Energy Aware Wireless Systems with Adaptive Power-Fidelity Tradeoffs

Vijay Raghunathan, *Student Member, IEEE,* Cristiano L. Pereira, Mani B. Srivastava, *Senior Member, IEEE,* and Rajesh K. Gupta, *Fellow, IEEE*

***Abstract*— Wireless networked embedded systems, such as multimedia terminals, sensor nodes, etc., present a rich domain for making energy/performance/quality tradeoffs based on application needs, network conditions, etc. Energy awareness in these systems is the ability to perform tradeoffs between available battery energy and application quality requirements. In this paper, we show how operating system directed dynamic voltage scaling and dynamic power management can provide for such a capability. We propose a real-time scheduling algorithm that uses runtime feedback about application behavior to provide adaptive power-fidelity tradeoffs. We demonstrate our approach in the context of a static priority based preemptive task scheduler. Simulation results show that the proposed algorithm results in significant energy savings compared to state-of-the-art dynamic voltage scaling schemes with minimal loss in system fidelity. We have implemented our scheduling algorithm into the eCos real-time operating system running on an Intel XScale based variable voltage platform. Experimental results obtained using this platform confirm the effectiveness of our technique.**

***Index Terms*— Wireless Embedded Systems, Real Time Systems, Dynamic Power Management, Dynamic Voltage Scaling**

I. INTRODUCTION

WIRELESS embedded systems such as multimedia terminals, 3G cell phones, ad-hoc networks of wireless sensors, wireless toys and robots, etc., are increasingly computationally capable and often sport high bandwidth network connections. As a result, they run sophisticated algorithms and execute multiple large multimedia/data intensive applications. Since several of these applications have real-time constraints, a real-time operating system (RTOS) is often used in the implementation of these systems.

The battery operated nature of these systems requires them to be highly energy efficient to maximize device lifetimes. The drive to prolong system lifetime has resulted in several low power hardware design techniques (see [1], [2] for an overview of these techniques). In addition to using low power design techniques, a commonly used approach for energy aware system operation is Dynamic Power Management (DPM), in which unused system components are shutdown or sent into low power states [3]. An alternative, and more effective when applicable, technique used to increase energy efficiency is Dynamic Voltage Scaling (DVS) in which the supply voltage and clock frequency of the processor are changed dynamically to just meet the instantaneous performance requirement [4]. The RTOS is uniquely poised to efficiently implement DPM and DVS policies due to the following reasons: (i) since the RTOS coordinates the execution of the various application tasks, it has global information about the performance requirements and timing constraints of all the applications, and (ii) the RTOS can directly control the underlying hardware platform, fine-tuning it to meet specific system requirements.

In this paper, we use an RTOS directed approach to enable energy aware system operation by exploiting the following characteristics of wireless embedded systems:

- Most wireless systems are resilient to packet losses and errors. The operating scenarios of these systems invariably involve data losses and errors (e.g., due to noisy wireless channel conditions), and the application layer is designed to be tolerant to these impairments. Most wireless systems, therefore, are "soft" real-time systems where a few deadline misses only lead to a small degradation in the user-perceived application quality.
- Wireless embedded systems offer a power-fidelity tradeoff that can be tuned to suit application needs. For example, the precision of the computation (e.g., quality of the audio or video) can be traded off against the power consumed for the computation. As another example, wireless channel induced errors can be reduced by increasing the transmission power of the radio.
- Wireless embedded systems have time varying computational loads. Performance analysis studies have shown that for typical wireless applications, the instance to instance task execution time varies significantly, and is often far lower than the worst case execution time (WCET). Table I gives the WCET and best case execution time (BCET) for a few benchmarks and clearly illustrates the large workload variability [5]. Although the execution times vary significantly, they depend on data values that are obtained from physical real-world signals (e.g., audio or video streams), and are likely to have some temporal correlation between them. This enables us to predict task instance execution times with reasonable accuracy.

A. Paper contributions

The contributions of this paper are threefold:

1) We present an enhanced fixed priority schedulability analysis for variable voltage systems that incorporates the timing overheads incurred due to voltage/frequency changes and processor shutdown/wakeup. This analysis

Manuscript received October 29, 2002; revised June 7, 2003. This paper is based in part on research funded through the DARPA PAC/C Program under AFRL Contract #F30602-00- C-0154, and through Semiconductor Research Corporation System Task Thrust ID 899.001.

V. Raghunathan and M. B. Srivastava are with the Department of Electrical Engineering, University of California, Los Angeles, CA 90095 USA (e-mail: vijay@ee.ucla.edu).

C. L. Pereira and R. K. Gupta are with the Department of Computer Science and Engineering, University of California, San Diego, CA 92093 USA.

TABLE I
VARIATION IN EXECUTION TIMES (IN CLOCK CYCLES) FOR A FEW MULTIMEDIA BENCHMARKS [5]

Program	Description	BCET	WCET
DES	Data encryption	73,912	672,298
DJPEG	JPEG decompression (128x96, color)	12,703,432	122,838,368
FDCT	JPEG forward DCT	5,587	16,693

can be used to accurately analyze the schedulability of *non-concrete* periodic task sets, scheduled using the Rate Monotonic (RM) or the Deadline Monotonic (DM) priority assignment schemes.

2) We present a proactive DVS technique that exploits the above described characteristics of wireless embedded systems to achieve significant energy savings. A key feature of our technique, not addressed in prior work, is that it yields an adaptive tradeoff between energy consumption and system fidelity/quality.
3) Our DVS technique has been implemented into the kernel of the eCos [6] real-time operating system running on an Intel XScale [7] processor. We present measured results to demonstrate the effectiveness of our technique. As part of our implementation, we have also developed a complete software architecture that facilitates application-RTOS interaction for efficient DPM/DVS.

The proposed DVS technique exploits task runtime variation by predicting task instance execution times and performing DVS accordingly. Most previously proposed predictive DVS algorithms [4], [8] are processor utilization based, and hence perform poorly in the presence of latency constraints. In contrast, our technique is tightly coupled to the schedulability analysis of the underlying real-time scheduling scheme, thereby yielding better results than utilization based approaches. The few deadline misses that result from occasional mis-prediction are not catastrophic to performance due to the inherent error-tolerant nature of wireless systems. In addition, our algorithm uses an adaptive feedback mechanism to control the number of task deadline misses. This is done by monitoring recent deadline miss history, and accordingly adapting the prediction to be more aggressive/conservative. Finally, since our technique is based on the enhanced schedulability analysis mentioned above, it can be used to schedule non-concrete task sets, where the initial phase offsets of tasks are unknown. This is unlike several existing variable voltage real-time scheduling schemes, which require the initial phase offsets of the tasks to be known in order to perform hyper-period scheduling.

B. Related work

Energy awareness in wireless embedded systems builds upon advances in DPM, DVS, and real-time scheduling techniques. We briefly discuss related work in these areas.

Shutdown based DPM techniques have been proposed for several system components including processors, hard disks, and wireless network interfaces. The goal of system-level DPM techniques is to obtain an optimized power-state transition policy [9]–[12].

Early work on DVS was in the context of a workstation-like environment [13], [14] where average throughput is the metric of performance, and latency is not an issue since there are no real-time constraints. In these techniques, the task scheduler adjusts clock frequency and supply voltage in each discrete time interval based on the processor utilization in the preceding interval. Similar utilization based predictive techniques were studied in [4], [8]. Numerous other DVS techniques have been proposed for low power real-time task scheduling, both for uniprocessor [15]–[25] as well as multiprocessor [26]–[28] systems. A detailed survey of these techniques is presented in [29]. Most uniprocessor DVS techniques target systems with hard deadline constraints, and several of them assume that each task instance executes for its WCET. Even the few schemes that allow deadline misses [4], [8] involve utilization based speed setting decisions, and hence exhibit poor real-time behavior. Moreover, they only consider systems with a single application executing (i.e., a unitasking environment). Pering et al. [20] propose a thread based DVS technique for uniprocessor multitasking systems, based on execution time prediction. Similar to Pering's work, our work considers a uniprocessor, multitasking, real-time environment, and shows that permitting a few deadline misses leads to a significant increase in energy savings and improves the energy scalability of the system. Our work differs from the Pering's work in two ways, (i) while Pering's algorithm permits missed deadlines, it does not provide any control on the number of deadline misses. Through appropriate parameter selection, the proposed algorithm permits the user to control the number of deadline misses, and trade them off for increased energy savings, and (ii) Pering's algorithm is a purely online one, whose complexity scales linearly with the number of tasks present in the system. In contrast, since the proposed algorithm is based on an offline schedulability analysis, the scheduler overhead is independent of the number of tasks in the system, resulting in better scalability. This can be significant for systems with a large number of tasks since online DVS algorithms are invoked during every context switch.

The rest of this paper is organized as follows. Section II presents examples to illustrate the power management opportunities that exist during real-time task scheduling. Section III describes our system model. Section IV presents the enhanced RM schedulability analysis for variable voltage systems. Section V discusses our adaptive power-fidelity tradeoff technique. Section VI details our simulation based performance analysis. Section VII describes the implementation of our technique into eCos. Section VIII presents the conclusions.

II. BACKGROUND AND EXAMPLES

Next, we describe the basic scheduling approach adopted in our work, and present examples to illustrate the DPM/DVS opportunities that arise during real-time task scheduling.

