations of DVS for hard real-time systems which addresses all these issues.

The rest of this paper is organized in the following manner. Section 2 presents the models that we use and our terminology. Section 3 describes our voltage scaling algorithm. Section 4 provides the details of our hardware prototype and the software implementation of our energy-aware kernel in a Real Time Operating System (RTOS) followed by Section 5 describing the results. The paper concludes with a brief discussion in Section 6.

2. MODELS AND TERMINOLOGY

2.1 Task Model

A typical real-time task model comprises of periodic real-time tasks where each task T_i is characterized by:
P_i : Period of task T_i.
D_i : Relative deadline of T_i ($=P_i$).
$WCET_i$: Worst case execution time of task i under fault-free conditions.
AET_i : Actual execution time of T_i under fault-free conditions.

All the tasks are independent, i.e., no task depends on the output of any other task. Also, tasks must run to completion in order to get any benefit from them, i.e., they do not have the increased reward for increased service property [24].

AET_i is assumed to vary uniformly in the range [$a \times WCET_i$, $WCET_i$], $0 < a \leq 1$. As a increases, the task's execution time becomes more deterministic.

2.2 Power Model

The processor can operate at two voltage levels: v_H and v_L ($v_H > v_L$). The slowdown factor by which the processor is slower at voltage v relatively to when at voltage v_H is:

$$slowdown(v) = \frac{v(v_H - v_T)^2}{v_H(v_H - v)^2} \quad (3)$$

where v_T is the threshold voltage.

We define one unit of execution as the computation performed by the processor at v_H in unit time. Hence, one unit of execution will take slowdown(v) units of time at voltage v. The ratio of energy consumed per cycle by a processor at voltage v relatively to that at voltage v_H is

$$Energy\_ratio\_per\_cycle(v) = \left(\frac{v}{v_H}\right)^2 \quad (4)$$

We have taken into account the cost associated with voltage scaling in our implementation. We calculated an upper bound on this overhead and accounted for it by merging it into the worst-case profile information of tasks.

3. VOLTAGE SCALING ALGORITHM

In this work we implement the algorithm described in [1]. We assume a periodic task set and for each periodic frame (defined as the least common multiple of periods of all the tasks) all tasks are released at the beginning of the frame and must complete by the end of the current frame. Central to this algorithm is the function $\Psi(t)$ which has the property that if the amount of unfinished work at time t is not greater than $\Psi(t)$, there is enough processing capacity to meet all deadlines under the scheduling algorithm that is being used, and thus provides a sufficient condition for schedulability.

3.1 Algorithm

The algorithm consists of two phases, offline and online [1]. In the offline phase, which is executed before the system is actually used, we obtain the function $\Psi(t)$ over the duration of the frame.

In the online phase, when the system is operating, we set the voltage level to guarantee that the unfinished work will never exceed $\Psi(t)$ at time t. This algorithm always sets the processor voltage to v_L when the unfinished work is guaranteed to be below $\Psi(t)$, and v_H otherwise. The offline phase is executed in the following way. Let u be the worst-case utilization of the processor at high voltage (i.e., the utilization if every task ran to its WCET and was run at high voltage). For each task T_i, a corresponding inflated $T_i^{'}$ is constructed with the same period as T_i but whose execution time is deterministic and is equal to $\frac{1}{u}$ times the WCET of T_i. The set of tasks $T^{'} = T_1^{'}, T_2^{'}, T_3^{'}, \cdots, T_n^{'}$ has utilization 1 by construction. We then use the EDF scheduling algorithm to schedule it, and $\Psi(t)$ is the unfinished work according to this schedule.

In the online phase, when a task starts executing (or restarts following a preemption), the worst case unfinished work is compared to $\Psi(t)$. If at $\Psi(t)$, the task is run at v_h until it is completed or preempted by another task. If it is below $\Psi(t)$, we determine the next time τ when the worst-case unfinished work will equal $\Psi(\tau)$. The CPU is run at v_L until either τ is reached or the task has completed or been preempted. When a task is completed or preempted, we recalculate the worst-case unfinished work in the system. If the task is preempted, the current unfinished work is calculated by subtracting the amount of work done since the last time the task was resumed or executed from the unfinished work in the system. If the task is over, the worst-case unfinished work is calculated depending upon the following conditions:

If the deadline of the next task is greater than the deadline of this task (that just finished), we subtract the actual remaining execution time of the task from the unfinished work to get the current unfinished work in the system.

