Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling

Yifan Zhu and Frank Mueller *
Department of Computer Science/Center for Embedded Systems Research
North Carolina State University,Raleigh, NC 27695-7534
mueller@cs.ncsu.edu, phone: +1.919.515.7889, fax: +1.919.515.7925

Abstract

Dynamic voltage scaling (DVS) is a promising method for embedded systems to exploit multiple voltage and frequency levels and to prolong battery life. However, pure DVS techniques do not perform well for systems with dynamic workloads where the job execution times vary significantly. In this paper, we present a novel approach combining feedback control with DVS schemes targeting hard real-time systems with dynamic workloads. Our method relies strictly on operating system support by integrating a DVS scheduler and a feedback controller within the EDF scheduling algorithm. Each task is divided into two portions. Within the first portion, the objective is to exploit frequency scaling for the average execution time. We reserve enough time for the second portion to meet the deadline requirements up to the worst-case execution time following a last-chance approach. Feedback techniques make the system capable to select the right frequency and voltage settings for the first potion, as well as guaranteeing hard real-time requirements for the overall task. Simulation experiments demonstrate the ability of our algorithm to save up to 29% more energy than previous work for task sets with different dynamic workload characteristics.

1. Introduction

Energy consumption is a major concern for real-time embedded systems due to their limited battery capacity. Contemporary embedded processors support multiple voltage and clock frequency settings. The energy consumption of a processor can be reduced by modulating voltage and frequency dynamically because the power dissipation of a CMOS circuit is proportional to its clock frequency and its voltage square[2]. We refer to dynamic voltage scaling (DVS) in the following whenever frequency or voltage are changed during execution.

In hard real-time systems, DVS techniques need to maintain the system timing requirements. Subsequently, minimizing energy consumption under DVS becomes a hard problem. Prior DVS techniques have been demonstrated to obtain significant energy savings for time-constrained embedded systems [21, 17, 1, 20, 1, 8]. However, pure DVS techniques do not perform well for dynamic systems where the system workloads vary significantly. Traditionally, hard real-time scheduling relies on *a priori* knowledge of the worst-case execution time (WCET) of a task to guarantee the schedulability of the system. However, experiments have shown a wide variation between longest and shortest execution times for many actual applications. In [22], actual execution times of real-world embedded tasks are observed to vary by as much as 87% relative to their measured WCET. Budgeting for the WCET may result in excessive energy consumption even though actual utilizations are low compared to the worst case. Many of the existing hard real-time DVS schemes are not able to adapt well to dynamically changing workloads. For example, we compared the energy consumption of Look-ahead RT-DVS [17] between constant workloads and fluctuating workloads, as depicted in Figure 1. The constant workloads consist of tasks whose actual execution times always equal to 50% of their WCET. The fluctuating workloads consist of tasks with an average execution time of 50% WCET. Their actual execution times fluctuate between 20% and 80% of their WCET (following variation patterns discussed later, similar to Figure 8). Figure 1 shows that, in the worst case, Look-ahead RT-DVS degrades up to 61% for fluctuating workloads.

The objective of our work is to develop a novel DVS technique targeting such dynamic changing workloads. We combine feedback control theory with DVS for hard real-time systems. Feedback control techniques have been shown to be a promising approach for real-time scheduling in prior work [12, 14, 16]. But all of them are for soft real-time systems, where occasional deadline misses are acceptable. Our work extends beyond previous work and is, to the best of our knowledge, the first study of using feedback control techniques on DVS for hard real-time systems. On one hand, feedback techniques enable the system to select the right frequency/voltage settings so that energy consumption is significantly reduced. On the other hand, feedback control helps to guarantee the timing constraints of hard real-time tasks so that no tasks ever miss their deadlines.

This paper is structured as follows. In Section 2, we give an framework overview of the feedback-DVS scheme. We then describe the different elements of our feedback-DVS framework in detail, *i.e.*, the voltage-frequency selector in Section 3, and the feedback controller in Section 4. Section 5 is an example showing how our scheme works on practical task sets. Section 6 presents the experimental results to

*This work was supported in part by NSF grants CCR-0208581, CCR-0310860 and CCR-0312695.

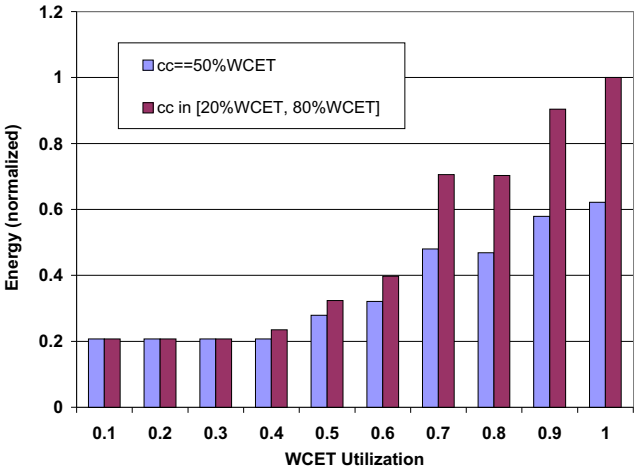


Fig. 1. Look-ahead RT-DVS Energy for Constant/Fluctuating Workload

demonstrate the performance of our feedback-DVS scheme under different workload conditions. Section 7 discusses related work, and Section 8 summarizes our efforts.

2. Feedback-DVS Framework Overview

Prior research on DVS for hard real-time system was primarily concerned with guaranteeing the schedulability of the task sets while energy consumption is minimized. But in a dynamic real-time environment where the workloads vary significantly from time to time, the DVS scheduler should not only produce a valid processor speed for each scheduling unit, it should also be able to adapt to the ever-changing workloads as fast as possible. One important performance metric of such a system is how fast the DVS scheme can adjust the processor according to different workloads so that energy consumption is significantly reduced. To address this issue, we propose a framework called feedback dynamic voltage scaling (feedback-DVS). In this framework, we consider the scheduling problem in hard real-time systems with the earliest deadline first (EDF) policy. The framework is based on feedback control that incrementally corrects system behavior to achieve its targets, while the hard real-time timing requirements are still preserved. We assume that the processor can operate at several discrete voltage/frequency levels, which represents contemporary processor technology on support of DVS. When there is no task running on the processor, the processor enters an idle state at a particular voltage/frequency level, usually the lowest voltage/frequency level on that processor.