TABLE II
TASK TIMING PARAMETERS FOR EXAMPLES 1 AND 2

Task	Time Period	WCET	Deadline
Audio decoding	60	10	60
Protocol processing	70	15	70
Video decoding	120	40	120

A. Task scheduling in RTOS

The task scheduler of an RTOS is responsible for scheduling a given set of tasks such that real-time constraints are satisfied. Schedulers differ in the type of scheduling policy they implement. A commonly used scheduling policy is Rate Monotonic (RM) scheduling [30]. This is a fixed-priority based preemptive scheduling scheme where tasks are assigned priorities in the inverse ratio of their time periods. Fixed priority based preemptive scheduling is commonly used in operating systems such as eCos, WinCE, VxWorks, QNX, uC/OS, etc., that run on wireless embedded devices such as handheld multimedia nodes. An alternative scheduling scheme is Earliest Deadline First (EDF) scheduling [30], where task priorities are assigned dynamically such that task instances with closer deadlines are given higher priorities. EDF can schedule task sets with a higher processor utilization than can be scheduled by a static priority based scheduling scheme such as RM. However, dynamic priority based scheduling is also more complex to implement since task priorities keep changing during runtime. No matter which scheduling policy is used, the RTOS ensures that at any point of time, the currently active task is the one with the highest priority among the ready to run tasks. The problem of task scheduling on a uniprocessor system in the presence of deadlines is known to be solvable in polynomial time if the system is preemptive in nature [31]. However, the low energy scheduling problem was shown to be NP-Complete in [32], by a reduction from the "*Sequencing with deadlines and set-up times*" problem, described in [31].

B. Power management opportunities during task scheduling

1) Static slack: It has been observed in many systems that, during runtime, even if all task instances run for their WCET, the processor utilization is often far lower than 100%, resulting in idle intervals. This slack that inherently exists in the system due to low processor utilization is henceforth referred to as *static slack*. It can be exploited to reduce energy consumption by statically slowing down the processor and operating at a lower voltage. While processor slowdown improves utilization, excessive slowdown may lead to deadline violations. Hence, the extent of slowdown is limited by the schedulability of the task set at the reduced speed, under the scheduling policy used. The following example illustrates the use of static slowdown to reduce energy consumption.

Example 1: Consider a simple mobile multimedia terminal shown in Fig. 1[1]. The system receives real-time audio and video streams over a wireless link, and plays them out. Thus, the three main tasks that run on a processor embedded in this system are protocol processing, audio decoding, and video decoding. The timing parameters for these tasks are listed in Table II. For simplicity, the initial phase offsets for the tasks are set to zero (however, our algorithm, presented in Section V, makes no such assumptions). The resulting schedule for the time interval [0, 120] when this task set is scheduled on a single processor using the RM priority assignment scheme, is shown in Fig. 2(a). It can be seen from the figure that the system is idle during time interval [90, 120]. This slack can be utilized to lower the operating frequency and supply voltage, thereby reducing the energy consumption. Fig. 2(b) shows the schedule for the same task set with the processor slowed down[2] by a factor of $\frac{4}{3}$. As seen from the figure, processor slowdown leads to a reduction in slack[3]. Note that any further reduction in processor speed will result in the video decoding task missing its deadline. When the supply voltage is also scaled, this leads to a decrease in power consumption from 420mW to 184mW, using the power vs. frequency curve for the StrongARM processor, shown in Fig. 4. The energy consumption over the interval [0, 120], therefore, decreases by 41%, compared to a shutdown based policy.

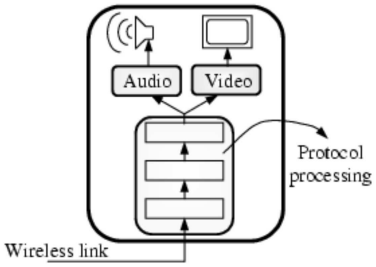


Fig. 1. Mobile multimedia terminal used in Examples 1 and 2.

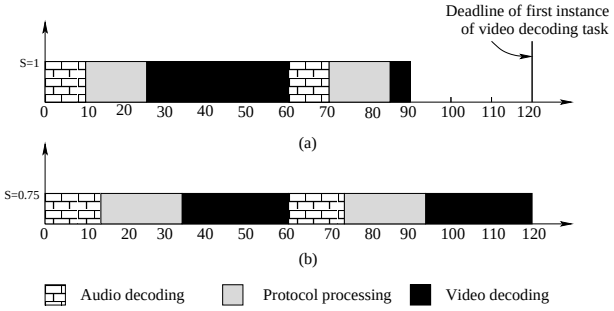


Fig. 2. Task execution schedule for Example 1: (a) Original (b) Statically optimized.

2) Dynamic slack: Static slack is not the only kind of slack present in the system. During runtime, due to the variation in the task instance execution times, there is additional slack created when a task instance finishes executing before its WCET. This slack that arises due to execution time variation is henceforth referred to as *dynamic slack*. Dynamic slack can be exploited for DVS by dynamically varying the operating

[1]The system shown in Fig. 1 could, in general, be any multitasking, uniprocessor, battery powered system, such as a 3G cell phone or a PDA.

[2]For simplicity, we assume that such an exact slowdown is possible. Our algorithm handles the case only a finite set of frequencies are available.

[3]While in this example, uniform slowdown of all tasks resulted in a complete elimination of static slack, in general different tasks might require different slowdown factors. Our algorithm does this, if necessary.

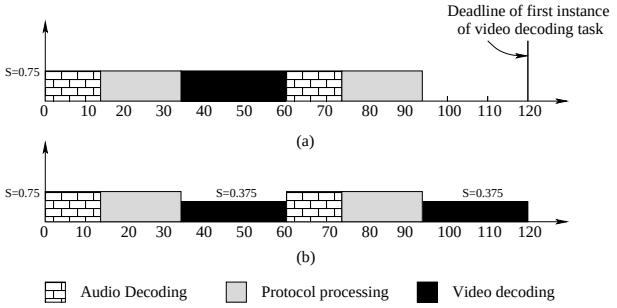


Fig. 3. Task execution schedule for Example 2: (a) Statically optimized (b) After dynamic slowdown.

frequency and supply voltage of the processor to extend the task instance's execution time to its WCET. Thus, the lower a task instance's actual execution time, the more its energy reduction potential. However, to realize this potential, we need to know/estimate the execution time of each task instance. The following example illustrates how dynamic slack can be exploited for DVS.

Example 2: Consider the statically optimized schedule for the task set of Example 1 shown in Fig. 2(b). Fig. 3(a) shows the resulting schedule when the video decoding task instance requires only 20 time units to complete execution at maximum processor speed. As a result, the video decoding task now completes execution at time $t = 60$ units, and does not get preempted. As seen from the figure, this execution time variation creates some dynamic slack. To utilize this slack, the processor frequency and supply voltage are dynamically reduced further during the execution of the video decoding task. If the frequency were to be dropped by a factor of 2 over the already statically optimized value, the slack gets filled up again as shown in Fig. 3(b)[4]. Accompanied by appropriate voltage scaling, the energy consumption for the interval [0, 120] now reduces by a further 14% compared to the statically optimized schedule of Fig. 3(a).

The above examples illustrated the energy reduction opportunities present during real-time task scheduling. We next describe our system model, and present an enhanced RM schedulability analysis for variable voltage systems. Using this analysis, we then present our power aware scheduling algorithm that exploits the above illustrated DVS/DPM opportunities to yield large energy savings.

III. SYSTEM MODEL

A. Task timing model

A set of N independent periodic tasks is to be scheduled on a uniprocessor system. Associated with each task *i* are the following parameters: (i) T_i is its time period, (ii) C_i is its worst case execution time (WCET), (iii) B_i is the best case execution time (BCET), and (iv) D_i is its deadline. The BCET and WCET can be obtained through execution profiling or other performance analysis techniques [5]. Our techniques are applicable to the general case where D_i and T_i are unrelated. We discuss this further in Section IV.

[4]Note that in Fig. 3(b), the video decoding task is preempted at time $t = 60$ units, and resumes execution later to complete at time $t = 120$ units.

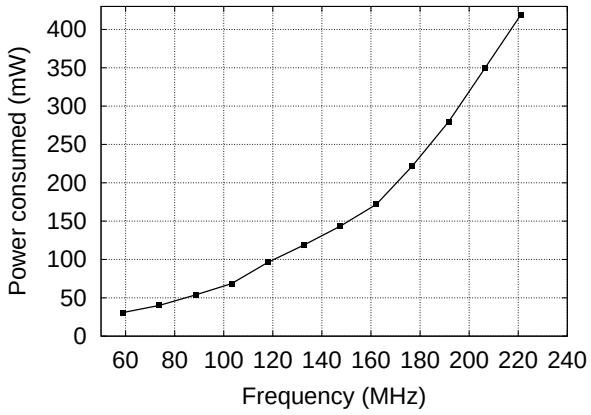


Fig. 4. Power consumption as a function of clock frequency for the StrongARM processor.