If the deadline of the next task is less than the deadline of the task that just finished, we generate another pseudo task in the system with the same period as that of the task that just finished (say T_i), an execution time of $WCET_i$ - AET_i , and the same deadline as that of T_i. The pseudo task is like any other task, except that the CPU is idle while "execut-

ing" it. Define by the offline_task(t) the task (if any) running at time t under the offline schedule. Let online_task(t) denote the task that is running under the online schedule at time t.

At time t the processor will be running at low voltage except when any of the following conditions is satisfied:
1. offline_task(t) = online_task(t).
2. The worst case unfinished work under the online algorithm is equal to $\Psi(t)$.

If any of these conditions are satisfied, the processor runs at high voltage. The above-mentioned algorithm guarantees that no task will miss its deadline.

3.1.1 Overheads

Voltage-clock scaling decision is taken at the following points:

1. When a task is preempted or completed.
2. When the offline curve intersects the online curve.

The overhead associated with this algorithm can be bounded by the fact that we know the task arrival epochs. Since a scaling action may have to be taken at the very beginning of the execution or at the intersection point, this bound is given by

$$\Omega = n(\omega_{DVS} + \omega_{intersect})$$

where the following notation is used:

n	Number of intersection points and task completions plus 1
ω_{DVS}	Overhead of DVS action
$\omega_{intersect}$	Overhead of calculating an intersection point

4. IMPLEMENTATION DETAILS

4.1 Hardware Platform

We implemented our prototype system on ADI Engineering's Intel Xscale BRH board. The Xscale platform supports nine frequency levels ranging from 200 Mhz to 733 Mhz [14]. CLK, the input reference clock, accepts an input clock frequency of 33 to 66 Mhz. The processor uses an internal PLL to lock to this reference clock and multiplies the frequency by a variable multiplier (changeable through software) to produce a high-speed core clock (CCLK). This gives the software the ability to change the frequency without having to reset the core. Intel Xscale also supports low voltage operation with a core supply as low as 0.95 V. At each voltage level, there is a maximum clock frequency (CCLK), though the core can also operate at lower frequencies if desired. The Xscale CPU core on this board is powered from a supply, created by a switching regulator (LT1767). The supply voltage can be changed by varying a single resistance from 1.2V to 1.8V.

If the feedback resistance (R64 in Figure 1) is short circuited, the supply voltage drops to 1.2 V. As shown in Figure 1, the power regulator circuit of the board is modified by placing an MM74HC4066 (quad analog switch) across the feedback resistance of the power regulator. This switch is controlled through the COM2 port, by connecting the Ready To Send (RTS) line of the serial port(COM2) to the control pin of this switch. The RTS line is either at +5 V or at -5 V. We set the RTS line to +5V to turn on the switch when 1.2 V is required. On the other hand, when 1.5 V is required, we set the RTS line to -5V to turn off the switch. Since the switch turns off when it is at 0 V, we use a diode and a 10 KΩ resistor to clip the voltage to 0 V if the input goes to -5 V.

Using this circuit, a total of two voltage settings 1.2 V and 1.5 V are made available to the kernel. Speed voltage combinations that we have used for our experiments are summarized in Table 1.

Table 1: Voltage-Speed Settings.

Voltage	CPU freq (MHz)
1.2	400
1.5	733

For measuring the input current to the core, the ferrite bead (FB10) is replaced by a small resistance of .01 Ω, which is connected to an Agilent 34401A multimeter [26]. The Agilent 34401A is capable of taking 1000 readings every second. Voltage-clock scaling overheads include the phase-locked loop (PLL) synchronization time of 2000 CPU cycles and, based on our analysis, approximately 40μseconds of voltage stabilization time.