We use a periodic, fully preemptive and independent task model in our feedback DVS framework. We assume there are n tasks in total, T_1, T_2, ..., and T_n. Each task T_i is defined by a tuple (P_i, C_i), where P_i is the period of T_i, and C_i is the measured worst-case execution time of T_i. Each task's relative deadline d_i is equal to its period, and all tasks start at time 0. The periodically released instances of a task are called jobs. T_{ij} is used to denote the j^{th} job of task T_i.

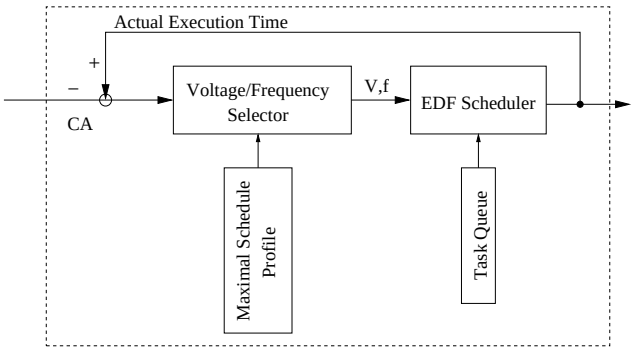


Fig. 2. Feedback-DVS Framework

Its release time is $P_i * (j-1)$ and its deadline is $P_i * j$. The hyperperiod H of the task set is defined as the least common multiplier (LCM) of the tasks' periods. At the end of each hyperperiod, the schedule repeats.

Figure 2 depicts the framework of our feedback-DVS scheme. It consists of a voltage-frequency selector, a maximal schedule profile and an EDF scheduler. The voltage-frequency selector calculates the error from the difference between the actual execution time of a task and C_A, the execution time of the first portion of a tasks (detailed in the task-splitting scheme in the next section). It then selects a voltage/frequency level according to the error and the maximal schedule profile. The error is used by a PID feedback controller to adjust the estimation of the execution time for the next job. The maximal schedule profile includes a running scenario of the task set from the start time 0 to the end of a hyperperiod. It is generated offline assuming each task's actual execution time always equals its worst-case execution time. The voltage-frequency selector uses the information in the maximal schedule profile to choose the right voltage-frequency level while guaranteeing that no tasks miss their deadlines. After the voltage/frequency level is determined, the EDF scheduler schedules the next ready task at the specific processor speed. Tasks are scheduled according to EDF policy, *i.e.*, the task with the earliest deadline is given the highest priority. The actual execution time of each task is further fed back to the voltage-frequency selector for later decision making. The next two sections detail the mechanism of the voltage-frequency selector and the feedback controller in our feedback-DVS frame.

3. Voltage-Frequency Selector

The voltage-frequency selector is responsible for selecting a voltage-frequency pair each time a task is scheduled. Since power consumption increases proportional to the processor frequency and to the square of the voltage in CMOS circuits [7], the minimal energy consumption is obtained by running every tasks at a uniform processor speed. But this is only a static optimal solution. In a dynamic environment where a task's actual execution time is unknown until the task completes, it is not possible to derive the optimal uni-

form speed in advance. Our objective is to approximate a close-to-optimal solution by monitoring the actual execution time of each task. The start point of our scheme is the following inequation, which is a modification of the standard EDF [10] schedulability test:

$$\alpha^{-1}\frac{C_k}{P_k} + \sum_{i\in\{1,\ldots,n\}\setminus\{k\}} \frac{C_i}{P_i} \leq 1 \quad (1)$$

where α is a scaling factor defined as the ratio of the current processor frequency to the maximal available frequency, *i.e.*, $\alpha = f_k/f_m$. Instead of scaling at a single speed for all tasks, only the highest priority task (the task with the earliest deadline under EDF) is scaled. All remaining tasks still execute at the maximum frequency f_m with a scaling factor of 1. The motivation of scaling only the current task is that a greedy scheme usually gives a near-optimal result when optimal solutions are unavailable.

For each task, its α value depends on the total available slack when the task is scheduled. For example, at time 0, the available slack for the first task T_1 is derived from Inequation 1 as $P_1(1 - \sum_{i=2}^{n}\frac{C_i}{P_i})$. Its α value is calculated as: $\alpha = \frac{C_1}{P_1(1-\sum_{i=2}^{n}\frac{C_i}{P_i})}$. In order to obtain an even lower speed for each task T_k and to make feedback control available for hard real-time systems, our scheme goes beyond that by splitting each task into two subtasks T_A and T_B. These two subtasks are allowed to execute at different frequency and voltage levels. As shown in Figure 3, T_B always

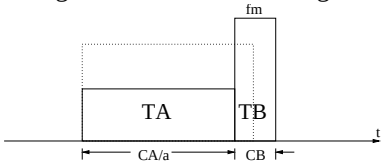


Fig. 3. Task Splitting

executes at the maximum frequency level f_m, while T_A is able to execute at a lower frequency level than it could without task splitting. We expect that a task can finish its actual execution within T_A while reserving enough time in T_B to meet the deadline if its WCET is exhibited in full. With this scheme, we can safely scale the frequency within T_A using available slack while T_B executes at maximum frequency following a last-chance approach [3]. In the next section, we can also see that such a task splitting scheme is necessary for applying feedback control on hard real-time systems. Let C_A and C_B be the worst-case execution cycles of subtask T_A and T_B, and s_k be the slack available to T_k when T_k is scheduled, from:

$$C_k = C_A + C_B, \frac{C_A}{\alpha} + C_B = C_k + s_k \quad (2)$$

we get:

$$\alpha = \frac{C_A}{C_A + s_k} \quad (3)$$

Equation 3 shows that when task splitting is used, the scaling factor α depends not only on the amount of available slack, but also on the number of execution cycles of T_A.