B. Power model

We use a power model of the StrongARM processor, which is based on actual measurements, for computing the power consumption. Fig. 4 shows the power consumption of the StrongARM, plotted as a function of the operating frequency. This plot is obtained from actual current and voltage measurements reported in [33] for the StrongARM SA-1100 processor. Using this curve, the power consumption of the processor for a given clock frequency can be calculated. By varying the supply voltage and clock frequency, the variable voltage system can be made to operate at different points along this curve. For a given clock frequency (i.e., speed setting), the energy consumption of a task instance can be computed as the product of the power consumption (obtained from the power-frequency curve shown above) and the execution time of the task instance. Further, the energy-speed curve is convex in nature [17]. Thus, if two task instances have to be completed by a deadline, due to Jensen's inequality $\left(\overline{E(r)} \geq E(\overline{r})\right)$ [34], it is more energy efficient to run the two tasks at a constant speed than to change the speed from one task to the other. As will be seen in the next section, our algorithm utilizes this fact by attempting to slow down tasks in a uniform manner, to the extent possible.

C. Overheads due to DVS/DPM

The variable supply voltage is generated using a DC-DC converter. During a voltage transition, it takes a finite amount of time for the output voltage of the DC-DC converter to transition from the present level to the desired level. Efficient DC-DC regulators with fast transition times were reported in [35], [36]. Complementary to the supply voltage variation is the accompanying variation of the clock frequency. The phase locked loop present in the clock generation circuitry also takes a finite amount of time to settle at its steady state value, after a frequency change. Further, shutting down and waking up the processor involve a non-zero time and power overhead. These overheads have to be explicitly accounted for during task scheduling. For variable voltage real-time systems, the timing overheads have to be incorporated into the task-set

schedulability analysis to guarantee the schedulability of tasks. The energy overheads have to be considered while computing the energy savings obtained through the use of DVS and DPM. Commercial processors till recently had large frequency transition overheads (for example, the StrongARM [37] takes 150 μseconds to change its frequency). However, these transition times are considerably lower in more recent processors as designers realize the effectiveness of DVS. The transition overhead is around 30 μseconds in the XScale processor [7]. We use a stall duration of 150 μseconds in our simulations, since the power model used is that of a StrongARM processor. However, in our implementation (Section VII), the stall duration is only 30μseconds, since our experimental testbed is based on the XScale processor. Finally, although some processors [20] developed at academic institutions allow the computation to continue while the voltage and frequency are being changed, commercially available processors such as the XScale do not permit this. Therefore, in our simulations, the processor is stalled for the duration of the frequency change, resulting in some wasted energy (although this is insignificant since the processor clock is frozen). We account for all the above mentioned overheads in our energy calculations.

IV. RM SCHEDULABILITY ANALYSIS

Several schedulability analysis results exist for RM scheduling [38]. However, none of them consider variable voltage/frequency processors. Since processor shutdown, processor wakeup, and voltage and frequency changes take a finite amount of time, they affect the timing behavior of the system. In this section, we present an enhanced schedulability analysis for RM scheduling, which takes into account the overheads introduced due to DPM/DVS. We denote the time taken to shutdown/wakeup the processor by T_S, and the time taken for a voltage/frequency change by T_V[5].

A. The $D_i \leq T_i$ case

The schedulability analysis for RM scheduling for the case $D_i \leq T_i$ is presented in [38], in which necessary and sufficient conditions are derived for the schedulability of a non-concrete (i.e., unknown initial phases for the tasks) periodic task set. The *response time* of a task instance is defined as the amount of time needed for the task instance to finish execution, from the instant at which it arrived in the system. The worst case response time (WCRT), as the name indicates, is the maximum possible response time that a task instance can have. For a conventional fixed speed system, the WCRT of a task under the RM scheduling scheme is given by the smallest value of R_i that satisfies the equation [38]:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \tag{1}$$

where $hp(i)$ denotes the set of tasks with priority greater than that of task i, C_i denotes the worst case execution time of task i, $\lceil \cdot \rceil$ denotes the ceiling operator, and R_i is simply an intermediate variable used in computing the WCRT. Equation (1) can be solved using an iterative technique. The summation term on the right-hand-side of the equation represents the total interference that an instance of task i sees from higher priority tasks during its execution. Task i is schedulable iff the WCRT of the task is less than or equal to its deadline D_i. Our enhanced schedulability analysis is based upon three observations.

1) Slowing down a task by a factor α increases its computation time by the same factor. Therefore, given a set of slowdown factors $\alpha_i,\ i \in \{1, \ldots,\ N\}$, the computation time for task i now becomes $(\alpha_i \cdot C_i)$.
2) Each higher priority task instance that arrives during the lifetime of a lower priority task instance adds either T_V or $2 \cdot T_V$ to the response time of the lower-priority task instance, where T_V is the time taken to change the processor frequency/voltage. This is the preemption related overhead of DVS. It takes T_V time units to change the voltage to execute the higher priority task instance, and another T_V units to change back to resume execution of the preempted task instance, and the worst case for a task instance occurs when this procedure repeats for every higher priority task instance that arrives during its lifetime. So, the worst case interference that a higher priority task instance contributes is equal to $2 \cdot T_V$. However, when the higher priority task is being executed, suppose a task with priority in between that of the currently running task and the preempted task arrives. It is easy to see that this task instance only causes an interference of T_V.
3) The maximum amount of blocking that can be faced by a task instance, and which is not preemption related is $Max.\ \{2 \cdot T_{shut} + T_V,\ 2 \cdot T_V\}$, where T_{shut} is the time taken to shutdown/wakeup the processor. Since the processes of voltage/frequency change or shutdown cannot be preempted, each task instance may be blocked for an amount of time if it arrives just after a voltage/frequency change or a shutdown has been initiated.

Based on the above observations, the following theorem can be used as a sufficient condition for the schedulability of a task set under RM scheduling on a DVS enabled system.

Theorem 1: A non-concrete periodic task set is guaranteed to be schedulable on a variable voltage processor, under the RM scheduling policy if, for every task i, $WCRT(i) \leq D_i$ where $WCRT(i)$ is the smallest value of R_i that satisfies the equation:

$$R_i = \alpha_i \cdot C_i + Max.\{2 \cdot T_S + T_V, 2 \cdot T_V\} + \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times (\alpha_j \cdot C_j + 2 \cdot T_V) \right) \tag{2}$$

where α_n is the slowdown factor for task n.

B. The $D_i > T_i$ case

The schedulability analysis for the case when $D_i > T_i$ is more complex. For fixed speed systems, when D_i and T_i are

[5]The time taken for a voltage change could be different from the time taken for a frequency change. In that case, T_V is the maximum of the two values. Further, the voltage/frequency change will be accompanied by a context switch, whose overhead can also be added to T_V.

unrelated, the WCRT of task i is given by [38]:

$$WCRT(i) = Max._{\,0 \le q \le Q}\ \{W_{i,q} - q \cdot T_i\} \qquad (3)$$

where Q is the smallest non-negative integer value that satisfies $W_{i,Q} \le (Q+1) \cdot T_i$, and $W_{i,q}$ is the smallest solution to the equation:

$$W_{i,q} = (q+1) \cdot C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_{i,q}}{T_j} \right\rceil \times C_j \qquad (4)$$

The task set is schedulable if, and only if, $WCRT(i) \le D_i,\ \forall i \in \{1, \ldots, n\}$. Intuitively, the above analysis implies that the WCRT may not occur for the first instance of the task, and we may need to check the schedulability for multiple instances of the task. In the case of variable voltage systems, using the three observations listed in Section IV.A., the schedulability test reduces to a sufficient condition, and Equation (4) is changed to:

$$W_{i,q} = (q+1) \cdot \alpha_i \cdot C_i + Max.\{2 \cdot T_S + T_V, 2 \cdot T_V\} + \sum_{j \in hp(i)} \left(\left\lceil \frac{W_{i,q}}{T_j} \right\rceil \times (\alpha_j \cdot C_j + 2 \cdot T_V) \right) \qquad (5)$$

where α_i is the slowdown factor for task i. Note that the $D_i \le T_i$ case is just a special case of this enhanced equation. When $D_i \le T_i$, Equations (3) and (4) are satisfied for $Q = 0$. The offline component of our algorithm uses this enhanced sufficient schedulability test to compute static slowdown factors for each task.

V. ALGORITHM

We consider a priority based, preemptive scheduling model where the priority assignment is done statically, using the RM priority assignment policy[6]. Task instances that do not finish executing by their deadline are killed. Our algorithm adjusts the processor's supply voltage and clock frequency whenever a new task instance first starts executing, as well as every time it resumes execution after being preempted by a higher priority task. Also, our algorithm uses a constant speed setting between two points of preemption in order to maximally exploit the convexity of the energy-speed curve. The pseudo-code of our algorithm is shown in Fig. 5 and 6. The crucial steps of our algorithm are described next.

A. Static slowdown factor computation

This component of our algorithm involves a schedulability analysis of the task set. It is executed whenever a new task (e.g., audio/video decoding) enters the system and registers itself with the scheduler. Given the task set to be scheduled, the procedure COMPUTE_STATIC_SLOWDOWN_FACTORS performs a schedulability analysis (as described in Section IV), and computes the minimum operating frequency for each task, at which the entire task set is still schedulable. Every task is slowed down uniformly till one or more tasks reach their critical point (i.e., they are "just" schedulable). This is done by the function Scale_WCET(). At this juncture, further slowdown of any task with higher priority than any of the critical tasks will render at least one critical task unschedulable. Therefore, only the tasks, if any, with lower priority than any of the critical tasks can be slowed down further without affecting the schedulability of the task set. The procedure continues till there is no further scaling possible while satisfying all task deadlines. In summary, this iterative procedure gives us a new reduced frequency for each task such that the entire task set is just schedulable. The online component of our algorithm augments this static slowdown with a dynamic slowdown factor, computed at runtime, to increase energy savings.