4.2 Software Architecture

We implemented our voltage scaling algorithm in the Embedded Configurable Operating system (eCos). eCos is a highly configurable real-time operating system [23]. Most embedded operating systems today provide more functionality than is usually required by individual applications. This extra code adds more complexity in the software. eCos gives the developer the capability to remove the functionality that is not needed. It has more than 200 configuration points (which can be chosen using a configuration tool) and can be used for fine-grained scalability. eCos can provide the basic functionality of an RTOS with memory footprint in tens to hundreds of KB.

The hardware abstraction layer provides a software layer that gives general access to hardware. The kernel includes interrupt and exception handling, thread and synchronization support, a choice of schedulers, timers, counters and alarms. Device drivers include standard serial, ethernet, Flash ROM and others. eCos also has a GNU debugger which provides communication between target and host. It also includes ISO C and math libraries.

Both eCos and the application run in supervisor mode. There is no division between user and kernel mode in eCos. It is the responsibility of the application developer to take proper care while writing embedded applications as a single improper pointer access can wipe out the whole system. In many operating systems interrupts are disabled during the execution of schedulers: this is not the case with eCos, ensuring low interrupt latency. eCos implements two static-priority-based preemptive schedulers:

1. Multilevel Queue Scheduler: This scheduler allows the execution of multiple threads, each at its priority level. Priority levels can be configured from 0 (highest priority) to 31 (lowest priority). This scheduler allows preemption of a lower priority thread by a higher priority thread and it also supports time slicing within a

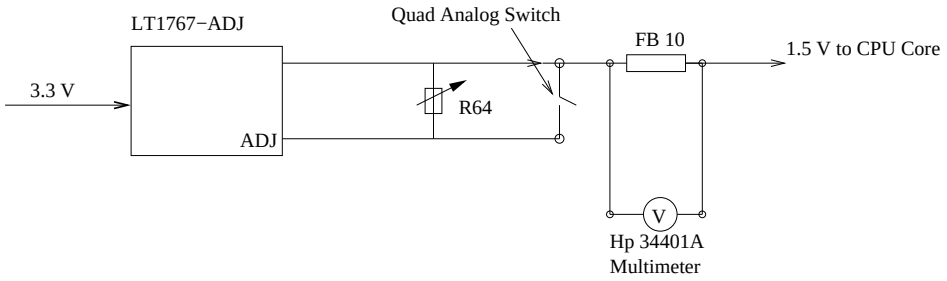

Figure 1: Circuit Diagram.

priority level.

2. Bitmap Scheduler : This scheduler allows the execution of threads at a pre-defined priority level and no two threads can have the same priority level. It also implements preemptive scheduling though time slicing is irrelevant in this scheduler.

Currently available eCos schedulers do not have the notion of periods and deadlines, and all of them are fixed priority-based schedulers. We have implemented a dynamic priority based scheduler in eCos, based on the earliest deadline first (EDF) policy [17]. Priorities are therefore assigned at run time based on the tasks' deadlines. We call it the Power Aware EDF (PA-EDF) Scheduler.

4.3 Implementation of EDF in eCos

The task structure in eCos kernel has been modified to have the following additional parameters:

- WCET
- Period
- Deadline

New APIs have been implemented inside the kernel to create and run a task using EDF:

- cyg_edf_thread_create(): This function is responsible for creating thread's data structure inside the kernel and differs from the already existing cyg_thread_create in a way that it takes Period and WCET as input parameters.
- cyg_edf_thread_wait(): This function is called by the thread itself at the end of its execution. It puts the current task in sleep mode for a time equal to its period and then the scheduler is called.

We have implemented our PA-EDF scheduler on top of the Bitmap scheduler. All the calls to the native Bitmap scheduler are captured by our PA-EDF scheduler, which first modifies the priorities of individual tasks based on their respective deadlines and then calls the Bitmap scheduler.

4.4 Voltage Scaling algorithm in eCos

We first run EDF to generate the offline curve based on the inflated WCET of different tasks. This curve is stored in a 2 dimensional array (time and offline-load) inside the kernel. We lookup this array based on time relative to start time. The EDF scheduler in eCos has been modified in a way that it first calculates the online load based on the WCET of different tasks and then updates it accordingly whenever a task is completed or pre-empted.