Other methods, such as idle time utilization and slack passing, are also used in our scheme to achieve a low energy consumption as detailed in [4]. Both schemes, only briefly outlined here, are based on a comparison between the *actual* schedule and the worst-case or *maximal* schedule,*i.e.*, the schedule produced by EDF when the execution time of every task's job has its maximum value given by the WCET. We call the schedule produced by our DVS-EDF algorithm the *actual* schedule, where the execution time of every task's jobs may be scaled. The maximal schedule is constructed offline in O(N) complexity, where N is the total number of jobs executed in a hyperperiod H. The key to idle utilization is to add an idle task into the original task set.

The actual execution time of the idle task is always zero, so that the actual system execution is not affected by the behavior of the idle task, *i.e.*, the idle task only exists in the maximal schedule. The WCET and the period of the idle task are chosen in such a way that the total utilization of the new task set becomes 100%. Specifically, we let $P_{idle} = P_{min}$. By choosing the minimum period among all tasks for the idle task, slack in the maximal schedule becomes available as early as possible for scaling other tasks. Slack passing is a technique to decrease the complexity of calculating the amount of slack in the system. Instead of computing the available slack from the scratch for each newly released job, the previous job passes its unused amount of slack ($s_{k,r}$) to the next job. The unused slack is further augmented by any idle slots between the deadline of the previous job and the next job, as follows:

$$s_{k+1} = s_{k,r} + idle(d_k...d_{k+1}) \quad (4)$$

where $idle(d_k...d_{k+1})$ is the sum of WCETs of all idle tasks in $[d_k...d_{k+1}]$. Next, we demonstrate that task preemption requires special handling and derive formulas to compute $s_{k,r}$.

3.1. Preemption Handling

When preemption occurs, the preempted task will relinquish its remaining slack and pass it on to the next task, just as it does when a task completes. This follows a greedy scheme in that we try to pass as much slack as possible to scale the running task and to speculate on its early completion to aggregate more slack for following tasks. There are also two differences here. First, the preempted task itself cannot generate any slack based on its own execution at the preemption point since the task's completion time is unknown. Hence, no additional slack is added to its inherited total slack. Second, the preempted task still needs some time to complete its execution in the future. The remaining execution time must be reserved in advance to avoid future deadline misses caused by over-exploiting slack from other tasks. At the preemption point, the expected remaining execution time $left_{ij}$ of the preempted task is:

$$left_{ij} = C_i - c_{ij} \times \alpha\prime \quad (5)$$

where c_{ij} is the actual execution time up to the preemption point. Our slack passing scheme promises that the pre-

empted task will not miss its deadline by reserving corresponding slack:

$$s_{k,r} = s_k - left_{ij} \qquad (future\ slots) \qquad (6)$$

The old slack is derived from Equation 4 and the resulting slack $s_{k,r}$ can be passed to the next task.

Future slot allocation in this manner is essential to ensure the feasibility of the schedule under DVS. Future slots will be allocated only if the maximal schedule does not include sufficient slots for the preempted task's job between the preemption point and its deadline. We devised multiple schemes for reserving these slots.

- Forward sweep: When a task $T1$ is preempted and requires $left_{1j}$ future slots, the preempting task $T2$ deducts this amount from its available slack s. If $left_{1j} > s$, then $T2$ remains without slack. If another task $T3$ is initiated, the calculation repeats itself.
- Backward sweep: Future slots of $T1$ are allocated in idle slots within the maximal schedule from its deadline $d1$ backwards. Any of these idle slots become unavailable for slack generation, *i.e.*, these slots are excluded in Equation 4.

An example is depicted in Figure 4. The upper time line of idle slots presents a excerpt of the maximal schedule that depicts idle task allocations, only. The lower time line shows the dynamic schedule of tasks. Upon release of $T2$ at t2, $T1$ is preempted. Let us assume that $T1$ does not have sufficient static slots (three slots) beyond t2 to finish its execution. Hence, it has to rely on future idle slots. During $T2$'s execution, $T3$ is released. Both $T2$ and $T3$ have smaller deadlines than $T1$ ($d2 < d3 < d1$). Subsequently, $T1$ only resumes some time after $T3$ completes.

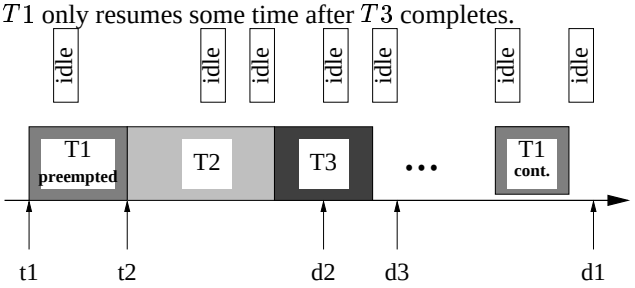


Fig. 4. Future Slot Reservation

Future slot allocation of $T1$ then depends on the chosen scheme. The forward sweep results in zero idle slack for $T2$ and $T3$ since idle slots during the tasks' periods are not sufficient to cover $T1$'s future needs of three slots at the respective invocation times. The backward sweep, on the other hand, reserves the last 3 idle slots (from d1 backwards), such that $T2$ and $T3$ may consume at least two and one idle slots for scaling, respectively, even if they use up their time quantum in full.

Overall, the forward sweep is not as greedy as the backward sweep in the sense that earlier tasks may not be scaled due to $T1$'s future slots. A forward sweep is likely to result in zero slack for the preempting task $T2$ if $P2 << P1$, *i.e.*, if its period is much shorter. There are simply fewer idle slots available, which may not suffice to cover $T1$'s future requirements. More idle slots past $d2$ will be required in this case. The backward sweep always results in the most greedy approach in delaying the needs of $T1$ as long as possible. This is consistent with the observation that early completion is likely to generate slack for each task, a property inherent to our algorithm.