[6]Although not analyzed in this paper, our algorithm is also applicable to dynamic priority scheduling, such as Earliest Deadline First scheduling, using an appropriately modified schedulability analysis.

B. Dynamic slowdown factor computation

The online component of our algorithm invokes the COMPUTE_DYNAMIC_SLOWDOWN_FACTOR procedure, listed in Fig. 6, to dynamically alter the supply voltage and operating frequency of the system in accordance with recent task execution statistics. Thus, while the offline component of our algorithm computes *task* specific static slowdown factors, the online component augments this with *task instance* specific dynamic slowdown factors. The COMPUTE_DYNAMIC_SLOWDOWN_FACTOR procedure is called each time a new task instance starts, or resumes execution after being preempted by a high priority task. The algorithm sets the processor speed and voltage based on an estimate of the task instance execution time.

1) Runtime prediction strategy: A novel feature of our algorithm is that it involves a *proactive* DVS scheme. In contrast, a majority of existing DVS techniques are reactive in nature. Therefore, in conventional schemes, any slack that arises due to a task instance completing execution early is distributed to task instances that follow, using (possibly complex) dynamic slack reclamation algorithms. Our technique, on the other hand, attempts to avoid the creation of slack altogether through the use of a predictive technique, eliminating the need for slack reclamation. In our scheme, the execution time of a task instance is predicted as some function of the execution times of a fixed number of previous task instances. An execution history database is maintained for each task, that is updated every time a task instance finishes executing or is killed due to a deadline miss. The function used to compute the execution time determines the type of predictor. We have evaluated our techniques using a simple average predictor model, as well as an exponentially weighted average one. Both these are simple prediction schemes that place a light computational load on the scheduler. Since the results were very similar in the two cases, we only report the results obtained using the weighted average model. The use of a more complex predictor will improve prediction accuracy. However, this will increase the computational burden on the scheduler since the predictor is executed every time a task instance starts or restarts after preemption.

2) Improving prediction accuracy: In order to improve prediction accuracy while retaining simplicity, we extend the prediction strategy to use *conditional prediction* as described below. Every time a task instance is preempted, the predicted

```
Procedure COMPUTE_STATIC_SLOWDOWN_FACTORS
Inputs: τ, Time_Periods[ ], WCETs[ ], Deadlines[ ]
Outputs: Static_Slowdown_Factors[ ]
{
  SET S = τ                  //Tasks that can be slowed down further
  SET S1 = φ                 //Tasks that will miss deadline upon further scaling
  Current_Scaling_Factor = 1;
  For each task i in S
    Static_Slowdown_Factor[i] = 1;
  While (S != φ) {
    f = Scale_WCET (Time_Periods[ ], WCETs[ ], Deadlines[ ], S, Static_Slowdown_Factor[ ]);
    S1 = Tasks that will miss deadlines with further scaling;
    Current_Scaling_Factor *= f;
    For each task i in S
      Static_Slowdown_Factor[i] = Current_Scaling_Factor;
    S = All tasks with priority less than the lowest priority task in S1;
  }
}
```

Fig. 5. Pseudo-code for the offline component of the proposed algorithm.

remaining time for that task instance is recalculated as the expected value of the execution history distribution from the already elapsed time to the WCET of the task. This gives the predicted remaining time of the task, given that it has already run for the elapsed time. This improves the prediction accuracy, and reduces the probability of under-prediction, in turn reducing the probability of missed deadlines. As a side effect, it also reduces the impact of the original prediction model. This explains why we obtained similar results using the simple average and weighted average predictors.

3) Adaptive power-fidelity tradeoff: Some applications may not be able to tolerate the number of deadlines that are missed using the above described predictive scheme. Therefore, we introduce an adaptive feedback mechanism into the prediction process. The predicted execution time of a task instance is altered using an adaptive multiplicative factor, as shown in the pseudo-code of Fig. 6. Deadline miss history is monitored using a moving window mechanism, and if the number of deadline misses is found to be increasing, the prediction is made more conservative, reducing the probability of further deadline misses. Similarly, a low/decreasing deadline miss history results in more aggressive prediction in order to increase energy savings. This is implemented as follows. If there are more than $T1$ deadline misses in the last $WINDOW$ task instances, the algorithm becomes more conservative and increases the adaptive factor by I. Similarly, if the number of deadline misses in the last $WINDOW$ task instances is less than $T2$, then the algorithm becomes more aggressive and decreases the adaptive factor by D. A lower bound, $ADAPTIVE\_LB$, dictates the maximum aggressiveness of the algorithm. The prediction scheme is now adaptive to a recent history of missed deadlines, becoming more conservative in response to deadline misses and becoming more aggressive if no deadline misses occur over the past few instances. This adaptive control mechanism ensures that the number of deadline misses (which is representative of system fidelity) is kept under tight check at all times. The choice of the various parameters depends on how aggressive/conservative the user wants to be in the energy-fidelity tradeoff, a detailed analysis of which is beyond the scope of this paper. Note that in order to keep the algorithm and implementation simple, we have made the deadline miss history and all the parameters of the adaptive algorithm global parameters. An alternative, although more complicated, approach would be to perform the adaptation on a per-task basis. Such an approach would permit better control on the fidelity of individual applications. Finally, note that this adaptive scheme is also useful in the case when applications can tolerate more deadlines than missed through the use of the baseline predictive scheme. In such a situation, the adaptive scheme can be used to obtain an increase in energy savings at the cost of more deadline misses. Thus, the application can adjust the operating point along an energy-fidelity curve, which enhances its energy scalability. For example, as the system runs out of energy, it can scale down its fidelity gracefully. To the best of our knowledge, existing DVS schemes do not provide this capability.

4) Voltage variation policy: Our algorithm recomputes the voltage setting every time a task instance first starts executing, or restarts after being preempted. The pre-computed static slowdown factor for the task is augmented with a dynamic slowdown factor for the specific task instance that is about to start executing. The dynamic slowdown factor is computed by stretching the task instance's predicted execution time to reach its WCET. The product of the static and dynamic factors is the final slowdown factor. The processor's operating frequency is then decreased by this factor, the corresponding supply voltage is set, and execution of the task instance begins. This dynamic slowdown spreads the execution to fill otherwise idle time intervals, enabling operation at a lower supply voltage and clock frequency.

```
Procedure COMPUTE_DYNAMIC_SLOWDOWN_FACTOR
Inputs: WCET, Deadline_Miss_History, Execution_History
Output: Dynamic_Slowdown_Factor
{
  P = PREDICT_TIME (Execution_History);
  α = ADAPTIVE_FACTOR (Deadline_Miss_History, α);
  P = P * α;
  Dynamic_Slowdown_Factor = WCET / P;
}

Procedure PREDICT_TIME
Input: Execution_History
Output: Predicted_Time
{
  If (Elapsed_Time == 0)
    Predicted_Time = (Σ_{i=1}^{N} (a_i × Execution time of ith previous task instance)) / (Σ_{i=1}^{N} a_i);
    //N is the number of past instances being monitored
    //The coefficients a_i depend on the prediction scheme used
  Else
    Predicted_Time = Expected value of execution time distribution
                     from Elapsed_Time to WCET;
}

Procedure ADAPTIVE_FACTOR
Input: Deadline_Miss_History, Adaptive_Factor
Output: Adaptive_Factor
{
  x = Number of deadline misses in past WINDOW instances of the task;
  If (x ≥ T1) Adaptive_Factor += I;
  If (x ≤ T2) Adaptive_Factor -= D;
  If (Adaptive_Factor ≤ ADAPTIVE_LB) Adaptive_Factor = ADAPTIVE_LB;
}
```

Fig. 6. Pseudo-code for the online component of the proposed algorithm.

VI. SIMULATION BASED PERFORMANCE ANALYSIS

We next describe our simulation framework, and present simulation results comparing the proposed adaptive power-fidelity DVS algorithm to several existing DVS/DPM schemes to demonstrate its effectiveness.

A. Simulation model

To analyze the performance of the proposed adaptive power-fidelity DVS scheme, a discrete event simulator was built using PARSEC [39], a C based parallel simulation language. The simulation structure, shown in Fig. 7, consisted of two parallel communicating entities. The first one represented the RTOS, and implemented the task scheduler (enhanced with the DVS scheme). In addition to performing task scheduling, the RTOS entity also maintained the statistics of task instance execution times and deadline misses. The second entity, i.e., the task generator, periodically generated task instances with run times according to a trace or a distribution, and passed them to the RTOS entity.

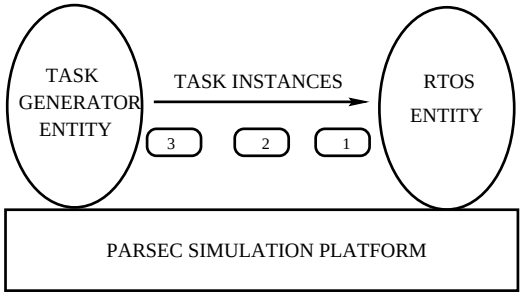


Fig. 7. System simulation model in PARSEC.