We have also implemented a DVS thread that compares the offline and online loads to find the appropriate CPU speed settings. If the online curve is below the offline curve to begin with, it finds out the intersection point of these curves and changes the setting as required and sleeps until the intersection point is reached. Every time a task is completed or pre-empted the scheduler recalculates the online and offline loads and awakens the DVS thread. Tasks are implemented as `for` loops as in [21]. These simulated tasks provided us the flexibility to vary the actual execution time during run time, which is very difficult to obtain using real-time applications. We have implemented a function `do_dvs` (Figure 2) inside the PA-EDF scheduler which interfaces with the Voltage-Clock scaling hardware. Currently `do_dvs` supports two voltage levels but it can be easily modified to support multiple voltage levels by passing appropriate voltage level as an input to it.

```
void Cyg_Scheduler::do_dvs()
{
CYG_REPORT_FUNCTION();
unsigned char *com2;
com2= (unsigned char *)0x3100000;
//0x3100000 is mapped to RTS line of COM2 port
/*A true value of high_slope indicates 1.5V-733Mhz
   setting and a false value indicates 1.2V-400Mhz
   setting */
if(high_slope){
/*switching frequency from 733Mhz to 400Mhz*/
__asm__("MOV R1, #4");
__asm__("MCR p14, 0, R1, c6, c0, 0");
/*switching voltage from 1.5V to 1.2V*/
*(com2+4)=0x0a;
high_slope=0; }
else{
/*switching frequency from 400Mhz to 733Mhz*/
__asm__("MOV R1, #9");
__asm__("MCR p14, 0, R1, c6, c0, 0");
/*switching voltage from 1.2V to 1.5V*/
*(com2+4)=0x08;
high_slope=1; }
CYG_REPORT_RETURN();
}
```

Figure 2: Voltage-Clock scaling API.

5. RESULTS

The experimental setup for these results is as follows. The execution times of all the tasks are specified in terms of the high-voltage setting. The task periods are chosen randomly to be integers between 200 and 1200 ms. Voltage-speed settings for our experiments are summarized in Table 1.

The parameters of importance are the worst-case processor utilization, U_w, the number of tasks, and the value of a (the actual execution is uniformly distributed in the range $[a \times WCET, WCET]$). Energy consumption is expressed as a percentage of the consumption if the processor was run at high voltage during execution of the tasks.

The impact of a on the CPU energy consumption for different worst case utilizations is shown in Figure 3. As a decreases, the variance in the execution time of tasks increases. This means that the a priori information we have about the tasks' execution times decreases. For a low utilization of 0.5 or less, the entire workload can usually be run at low voltage, and so the energy consumption is 40% of the high-voltage consumption. For worst case utilization of 0.6 or more, the value of a begins to have an impact on the energy consumption. For a small value of a, execution times of tasks are more variable, which means that (for a given worst case utilization) the system has more run-time slack and can run at lower voltage for longer. As a increases and the execution times do not vary as much, the system has less run-time slack and runs at high voltage for longer, and thus the energy consumption increases. When the worst case utilization is 0.8 or 0.9, the fraction of time the system has to run at high voltage increases and the energy consumption goes up.

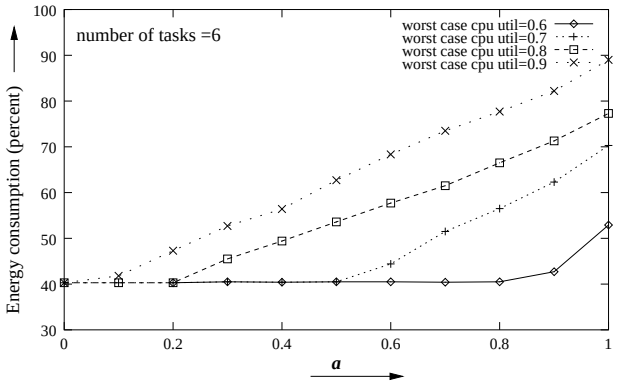

Figure 3: Impact of a on CPU energy.