4. PID Feedback Controller

Equation 3 shows that the scaling factor α depends not only on the amount of available slack but also on C_A, the number of execution cycles assigned to T_A. The slack passing and forward/backward sweep schemes, as described in the previous section, help us to determine the amount of slack available for each task. In this section, we focus on another key issue, *i.e.*, how to determine the value of C_A. Since C_A is based on the estimated worst-case execution time of the first subtask T_A, our objective is to let C_A approximate T_{ij}'s actual execution time c_{ij} so that T_{ij}'s actual execution can be completed at the low frequency level corresponding to α. If C_A were not exceeded by the actual execution time c_{ij}, there would be no need for the task to enter the second subtask portion T_B and switch to the maximum processor frequency. Hence, the entire task could execute at a low frequency, and a near-optimal energy consumption would be obtained.

In real-time applications, the actual execution time c_{ij} of tasks T_i often experiences fluctuations over different intervals. The fluctuations may result in tendencies leading to higher processing demands up to some point and receding demands after that peak point. Past work in dynamic real-time scheduling has demonstrated that adaptive techniques derived from control theory can enhance a schedule by reacting to tendencies in execution time fluctuations [12]. In order to devise a DVS-EDF algorithm adaptive to such a dynamic environment, we integrated a PID-feedback controller into our DVS systems.

Feedback control is one of the fundamental mechanism for dynamic systems to achieve equilibrium. In a feedback system, some variables, *i.e.*, controlled variables, are monitored and measured by the feedback controller and compared to their desired values, so-called set points. The differences (errors) between the controlled variables and the set points are fed back to the controller for further actions. Corresponding system states are usually adjusted according to the differences to let the system variables approximate the set points as closely as possible.

PID-feedback control is a continuous feedback controller. A PID controller consists of three different elements, namely, proportional control, integral control, and derivative control. Proportional control influences the speed of the system adapting to errors, which is defined as the difference between the controlled variable and the set point, by a pure proportional gain item. Integral control is used to

adjust the accuracy of the system through the introduction of an integrator on past error histories. Derivative control usually increases the stability of the system through the introduction of a derivative of the errors. The PID feedback controller can be described in three major forms: the ideal form, the discrete form and the parallel form. Although the discrete form is often used in digital algorithms to keep tuning similar to electronic controllers, the parallel form is the simplest one. The integral and derivative actions are also independent of the proportional gain in the parallel form. We choose the following parallel form as the base of our PID feedback implementation:

$$output = K_p * \epsilon(t) + \frac{1}{I}\int \epsilon(t)\,dt + D\frac{d\epsilon(t)}{dt} \quad (7)$$

where K_p, I and D are the proportional, integral and derivative coefficients, respectively, and $\epsilon(t)$ is the system error.

We integrated the above PID controller into our DVS scheme to control the number of execution cycles assigned to C_A. According to the objective described above, we choose the value of C_A as the controlled variable while c_{ij} is chosen as the set point. The system error is defined as the difference between the controlled variable and the set point, *i.e.*,

$$\epsilon(t) = c_{ij} - C_A. \quad (8)$$

The error is measured periodically by the PID controller. Its output is fed back to the DVS-EDF scheduler to adjust the value for C_A. Let C_{Aij} be the estimated C_A value for the j^{th} job of a task T_i. The following discrete PID control formula is used in our DVS-EDF scheduler:

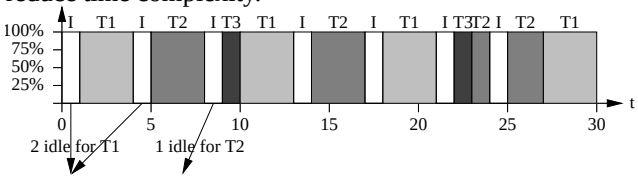
$$\Delta C_{Aij} = K_p * \epsilon(t) + \frac{1}{I}\sum_{IW}\epsilon(t) + D\frac{\epsilon(t)-\epsilon(t-DW)}{DW}$$
$$C_{Ai(j+1)} = C_{Aij} + \Delta C_{Aij} \quad (9)$$

where K_p, I and D are proportional, integral, and derivative coefficients, respectively. $\epsilon(t)$ is the monitored error. The output ΔC_{Aij} is fed back to the system and is used to regulate the next anticipated value for C_A. IW and DW are tunable window sizes such that only the errors from the last IW (DW) task jobs will be considered in the integral (derivative) term. We use $DW = 1$ to limit the history, which ensures that multiple feedback corrections do not affect one another.

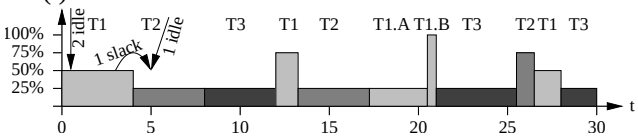
Due to the task slitting scheme, all tasks can still meet their deadline, even if the PID feedback controller does not adjust the C_{Aij} value close enough to c_{ij}. For example, if $C_{Aij} \leq c_{ij}$, the task will enter its second portion and run T_B at the maximal frequency level. The feedback scheme, together with the task splitting scheme, guarantees the deadline requirements of real-time tasks.

5. Example

Combining all the techniques illustrated above, we now turn to a description of the entire algorithm. Our algorithm starts with an offline construction of the static maximal EDF schedule within the interval of the hyper-period. Figure 5(i) shows an example of such a maximal EDF schedule. The example includes a task set of three tasks T1=$\{3,8\}$, T2=$\{3,10\}$ and T3=$\{1,14\}$, where $T_i = \{C_i, P_i\}$ denotes task T_i's worst case execution time C_i and its period P_i. An idle task I=$\{1,4\}$ is also included in the maximal schedule to fill underutilized processor time niches. Every task's actual execution time is 1 except the first job of T1, who has an actual execution time of 2. All scheduling events (task release, preemption, resumption, and completion) of the maximal EDF schedule are stored in a look-up table to reduce time complexity.