B. Simulation results

We performed simulations on two task sets (henceforth called *Task Set 1* and *Task Set 2*) that are based on standard task sets used in embedded real-time applications [40], [41]. The task instance execution times were generated using a Gaussian distribution[7] with $Mean = \frac{BCET+WCET}{2}$, and

[7]We obtained similar results using a uniform distribution. Therefore, only the results for the Gaussian distribution case are presented.

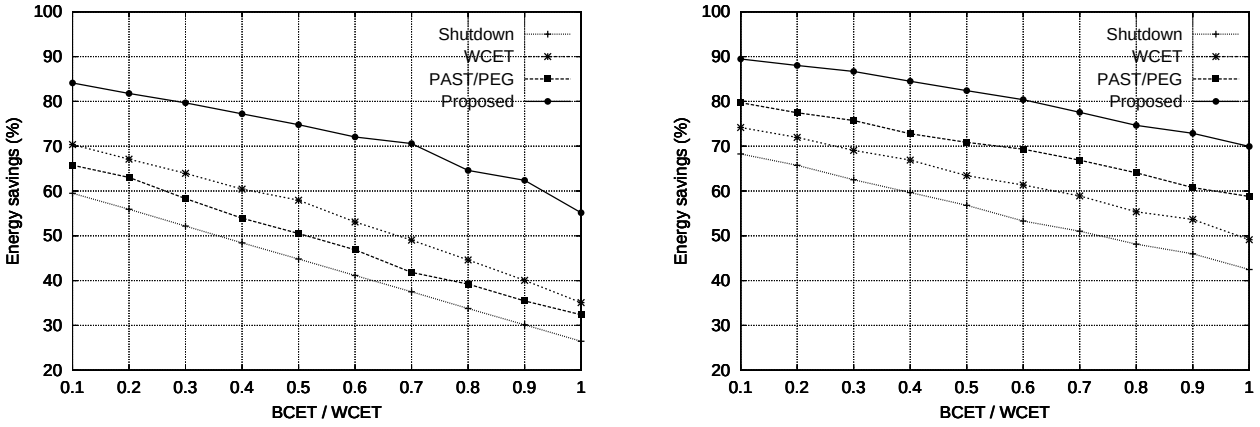


Fig. 8. Energy savings of various DVS/DPM schemes using RM scheduling for (a) Task Set 1 and (b) Task Set 2.

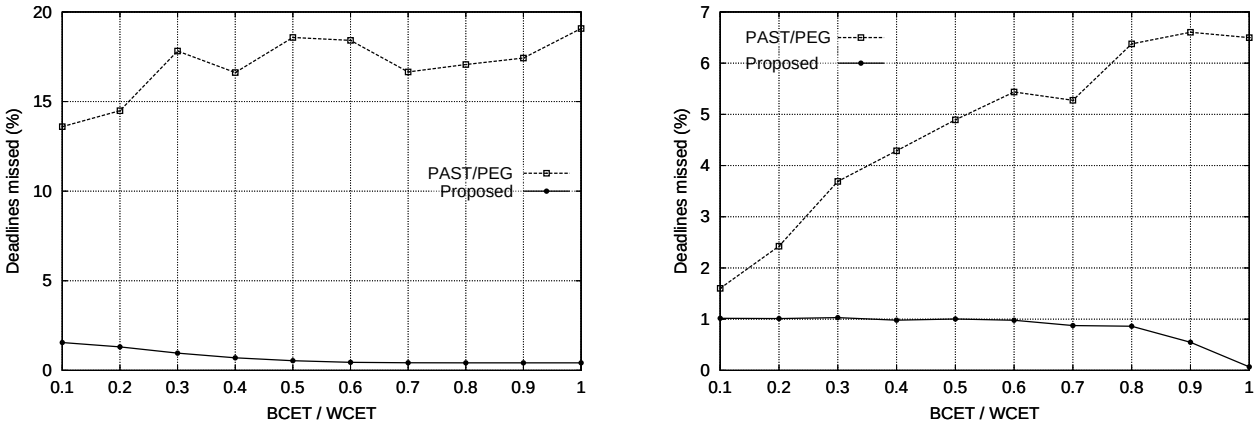
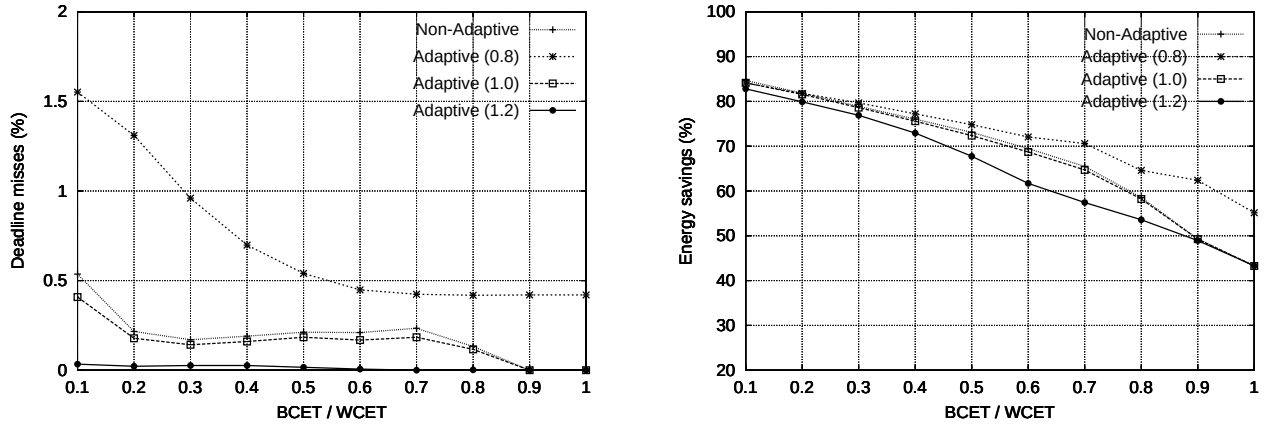


Fig. 9. Deadline miss percentage for the proposed scheme and the PAST/PEG scheme using RM scheduling for (a) Task Set 1, and (b) Task Set 2.

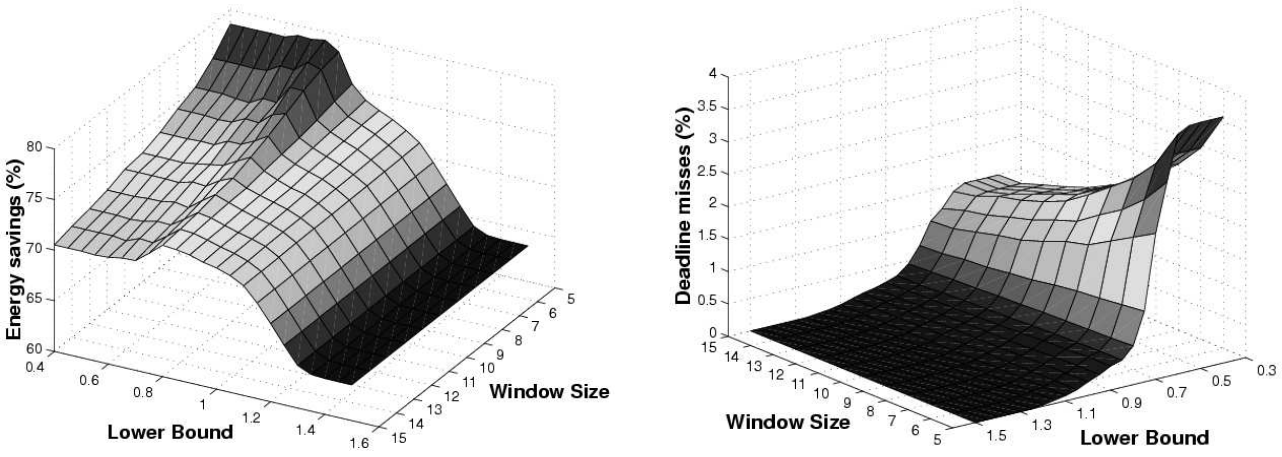
$Standard\ Deviation = \frac{WCET-BCET}{6}$. To demonstrate that our technique works even for non-concrete task sets, every task i was given a random initial phase offset in the interval $[0, T_i)$. We performed four experiments on each task set. In the first experiment, only shutdown based DPM was employed, and the supply voltage and frequency were fixed at their maximum values. Next, we implemented the low-power scheduling technique of [18]. This is a WCET based scheme, where DVS is done only when there is a single task remaining to execute. In the third experiment, we implemented the PAST/PEG DVS scheme [4], [8]. Similar to our algorithm, this is a predictive DVS scheme. However, all DVS decisions in the PAST/PEG scheme are purely utilization based. Finally, the proposed adaptive power-fidelity DVS scheme was implemented. The experiments were repeated while varying the BCET from 10% to 100% of the WCET in steps of 10%. For the adaptive scheme, the following parameter values were used: $N = 10$, $WINDOW = 10$, $I = 0.05$, $D = 0.01$, $T1 = 1$, and $T2 = 0$. The various parameters are described in Section V.B.3.