The effect of the number of tasks on the system energy is due to the fact that a greater number of tasks in the system for a given load provides more opportunity to the system to effectively exploit the slack. Information about the early completion of a task is only available when the task finishes. In general, the higher the number of tasks, the greater is the advantageous impact of small values of a. This is illustrated in Figure 4 where for a small value of a, a 6-task system performs noticeably better then systems with a lower number of tasks but this gap narrows down as we move to higher values of a as shown in Figure 5.

6. CONCLUSION AND FUTURE WORK

We have implemented a dynamic voltage scaling scheduler

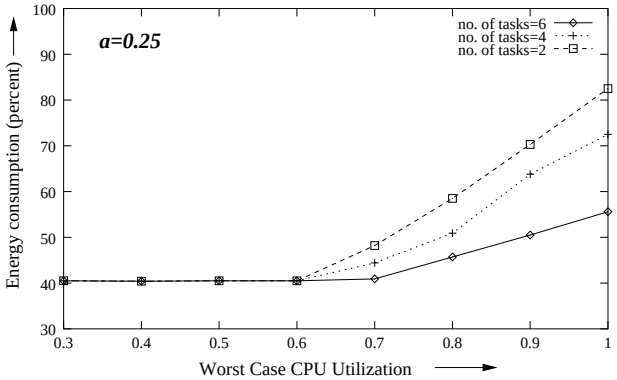

Figure 4: Impact of the number of tasks on CPU energy consumption for a=0.25.

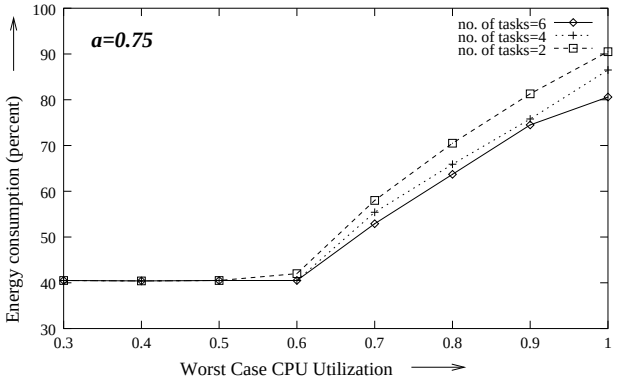

Figure 5: Impact of the number of tasks on CPU energy consumption for a=0.75.

in eCos. This scheduler provides a way to claim run-time slack in the system without missing any hard deadline and can be used as an effective solution for power constrained hard real-time system. We have modified an Intel Xscale board for DVS to validate our algorithm.

Overall, we have demonstrated the energy savings achieved by incorporating dynamic voltage scaling on a commercial microprocessor using a power aware embedded OS. Our DVS-enabled system (OS and hardware) when used for embedded applications, can lead to significant energy savings.

There are several directions in which future work can go. We have implemented a system with two voltage levels that can be modified to multiple voltage-speed settings. Other possible directions include implementing a DVS-based scheduler for IRIS (Increase Reward Increase Service) [24] tasks and tasks with precedence graphs [25].

Acknowledgements

We would like to thank Sandeep Padmanabhan from the Arizona State University for his valuable help and ideas on hardware modifications as well as technical details on the overhead of DVS.

7. REFERENCES

[1] C. M. Krishna and Y. H. Lee, "Voltage-Clock-Scaling Adaptive Scheduling Techniques for Low Power in Hard

Real Time Systems," *IEEE Real-Time Technology and Applications Symposium*, May 2000, pp. 156–165.

[2] O. S. Unsal and I. Koren, "System-Level Power-Aware Design Techniques in Real Time Systems," *IEEE* , July 2003, pp. 1–15.

[3] V. Swaminathan, C. B. Schweizer, K. Chakrabarty and A. A. Patel, "Experiences in Implementing an Energy-Driven task Scheduler in RT-Linux," *Proc. Real-Time and Embedded Technology and Applications Symposium*, 2002, pp. 229–239.