(i) Static Worst-Case EDF Schedule with Idle Task I



(ii) Our Feedback DVS at Beginning of 1st Hyperperiod

Fig. 5. Discrete Scaling Levels for 3 Tasks

Next, the task set is scheduled according to our algorithm (without the idle task). Additional operations to calculate slack and to set the CPU frequency/voltage are inserted at scheduling points. As shown in Figure 5(ii), when the first task T (with the earliest deadline) is activated at time 0, its initial slack is assigned according to Equation 4. The initial $s_{1,0}$ is set to 0 since no previous task had been scheduled. The value of $idle(0..d_1)$ is obtained from the pre-calculated maximal EDF schedule. Then, a frequency scaling factor α is set according to Equation 3: $\alpha = C_A/(C_A + s_k)$. The CPU frequency is set to $\alpha * f_m$. When the first task completes, unused slack is adjusted and passed on to the next task according to Equations 5 and 6. The estimated value of C_a for the first task is updated according to our feedback scheme. When the second task is scheduled, its slack is again determined by Equation 4, this time with a non-zero slack on the right-hand side of the equation (since the first task passes no unused slack). The frequency level is determined in a similar way as the first task. For later task instances, the feedback scheme chooses C_A to approximate the task's actual execution time. Hence, the entire task is scaled at a low frequency level. Preemption handling, as described in Section 3.1, is also applied but not shown here to simplify the example.

An algorithmic description of our DVS-EDF scheme integrated with the PID feedback control is given in Figure 6. This algorithm is a refinement of our previous work [4] and integrates the PID feedback scheme and preemption handling with future slot reservation. The online complexity of our algorithm is $O(n)$ for n tasks, because the length of slots in the maximal schedule during the interval between the release time and deadline of the current task have to be updated when a task is released or completes. The number

Procedure Initialization
  for each $T_k \in \{T_1, T_2, \ldots, T_n\}$ **do**
    $C_{Ak} \leftarrow C_k/2$
    $left_{k0} \leftarrow C_k$
    $t_i \leftarrow 0$
  $U \leftarrow \frac{C_1}{P_1} + \frac{C_2}{P_2} + \ldots + \frac{C_n}{P_n}$
  $P_{n+1} \leftarrow P_1$
  $C_{n+1} \leftarrow P_1 \times (1 - U)$
  $c_{n+1} \leftarrow 0\}$
  $slack \leftarrow 0$

Procedure TaskActivation($\mathbf{T_{ij}}$)
  if processor was idle for d **then**
    $slack \leftarrow slack - d$
  if T_{pk} preempted/interrupted **then**
    $left_{pk} = C_p - c_{pk} \times \alpha\prime$
    $slack \leftarrow slack - idle(d_{ij}..d_{pk})$
  if $left_{pk} > slots(T_{pk}, now..d_{pk})$ **then**
    $reserve_{pk} \leftarrow left_{pk} -$
      $slots(T_{pk}, now..d_{pk})$
    allocate $reserve_{pk}$ in
      $idle(now..d_{pk})$
        $completed(now..d_{pk})$
    $slack \leftarrow slack - reserve_{pk}$
else (T_{pk} *completed execution*)
  if $now > d_{pk}$ **then**
    $slack \leftarrow slack - idle(d_{pk}, now)$
  $slack \leftarrow slack + idle(d_{pk}..dij)$
$\alpha\prime \leftarrow \min\{\frac{f_1}{f_m}, \ldots, \frac{f_m}{f_m} | \frac{f_i}{f_m} \geq \frac{C_{Aij}}{C_{Aij}+slack}\}$
if $(\alpha\prime = 1)$ **then**
  $C_A \leftarrow 0$
else
  $C_A \leftarrow slack \times \alpha\prime/(1 - \alpha\prime)$
SetInterrupt($T_i, C_A/\alpha\prime$)
SetFrequency($\alpha\prime$)

Procedure TaskCompletion($\mathbf{T_{ij}}$)
  $slack \leftarrow slack - c_{ij} + C_i$
  $\epsilon \leftarrow c_{ij} - C_{Aij}$
  $\Delta C_{Aij} \leftarrow K_p * \epsilon(t_i) + \frac{1}{I}\sum_{IW} \epsilon(t_i) +$
    $D\frac{\epsilon(t_i) - \epsilon(t_i - DW)}{DW}$
  $C_{Ai(j+1)} = C_{Aij} + \Delta C_{Aij}$
  $t_i \leftarrow t_i + 1$
  $left_{i(j+1)} = C_i$
  if $reserve_{ij} > 0$ **then**
    release $idle(now..d_{ij}) +$
      $completed(now..d_{ij})$
      up to $|reserve_{ij}|$
Procedure SetInterrupt($\mathbf{T_{ij}}, \mathbf{C_A}$)
  Set timer interrupt for T_{ij},
  triggered C_A time units later
Procedure SetFrequency($\alpha\prime$)
  $f \leftarrow \alpha\prime \times f_m$

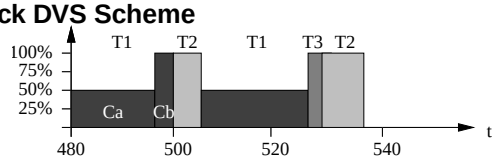
Fig. 6. Pseudocode of Feedback DVS Scheme

of slots in this interval is bounded by the number of tasks since only a constant number of jobs for each task and a constant number of preemptions may occur in this interval. We use the following notation:

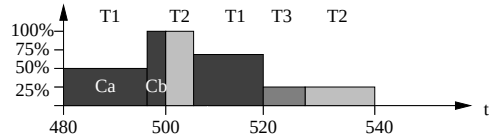
- T_{ij}: the j-th job of task T_i
- ij, pk : indices for the current and previous tasks relative to T_{ij}
- now: the current time
- r_{ij}: the release time of T_{ij}
- d_{ij}: the deadline of T_{ij}
- C_i: the WCET of T_i (without scaling)
- c_{ij}: the actual execution time of T_{ij} up to now (with scaling)
- $left_{ij}$: the remaining WCET of T_{ij} (without scaling)
- $slack$: system current slack
- $idle(t1..t2)$: the amount of idle slots between times [t1,t2]
- $completed(t1..t2)$: slots of already completed tasks between times [t1,t2]
- $slots(T_{ij}, t1..t2)$: the amount of time slots reserved for T_{ij} in the worst case between times [t1,t2]