Fig. 8 shows the energy savings obtained for the two task sets for each of the above mentioned DVS/DPM schemes. The results are normalized to the case when no DVS/DPM is used. As can be seen in the figure, the proposed adaptive power-fidelity DVS algorithm results in significantly higher energy savings compared to the shutdown, WCET based, and PAST/PEG algorithms. Fig. 9 shows the percentage of deadlines missed by the proposed algorithm, and the PAST/PEG algorithm. The PAST/PEG scheme results in a large number of deadline misses because it takes DVS decisions purely based on processor utilization. As mentioned before, in real-time multi-tasking systems, the schedulability of the task set is only weakly related to the processor utilization. Therefore, the PAST/PEG algorithm results in a significant loss in real-time behavior. The algorithm proposed in this work is tightly coupled to the schedulability analysis of the underlying real-time scheduling scheme used, resulting in far fewer deadline



(a) (b)

Fig. 10. Impact of the parameter $ADAPTIVE\_LB$ on (a) Percentage of deadline misses and (b) Energy savings, for Task Set 1.



(a) (b)

Fig. 11. Variation of (a) Energy savings and (b) Deadline miss percentage with parameters $WINDOW$ and $ADAPTIVE\_LB$ for Task Set 1.

misses, as is evident from Fig. 9.

We performed additional experiments with Task Set 1 to illustrate the adaptive power-fidelity tradeoff that our scheme provides. We ran our algorithm for three different values of the parameter $ADAPTIVE\_LB$. Fig. 10(a) and 10(b) show the energy savings and deadline miss percentages, respectively, for $ADAPTIVE\_LB \in \{0.8, 1.0, 1.2\}$, and for a non-adaptive version of our algorithm. From the figures, it is clear that by increasing $ADAPTIVE\_LB$, one can trade-off energy savings for fewer deadline misses. Such an adaptive scheme enhances the system's energy-scalability. For example, as the battery drains out, $ADAPTIVE\_LB$ can be decreased, thereby gradually degrading the system's fidelity. Fig. 11(a) and 11(b) plot the energy savings and deadline misses as a function of the parameters $ADAPTIVE\_LB$ and $WINDOW$, for a $\frac{BCET}{WCET} = 0.5$. It is evident that as $ADAPTIVE\_LB$ increases, the energy savings decrease, and the deadline misses increase. As the window size increases, the energy savings decrease and the deadline misses decrease. Also, note that choosing a very low value for $ADAPTIVE\_LB$ is not very energy efficient. This is because, the large number of missed deadlines cause the voltage to be increased very frequently in response, lowering the energy savings obtained. In order to clearly show the energy-fidelity tradeoff, we have plotted the energy savings vs. the deadline miss percentage in Fig. 12 for $WINDOW = 10$ and different values of $ADAPTIVE\_LB$. As can be seen in Fig. 12, a value of 0.8 for $ADAPTIVE\_LB$ seems to yield the maximum energy savings, for this particular task set.

VII. IMPLEMENTATION

In addition to validating the proposed adaptive power-fidelity technique through simulations, we also evaluated its performance by implementing it in an RTOS running on a variable voltage hardware platform. In this section, we first

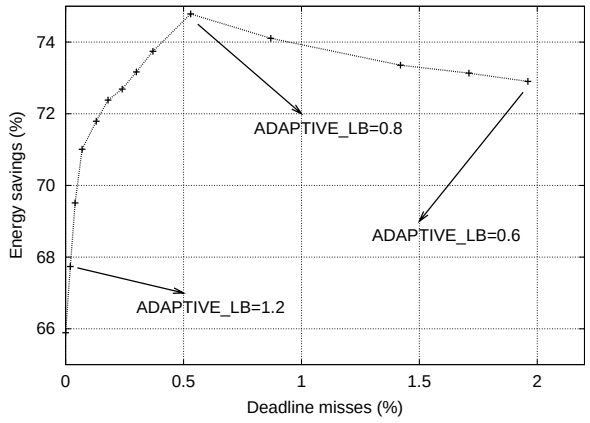


Fig. 12. Energy savings vs. Deadline misses for $WINDOW = 10$ and various values of $ADAPTIVE\_LB$ for Task Set 1.

give a brief overview of a structured software architecture that we have developed to facilitate application-RTOS interaction for effective energy management. Then, we describe our implementation in detail, and present measured results that demonstrate the effectiveness of our DVS algorithms.

A. Software architecture

We view the notion of power awareness in the application and OS as a capability that enables a continuous dialog between the application, the OS, and the underlying hardware. This dialog establishes the functionality and performance expectations (or even contracts, as in the real-time sense) within the available energy constraints. A well structured software architecture is necessary for realizing this notion of power awareness. The power aware software architecture (PASA) [42], [43] that we have developed is composed of two software layers and the RTOS kernel. One layer is an API that interfaces applications with the OS, and the second layer makes power related hardware knobs available to the OS. Both layers interface to various OS services as shown in Fig. 13(a). The API layer is separated into two sub-layers. The Power Aware Application Programmer Interface (PA-API) sub-layer provides power management functions to the applications, while the other sub-layer, the Power Aware Operating System Layer (PA-OSL) provides access to existing and modified OS services. Active entities that are not implemented within the RTOS kernel should be implemented at this layer (e.g., threads created to assist the OS in DVS/DPM, such as a thread responsible for killing other threads whose deadlines were missed). The modified RTOS and the underlying hardware are interfaced using a Power Aware Hardware Abstraction Layer (PA-HAL). The PA-HAL gives the OS access to the power related hardware knobs while abstracting out the details of the underlying hardware platform.

B. Experimental setup

Using the PASA presented above, the proposed adaptive power-fidelity DVS algorithm has been incorporated into the eCos operating system, an open source RTOS from Red-Hat Inc. eCos was ported to an 80200 Intel Evaluation Board [44], which is an evaluation platform based on the XScale processor. The processor supports nine frequency levels ranging from 200MHz to 733MHz. However, two of them (200MHz and 266MHz) cannot be used in the 80200 board due to limitations in the clock generation circuitry [7]. In addition, the processor supports three different low power modes: IDLE, DROWSY, and SLEEP. The SLEEP mode results in maximal power savings, but requires a processor reset in order to return to the ACTIVE mode. The IDLE mode, on the other hand, offers the least power savings but only requires a simple external interrupt to wake the processor up. We use the IDLE mode in our experiments due to its simple implementation.

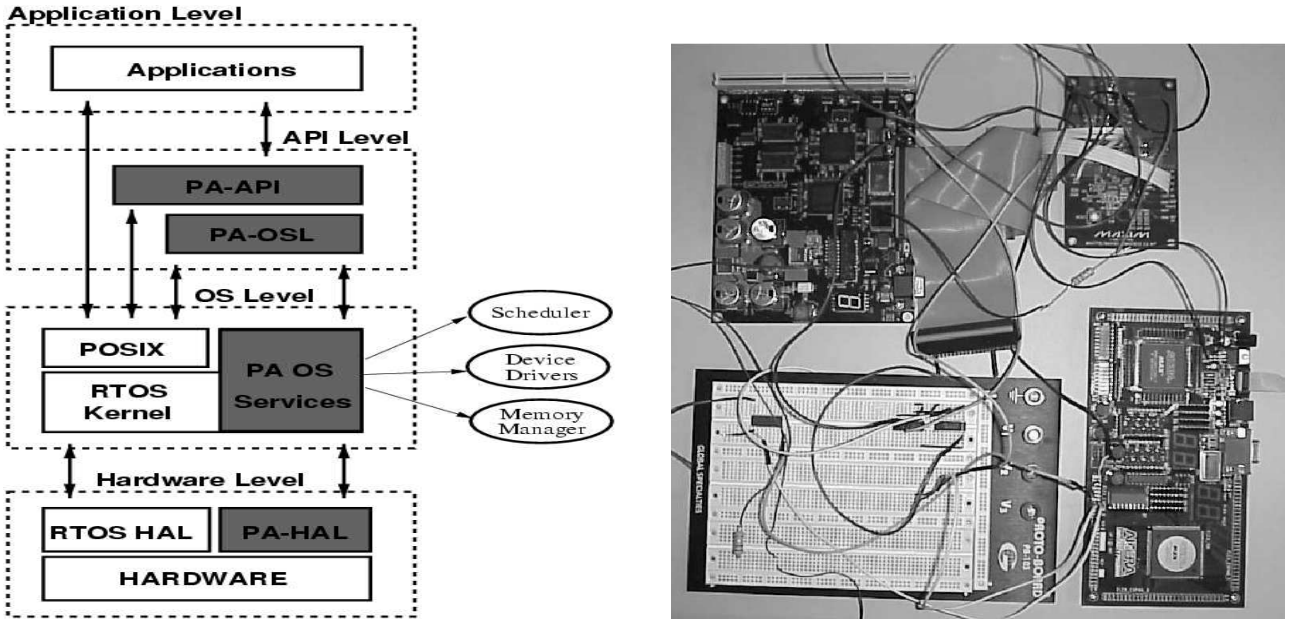
TABLE III
FREQUENCY-VOLTAGE PAIRS FOR THE INTEL XSCALE PROCESSOR USED IN OUR IMPLEMENTATION

Clock Frequency (MHz)	Supply Voltage (V)
733	1.5
666	1.4
600	1.3
533	1.25
466	1.2
400	1.1
333	1.0

Like most RTOSs, eCos requires a periodic interrupt to keep track of the internal OS tick, responsible for the notion of timing within the system. In the 80200 board, the only source of such an interrupt is the internal XScale performance counter interrupt. However, the interrupt is internal to the processor, and therefore cannot wake it up from the IDLE mode. To overcome this problem, we used a source of external interrupts to wake up the processor. The interrupt pin of the processor is connected to an Altera FPGA board, which generates periodic interrupts to wake up the processor. The period of the external interrupt is made equal to that of the smallest time period task to ensure that the processor is not woken up unnecessarily. The experimental setup, consisting of the XScale board, the MAXIM DC-DC converter board, the FPGA, and some interface circuitry is shown in Fig. 13(b).