[4] P. Pillai and K. G. Shin., "Real-time dynamic voltage scaling for low power embedded operating systems," *18th ACM Symposium on Operating Systems Principles*, October 2001, pp. 89–102.

[5] Y.-H. Lee and C. M. Krishna, "Voltage-clock scaling for low energy consumption in fixed-priority real-time embedded systems," *The International Journal of Time-Critical Computing Systems. Kluwer Academic Publishers*, pp. 303-317, 2003.

[6] Y. Shin, K. Choi, and T. Sakurai, "Power optimizations of real-time embedded systems on variable speed processors," *Proc. Int. Conf. Computer-Aided Design*, pp. 365-368, June 2000.

[7] G. Kuan, and X. Hu "Energy efficient fixed-priority scheduling for real-time systems on variable speed processors," *Proc. Design Automation Conference*, pp. 828-833, June 2001.

[8] T. Ishihara, and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proc. Int. Symp. Low Power Electronics and Designs*, 1998, pp. 197–202.

[9] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics," *Euromicro Conference on Real-Time Systems*, 2001, pp. 225–232.

[10] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," *International Symposium on Low-Power Electronics and Design*, 1998, pp. 76–81.

[11] V. Raghunathan, P. Spanos and M. B. Srivastava., "Adaptive power-fidelity in energy-aware embedded systems," *Proc. Design Automation Conf.*, 2001, pp. 106–115.

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

[13] T. Ma and K. G. Shin, "A user customizable energy-adaptive combined static/dynamic scheduler for mobile applications," *Real-Time Systems Symposium*, 2000, pp. 227–236.

[14] http://www.intel.com/design/intelxscale/.

[15] Transmeta Corporation. http://www.transmeta.com/.

[16] ADI Engineering Inc. BRH reference platform. http://www.adiengineering.com/productsBRH.html

[17] C. L. Liu, and J. W. Layland, "Scheduling algorithms for multiprogramming in hard real-time environment," *J. ACM 20, 1* , Jan. 1993, pp. 46–61.

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

[19] A. Chnadrakasan, A. Sheng, and R. W. Broderson, "Low-power CMOS design," *IEEE J. Solid-State Circuits* , 1992, pp. 472–484.

[20] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Proc. USENIX Symp. Operating Systems Design and Implementation* , 1992, pp. 13–23.

[21] Cristiano Pereira, Vijay Raghunathan, Shalab Gupta, Rajesh Gupta, and Mani Srivastava, "A Software Architecture for Building Power Aware Real Time Operating Systems," *Technical Report #02-07*, March 14, 2002 .

[22] Advance Configuration and Power Interface (ACPI), "http://www.acpi.info/" .

[23] The embedded Configurable operating system (eCos), "www.redhat.com/embedded/technologies/ecos" .

[24] J. K. Dey, D. F. Towsley, C. M. Krishna, and M. Girkar, "Efficient On-Line Processor Scheduling for a Class of IRIS (Increasing Reward with Increasing Service.)," *Real-Time Tasks.SIGMETRICS*, 1993, pp. 217–228.

[25] D. Roychowdhury, I. Koren, C. M. Krishna, "A Voltage Scheduling Heuristic for Real-Time Task Graphs.)," *Proc. of the Performance and Dependability Symposium (IPDS)*, 2003, pp. 217–228.

[26] Agilent. http://www.home.agilent.com

[27] Y. Zhang, Z. Lu, J. Lach, M. Stan, K. Skadron, "Optimal Procrastinating Voltage Scheduling for Hard Real-Time Systems", *Design Automation Conference*, 2005, pp. 905-8.

[28] Y. Zhu and F. Mueller "Feedback EDF Scheduling Exploiting Hardware-Assisted Asynchronous Dynamic Voltage Scaling," *LCTES 2005*

[29] A. Anantaraman, A. El-Haj-Mahmoud, R. Venkatesan, Y. Zhu, and F. Mueller, " EDF-DVS Scheduling on the IBM Embedded PowerPC 405LP," *The first Watson Conference on Interaction between Architecture, Circuits, and Compilers (PAC2 '04)*, October 2004.