The effect of the PID feedback scheme is shown in the following example. Consider a task set of three tasks T1={12,32}, T2={12,40} and T3={4,65}. Let the actual execution times of different jobs of a task fluctuate according to the execution time pattern 1, as depicted in Figure 8. Figure 7(a) is a snapshot of the DVS-EDF schedule for this task set without PID-feedback. Figure 7(b) depicts the DVS-EDF schedule for the same task set using feedback with PID parameters CP=0.9, CI=0.08 and CI=0.1.

We can see from the figures that the first job of T_3 and the second job of T_2 are scheduled to run at a much lower frequency in the PID feedback schedule than the one without PID-feedback. The first job of T_3 with an actual execution time of 2.57 starts at time 524 in the schedule without



(a) DVS−EDF Schedule without PID−Feedback



(b) DVS−EDF Schedule with PID Feedback

Fig. 7. Schedules: Simple and PID Feedback

PID-feedback, and starts at time 520 in the PID feedback schedule. The PID feedback scheme gets an execution time of 3.06 for its C_A according to Equation 4. With the closer approximation of c_{ij}, the PID scheduler is able to scale the task more aggressively than the one without PID-feedback. Similarly, the non-feedback schedule only gets an average execution time of 5.26 for the second job of T_2, which has an actual execution time of 7.07. But the PID feedback scheme obtains a $C_A = 6.76$, which is again closer to T_2's actual execution time. This demonstrates the superiority of our feedback-DVS scheme in adapting to dynamic workloads resulting in additional energy savings.

6. Experiments

We evaluated the performance of our schemes in a simulation environment which supports feedback-DVS scheduling. In order to make a comparison with our algorithm, Pillai and Shin's [17] Look-ahead RT-DVS algorithm was also implemented. We assume a processor model capable of operating at four different voltage and frequency levels, as depicted in Table 1. Comparable frequency and voltage setting were also used in the Look-ahead RT-DVS work [17] and the experimental work with StrongARM processors [18]. When there are no ready tasks available for scheduling, the

processor enters an idle state and operates at the lowest frequency and voltage level. We use a simplified energy model in our experiment as $E = fV^2t$. Energy values reported in the following experiments were normalized for ease of comparison.

frequency	voltage
25%	2 V
50%	3 V
75%	4 V
100%	5 V

Table 1. Processor Model for Scaling

Altogether 50 task sets were generated, each consisting of 3 tasks. In our experiments, we first investigated the performance of our scheme over fluctuating workload patterns. The objective in studying different patterns is to assess the sensitivity of feedback DVS to different types of fluctuations, which have been observed in interrupt-driven systems [15]. As shown in Figure 8, we constructed three synthesized execution time patterns to simulate realistic workloads. In the first pattern, the actual execution time of a

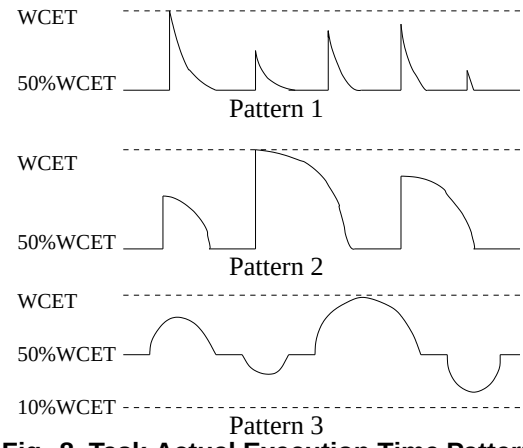


Fig. 8. Task Actual Execution Time Pattern

job starts at 50% of the task's WCET before spiking to a peak value c_m every 10th job. The peak value c_m is randomly generated for each spike from a uniform distribution between 50% of WCET and 100% of WCET. After the peak value is reached, the actual execution time of the following jobs drop exponentially (modeled as $c_i = 1/2^{(t-c_m)}$) until it reaches 50% of WCET again. This pattern simulates event-triggered activities that result in sudden, yet short-term computational demands due to complex inputs often observed in interrupt-driven systems. In the second execution time pattern, the peak execution time c_m still follows a random uniform distribution between 50% of WCET and 100% of WCET. But the actual execution time of the following jobs initially drops more gradually, modeled as $c_i = c_m sin(t + \pi/2)$. This pattern simulates events resulting in computational demands in a phase of subsequent complex inputs (with a decaying tendency). In the third execution pattern, the actual execution time of the jobs al-