1) Variable voltage supply: The variable voltage supply consists of a MAXIM 1855 DC-DC converter board, and some interface circuitry implemented using a PLD. When a voltage change is required, the processor sends a byte through the peripheral bus of the 80200 board to the interface circuitry which acts as an addressable latch. The outputs of this latch are connected to the digital inputs of the Maxim variable supply board. These inputs determine the output supply voltage of the Maxim board, which is fed back to the processor core. For the experiments, the system was configured to run at supply voltages from 1.0V to 1.5V and corresponding frequencies from 333 MHz to 733 MHz. The frequency-voltage pairs are listed in Table III. The frequency is changed using the XScale's internal registers.

2) Power measurement setup: We used a National Instruments Data Acquisition (DAQ) board to measure the power consumption. We isolated the power supply to the processor from the rest of the board, and measured the power consumed



(a) (b)

Fig. 13. (a) Power aware software architecture, and (b) Our experimental setup. The XScale board is on the top left, the Maxim board on the top right, and the FPGA board, which generates interrupts, is on the bottom right side. On the bottom left is a bread board with interconnections to trigger the DAQ board.

by the processor alone. For this purpose, we used a current-sense resistor (0.02 ohm) and sampled the voltage drop across it at a high sampling rate to obtain the instantaneous current drawn by the processor. We computed the energy consumption by integrating the product of the instantaneous supply voltage and current consumption over the interval of interest. The DAQ board was triggered by pulling signals out of the peripheral bus of the 80200 board to synchronize the power measurements with the task execution.

C. Experimental procedure and results

We implemented and compared three different DVS/DPM schemes, (i) shutdown based DPM, (ii) a non-adaptive version of the proposed algorithm, and (iii) the complete adaptive power-fidelity tradeoff DVS algorithm. As a baseline for comparison, we also conducted an experiment without any DVS/DPM. The following parameter values were used for the adaptive scheme, $T_1 = 2$, $T_2 = 0$, $WINDOW = 10$, $I = 0.1$, and $D = 0.05$. We used two values (0.95 and 0.85) for $ADAPTIVE\_LB$. We did not explore optimal choices for these parameters. Rather, our goal was to demonstrate the effectiveness of our algorithm by implementing it on a real system.

We used three different applications in our experiments: (i) an MPEG2 decoder, (ii) an ADPCM speech encoder, and (iii) a floating point FFT algorithm. Note that these applications run concurrently in the system. Each application executes as an eCos thread, and is scheduled using the RM priority assignment scheme. We profiled the execution time of each application for different input data to collect WCET statistics. For the MPEG2 decoder, different files were decompressed, and the WCET was measured separately for each one of them. The original FFT algorithm computes a fixed number (1024) of FFTs in one execution. In order to increase the variability in execution time, we also implemented a version of the FFT algorithm that computes a random number (obtained using a Gaussian distribution) of FFTs in each execution. Table IV shows the characteristics of the applications used. The WCET of each application instance was measured with the processor running at maximum frequency. The standard deviation is also given in Table IV to show the variability in the execution times for each application. We built three different task sets using these applications. The task sets, their component tasks, and the individual task characteristics are listed in Table V. The static slowdown factor is also listed for each task.

All the DVS/DPM algorithms shutdown the processor as soon as it becomes idle. The processor is woken up when the next external interrupt arrives. A limitation of the processor wake up strategy for our current testbed requires us to make all task periods a multiple of the highest-rate task. Thus, whenever the processor is woken up there is useful work to be done. This limitation could be eliminated through the use of a programmable interrupt generator. Further, note that this is not a limitation of the DVS algorithm or the software architecture itself. The static slowdown factors used by the proposed DVS algorithm are maintained in a table that is internal to eCos. Each task type is associated with a static factor, which is computed during system initialization. The dynamic and adaptive factors are maintained in a table of task instances. The task type table also contains a specified number (10 in our case) of execution times of previous task instances. The eCos kernel has full access to these tables. When a context

TABLE IV

APPLICATIONS USED IN THE EXPERIMENTS. T1 AND T2 BOTH PERFORM MPEG DECODING, BUT ON DIFFERENT FILES.

Task	Application	WCET at max. frequency (μs)	Std. Dev.
T1	MPEG2 (wg_gdo_1.mpg)	30700	3100
T2	MPEG2 (wg_cs_1.mpg)	26300	2100
T3	ADPCM	9300	3300
T4	FFT	15900	0
T5	FFT (Gaussian dist.)	13600	800

TABLE V

TASK SETS USED IN THE EXPERIMENTS. EACH TASK SET IS COMPRISED OF THREE TASKS.

Task Set	Component Task	WCET (μs)	Time Period (μs)	Deadline (μs)	Static Slowdown Factor
A	T2	26300	40000	40000	0.9495
	T3	9300	80000	80000	0.9495
	T4	15900	120000	120000	0.9495
B	T1	30700	47000	47000	0.8979
	T3	9300	94000	94000	0.8979
	T4	15900	141000	141000	0.8979
C	T1	30700	45000	45000	0.9207
	T3	9300	90000	90000	0.9207
	T5	13600	135000	135000	0.9207

TABLE VI

EXPERIMENTAL RESULTS FOR TASK SET A. THE TOTAL NUMBER OF TASK INSTANCES IS 415,207, AND 138 FOR TASKS T2, T3, AND T4 RESPECTIVELY.

DVS/DPM scheme used	Energy (J)	Power (W)	Ratio	No. of deadlines missed (T2/T3/T4)
No DVS/DPM	39.085	0.779	1	0/0/0
Shutdown	31.504	0.628	0.80	0/0/0
Non-adaptive	28.496	0.568	0.72	1/1/2
Adaptive (0.95)	26.581	0.527	0.68	3/2/1
Adaptive (0.85)	25.251	0.502	0.64	3/1/4

TABLE VII

EXPERIMENTAL RESULTS FOR TASK SET B. THE TOTAL NUMBER OF TASK INSTANCES IS 130,65, AND 43 FOR TASKS T1, T3, AND T4, RESPECTIVELY.

DVS/DPM scheme used	Energy (J)	Avg. Power (W)	Ratio	No. of deadlines missed (T1/T3/T4)
No DPM	12.546	0.798	1	0/0/0
Shutdown	11.265	0.716	0.89	0/0/0
Non-adaptive	9.811	0.624	0.78	1/0/1
Adaptive (0.95)	9.795	0.623	0.78	1/0/1
Adaptive (0.85)	8.828	0.562	0.70	1/1/31

switch occurs, the various tables are updated, and the voltage and frequency of the processor are adjusted.

The energy consumption, average power consumption, and number of deadline misses for the three task sets, under various DPM/DVS schemes, are shown in Tables VI, VII, and VIII. The column titled *Ratio* gives the energy consumption normalized to the case when no DVS/DPM is used. For the adaptive algorithm, the number in parentheses in the column *DVS/DPM scheme used* represents the value of $ADAPTIVE\_LB$. As expected, the energy savings increase as this parameter decreases, at the cost of more deadline misses. These results indicate that the proposed adaptive power-fidelity DVS algorithm does indeed result in considerable energy savings at the cost of a small number of missed deadlines.

VIII. SUMMARY AND FUTURE WORK

This paper presented an RTOS directed DVS scheme to reduce energy consumption in wireless embedded systems. A key feature of the proposed technique is that it yields an adaptive tradeoff between energy consumption and system fidelity. The proposed algorithm exploits low processor utilization, instance to instance variation in task execution times, and tolerance to missed deadlines of wireless systems to achieve this tradeoff. The technique has been incorporated into the eCos RTOS, and an energy efficient software architecture has been developed that facilitates application aware power management by enabling a dialog between the application and the RTOS. Involving the application in DVS/DPM can yield significant benefits. In many cases, the execution time is a superposition of several distinct distributions corresponding to different operating modes or distinct values of data. For example, a multiplier takes lesser time to compute its output if the operands are powers of two. Another example is an MPEG decoder whose histogram of run times has three distinct peaks corresponding to P, I and F frames. In such cases, an intelligent task can provide feedback to the OS in the form of a hint on the distribution of run times after the data values are known. As part of future work, we plan to investigate the energy reduction potential of such interactions in detail.

TABLE VIII

EXPERIMENTAL RESULTS FOR TASK SET C. THE TOTAL NUMBER OF TASK INSTANCES IS 130, 65, AND 43 FOR TASKS T1, T3, AND T5, RESPECTIVELY.

DPM Scheme Used	Energy (J)	Avg. Power (W)	Ratio	No. of deadlines missed (T1/T3/T5)
No DPM	13.080	0.838	1	0/0/0
Shutdown	12.342	0.772	0.94	0/0/0
Non-adaptive	10.892	0.693	0.83	0/1/18
Adaptive (0.95)	10.958	0.697	0.83	0/1/18
Adaptive (0.85)	9.990	0.637	0.76	11/16/32

REFERENCES

[1] A. P. Chandrakasan and R. W. Brodersen, *Low Power CMOS Digital Design*. Kluwer Academic Publishers, Norwell, MA, 1996.

[2] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, Norwell, MA, 1998.

[3] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Academic Publishers, Norwell, MA, 1997.

[4] T. A. Pering, T. D. Burd, and R. W. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms", in *Proc. ACM ISLPED*, pp. 76–81, 1998.

[5] Y. -T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration", *IEEE Trans. on CAD*, vol. 16, iss. 12, pp. 1477-1487, December, 1997.

[6] eCos real-time OS (`http://www.redhat.com/embedded/technologies/ecos`).

[7] Intel XScale microarchitecture (`http://developer.intel.com/design/xscale/`).

[8] D. Grunwald, et al., "Policies for dynamic clock scheduling", in *Proc. ACM Symp. on OSDI*, pp. 73–86, 2000.

[9] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation", *IEEE Transactions on VLSI Systems*, vol. 4, no. 1, pp. 42–55, March 1996.

[10] C. Hwang and A. Hu, "A predictive system shutdown method for energy saving of event driven computation", in *Proc. IEEE ICCAD*, pp. 28–32, 1997.

[11] T. Simunic, L. Benini, P. Glynn, and G. De Micheli, "Dynamic power management for portable systems", in *Proc. ACM MOBICOM*, pp. 11–19, 2000.

[12] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management", *IEEE Trans. on VLSI Systems*, vol. 8, iss. 3, pp. 299–316, June 2000.

[13] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy", in *Proc. ACM Symp. on OSDI*, pp.13–23, 1994.

[14] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithms for dynamic speed-setting of a low-power CPU", in *Proc. ACM MOBICOM*, pp. 13–25, 1995.

[15] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy", in *Proc. Annual Symp. on Foundations of Computer Science*, pp.374–382, 1995.

[16] I. Hong, M. Potkonjak, and M. B. Srivastava, "On-line scheduling of hard real-time tasks on variable voltage processors", in *Proc. IEEE ICCAD*, pp.653–656, 1998.

[17] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors", in *Proc. ACM ISLPED*, pp.197–202, 1998.

[18] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems", in *Proc. IEEE/ACM DAC*, pp. 134–139, 1999.

[19] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system", *IEEE Journal of Solid-State Circuits*, vol. 35, iss. 11, pp. 1571–1580, November 2000.

[20] T. A. Pering, T. D. Burd, and R. W. Brodersen, "Voltage scheduling in the lpARM microprocessor system", in *Proc. ACM ISLPED*, pp. 96–101, 2000.

[21] A. Manzak and C. Chakrabarty, "Variable voltage task scheduling for minimizing energy or minimizing power", in *Proc. IEEE ICASSP*, pp. 3239–3242, 2000.

[22] C. M. Krishna and Y. H. Lee, "Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems", in *Proc. IEEE RTAS*, pp. 156–165, 2000.

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

[24] F. Gruian,"Hard real-time scheduling for low energy using stochastic data and DVS processor", in *Proc. ACM ISLPED*, pp. 46–51, 2001.

[25] J. Pouwelse, K. Langendoen, and H. Sips, "Energy priority scheduling for variable voltage processors", in *Proc. ACM ISLPED*, pp. 28–33, 2001.

[26] J. Luo and N. K. Jha, "Power conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems", in *Proc. IEEE ICCAD*, pp. 357–364, 2000.

[27] J. Luo and N. K. Jha, "Battery aware static scheduling for distributed real-time embedded systems", in *Proc. ACM/IEEE DAC*, pp. 444–449, 2001.

[28] D. Zhu, R. Melhem, and B. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems", in *Proc. IEEE RTSS*, pp. 95–105, 2001.

[29] N. K. Jha, "Low power system scheduling and synthesis", in *Proc. IEEE ICCAD*, pp. 259–263, 2001.

[30] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment", *Journal of ACM*, vol. 20, pp. 46-61, January 1973.

[31] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-Completeness*. W. H. Freeman and Company, San Fransisco, CA, 1979.

[32] I. Hong, G. Qu, M. Potkonjak, and M. B.Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processors", in *Proc. IEEE RTSS*, pp. 178–187, 1998.

[33] A. Sinha and A. P. Chandrakasan, "Jouletrack: A web based tool for software energy profiling", in *Proc. IEEE/ACM DAC*, pp. 220–225, 2001.

[34] J. L. W. V. Jensen, "Sur les fonctions convexes et les inegalites entre les valeurs moyennes", *Acta Math.*, vol. 30, pp. 175–193, 1906.

[35] W. Namgoong and T. H. Meng, "A high-efficiency variable-voltage CMOS dynamic dc-dc switching regulator", in *Proc. IEEE ISSCC*, pp.380-381, 1997.

[36] V. Gutnik and A. .P. Chandrakasan, "Embedded power supply for low-power DSP", *IEEE Trans. on VLSI Systems*, vol.5, no.4, pp.425–435, December 1997.

[37] Intel StrongARM processors (`http://developer.intel.com/design/strong/`).

[38] A. Burns and A. Welling, *Real-Time Systems and their Programming Languages*. International Computer Science Series. Addison-Wesley, 1989.

[39] PARSEC parallel simulation language (`http://pcl.cs.ucla.edu/projects/parsec/`).

[40] A. Burns, K. Tindell, and A. Wellings, "Effective analysis for engineering real-time fixed priority schedulers", *IEEE Trans. on Software Engineering*, vol.21, pp.475-480, May 1995.

[41] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin, "Visual assessment of a real time system design: a case study on a CNC controller", in *Proc. IEEE RTSS*, pp. 300–310, 1996.

[42] C. Pereira, V. Raghunathan, S. Gupta, R. Gupta, and M. Srivastava, "A Software Architecture for Building Power Aware Real Time Operating Systems", Tech. Report #02-07, University of California, Irvine, March, 2002.

[43] Power Aware Distributed Systems project, UC Los Angeles and UC, Irvine (`http://www.ics.uci.edu/~cpereira/pads`).

[44] Intel 80200 Evaluation Board (`http://developer.intel.com/design/xscale/`).

Vijay Raghunathan received the B. Tech. degree in Electrical Engineering from the Indian Institute of Technology, Madras, in 2000, and the M.S. degree in Electrical Engineering from UCLA, in 2002, where he is currently pursuing a Ph.D. His research interests include system architectures and design methodologies for wireless embedded systems with emphasis on low power design, and energy efficient protocols for wireless communications. Vijay received the UCLA EE Department's Outstanding Masters Student Award for the year 2001-02 and the UC Regents graduate fellowship during the year 2000-01. He also received the best student paper award at the IEEE International Conference on VLSI Design in 2000. He is a student member of the IEEE.

Cristiano L. Pereira received the Bachelor degree in Computer Science from the Catholic University of Minas Gerais, Brazil, in 1997, and a MSc degree, also in Computer Science, from the Federal University of Minas Gerais, Brazil, in 2000. He is currently a Ph.D. student at the Computer Science and Engineering Department at the University of California, San Diego. His research interests are high level power management, operating systems and embedded software.

Mani B. Srivastava received the B.Tech. degree in Electrical Engineering from IIT Kanpur in 1985, and the M.S. and Ph.D. degrees from the University of California at Berkeley in 1987 and 1992 respectively. He is currently a professor in the Electrical Engineering Department at UCLA. Prior to joining UCLA, he was in the Networked Computing Research Department at Bell Laboratories from 1992 to 1996. His research interests at UCLA are in mobile and wireless networked embedded systems, focusing particularly on energy-aware computing and communications, low-power design, pervasive computing systems, and wireless sensor and actuator networks. He holds 5 US patents, and has published over 100 papers in the areas of wireless systems, sensor networks, and low-power embedded systems. He serves on the editorial board of IEEE Transactions on Mobile Computing and Wiley's Wireless Communications and Mobile Computing. He received the NSF CAREER award in 1997, the Okawa Foundation grant in 1997, and the President of India Gold Medal in 1985. He also received the best paper award at the IEEE International Conference on Distributed Computing Systems in 1997. He is a Senior Member of the IEEE.

Rajesh K. Gupta is a professor and holder of the Qualcomm endowed chair in embedded microsystems in the Department of Computer Science and Engineering at UC San Diego, California. He received his B.Tech. in Electrical Engineering from IIT Kanpur, India, M.S. in EECS from UC Berkeley and a Ph.D. in Electrical Engineering from Stanford University. His current research interests are in embedded systems, VLSI design and adaptive system architectures. Earlier he was on the faculty of Computer Science departments at UC Irvine and University of Illinois, Urbana-Champaign. Prior to that he worked as a circuit designer at Intel Corporation in Santa Clara, California on a number of processor design teams. He is author/co-author of over 150 articles on various aspects of embedded systems and design automation and three patents on PLL design, data-path synthesis and system-on-chip modeling. Gupta is a recipient of the Chancellor's Fellow at UC Irvine, UCI Chancellor's Award for excellence in undergraduate research, National Science Foundation CAREER Award, two Departmental Achievement Awards and a Components Research Team Award at Intel. Gupta is editor-in-chief of IEEE Design and Test of Computers and serves on the editorial boards of IEEE Transactions on CAD and IEEE Transactions on Mobile Computing. Gupta is a Fellow of the IEEE and a distinguished lecturer for the ACM/SIGDA and the IEEE CAS Society.