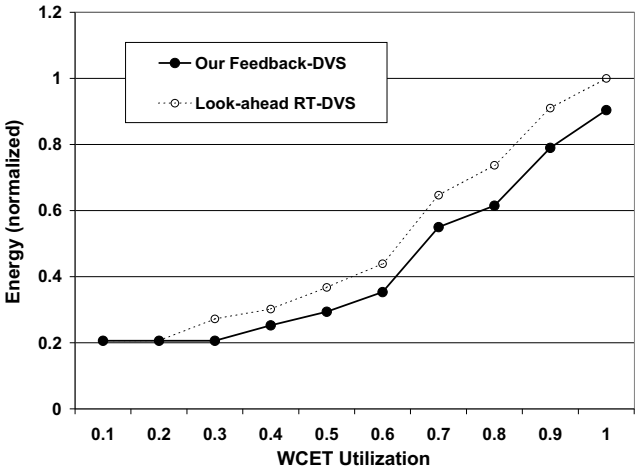


Fig. 9. Execution Time Pattern 1

ternates between positive and negative peaks every 10 jobs. Both the peak values in either direction are randomly generated from a uniform distribution between 50% of WCET and 100% of WCET. The actual execution time of the jobs following the peak value is modeled as $c_i = c_m sin(t)$ and $c_i = -c_m sin(t)$. This pattern represents periodically fluctuating activities with gradually increasing and decreasing computational needs around peaks. For each execution time pattern, the task sets' WCETs were uniformly distributed in the range [10,1000]. When tasks' WCETs were generated, each task's period was chosen so that the worst case utilization of the task set (*i.e.*, $\sum \frac{WCET_i}{P_i}$) varies from 0.1 to 1.0 in increments of 0.1. Different combinations of PID coefficients were investigated in our experiments. It was observed that both increasing or decreasing the proportional coefficient resulted in less accurate system estimations for C_A. The derivative item is less significant compared to the other two parameters. Increasing the integral window size improves the energy saving effect in the very beginning, but when IW becomes larger than 10, no dramatic system performance improvements were observed. We restrict ourselves here to report results based on the PID coefficients of $K_p = 0.9$, $I = 0.08$, $D = 0.1$. The derivative and integral window size were 1 and 10, respectively.

Figure 9 compares the energy consumption between our feedback-DVS scheme and the Look-ahead RT-DVS scheme under the execution time pattern 1. When the task set utilization is less than 0.3, it is observed that both schemes consume the same amount of energy. This is because task sets with low utilizations usually have enough slack and idle slots, so that all jobs were able to be scaled to the lowest speed level. In this case the processor always operates at the 25% frequency level and consumes the same amount of energy for both schemes. With the increase of the worst-case utilization, our feedback-DVS scheme started saving more energy than Look-ahead RT-DVS. Our scheme adapts to the changing workload better than Look-ahead RT-DVS and costs 8% to 24% less energy

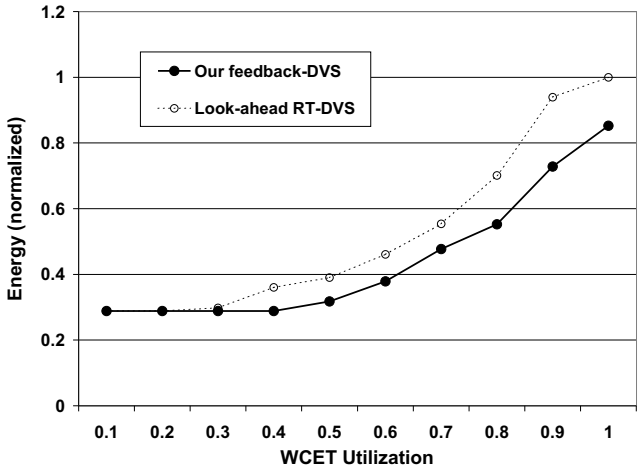


Fig. 10. Execution Time Pattern 2

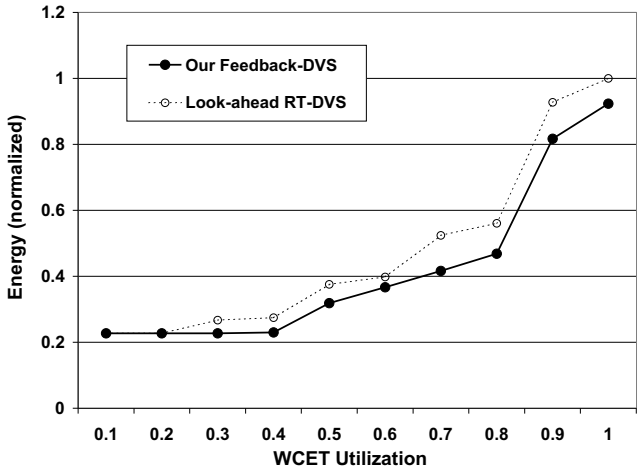


Fig. 11. Execution Time Pattern 3

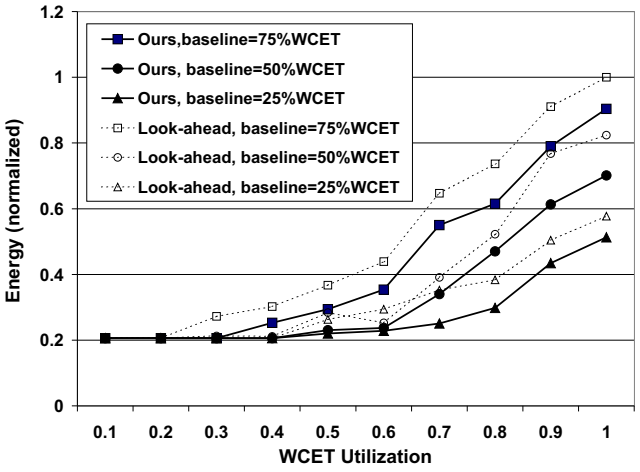


Fig. 12. Varying Baseline under Pattern 1

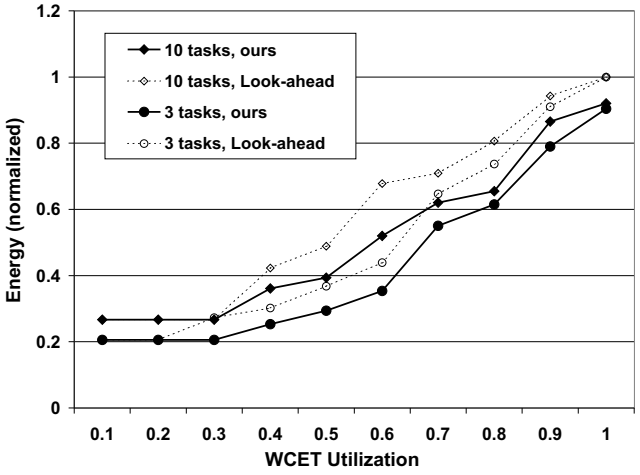


Fig. 13. 10-tasks vs. 3-task under Pattern 1

than it. The maximal energy savings (24%) can be observed at 80% utilization. Similar results can be observed for execution time pattern 2 and 3, as depicted in Figures 10 and 11. The maximal energy savings, 22% and 16%, are observed at 0.5 and 0.9 utilziations, respectively. The average energy saving over Look-ahead RT-DVS is around 15%. These experiments show that our feedback-DVS scheme is not sensitive to different patterns of fluctuating workloads.

In order to further observe the scalability of our algorithm, we generated three task sets following execution time pattern 1, but with different baseline values. While the pattern depicted in Figure 8 has a 50% WCET baseline, the other two task sets have baselines of 75% and 25% WCET, respectively. Shifting the baseline among different task sets also results in a change of their actual utilizations. Figure 12 compares the energy consumption between feedback-DVS and Look-ahead RT-DVS for these three task sets. The energy values are normalized to the maximal point of the 75% WCET baseline task set. The result shows that our scheme is able to scale to task sets with different baselines very well. Feedback-DVS saved up to 20% more energys than Look-ahead RT-DVS for a baseline of 75% WCET case. When the baseline is 25% of the WCET, up to 29% more energy savings are observed. The maximal energy saving appears in the task set with 25% WCET baseline since it provides the largest range for execution time fluctuation.

Figure 13 illustrates the performance of the feedback-DVS scheme by varying the number of tasks in the task sets. We compared the energy consumption between our algorithm and Look-ahead RT-DVS for task sets with 10 and 3 tasks. The energy values are normalized to the maximal point in Look-ahead RT-DVS for the 10-task set. We notice that there is little effect of varying the number of tasks on our scheme. We are able to save almost the same percentage of energy over Look-ahead RT-DVS between 10-task sets and 3-task sets. However, a larger number of tasks tends to result in lower overall energy consumption.

Besides the execution time patterns listed in Figure 8, we also investigated the task sets with truly random characteristics, *i.e.*, tasks' actual execution times do not follow any pre-defined patterns and are generated completely from a random uniform distribution. Experiments reveal that our feedback-DVS scheme does not give additional benefits over Look-ahead RT-DVS for such cases , because truly

random execution times cannot provide any useful history information to our feedback controller. This is the limitation of our (or any other) feedback scheme.

Overall, our Feedback DVS-EDF algorithm is able to exhibit considerable energy savings for different task sets. Feedback control in conjunction with DVS scheduling makes the system more adaptive to dynamically changing workloads. Our scheme achieves lower energy consumption levels than other less adaptive schemes.

7. Related Work

There have been a number of efforts of applying feedback techniques on general-purpose control systems. But only recently did researchers begin to incorporate feedback control to real-time scheduler with timing constraints [12, 13]. Lu *at al.* proposed a feedback control real-time scheduling framework for unpredictable dynamic real-time systems where task execution times diverge from their worst case [13]. Real-time system performance specifications are analyzed and satisfied systematically through a control theory-based methodology. Dynamic models of real-time systems are developed to identify different categories of real-time applications with different feedback control algorithms. While their feedback control framework is mainly used to satisfy general purpose real-time system requirements, our scheme focuses on exploiting feedback control schemes to reduce energy consumption.

Our work is more closely related to the ones in [14] and [16]. Lu *et al.* describe a formal feedback control algorithm combined with dynamic voltage/frequency scaling technologies for multimedia systems [14]. Both continuous and discrete DVS settings are exploited in a scheme to reduce energy consumption while still guaranteeing real-time requirements. An adaptive set-point is used to achieve fast responses with a stable multimedia throughput. Both their work and our approach exploit feedback control to DVS/DFS technologies. They target soft real-time/multimedia systems, while we focus on hard real-time systems where timing constraints must not be violated.

A general energy management scheme with feedback control was proposed by Minerick *at al.* [16]. An average energy usage is achieved by continuously adjusting the voltage/frequency of a processor to meet the energy consumption goal. A PI (proportional and integral) feedback controller is used to adapt the proper power setting based on previous energy consumptions without the prediction of future system workloads. While their objective is to obtain low energy consumption for general purpose systems, we target hard real-time systems with deadline requirements.

Dynamic voltage scaling has been studied by many previous researchers. Saewong *et al.* [19] proposed a series of voltage scaling schemes targeting different hardware configurations and task set characteristics. Their results showed that some non-optimal schemes may be more suitable than optimal schemes when the system has a high voltage scaling overhead. Lee *et al.* [9] presented a branch-and-bound algorithm to statically determine the operating frequency of real-time task sets. But due to the complexity of the algorithm, only two frequency levels are assumed in their model. The algorithm proposed in [11] derives optimal speed functions between an upper bound and a lower bound of processor cycles. Their online algorithm reclaims unused execution cycles to further reduce energy consumption. The algorithms in [17, 1, 5] are more closely related to ours. Pillai and Shin [17] proposed a set of dynamic DVS algorithms based on traditional hard real-time mechanisms, namely rate-monotone (RM) scheduling and EDF scheduling. They extended the schedulability test of RM and EDF algorithms to incorporate CPU frequency scaling. Unlike our algorithm that applies frequency scaling to only the current task, they assumed a unified frequency scaling factor upon all tasks. In their most aggressive variant, the look-ahead technique is used to achieve extensive energy savings by deferring as much work as possible. However, the frequency value obtained in their algorithm is not always the lowest possible frequency for a single task, as shown in [4].

Some of the other aggressive real-time DVS schemes exploit early completion of task executions based on statistical information of the workload under dynamic scheduling [1] or static priority scheduling [5]. The algorithm proposed in [1] was based on early completion of tasks and idle time up to the next task's activation. The feedback scheme in our algorithm adapts even to dynamically changing execution demands, not just statistical information. We exploit both the idle time prior to the next task's activation and any idle slots up to the deadline of the task in the maximal schedule.

The idea of deriving a feasible dual-level DVS schedule from an ideal case was first proposed by Gruian [5, 6]. It combines off-line and on-line scheduling at both task level and task-set level. Stochastic data was used to derive energy-efficient schedules. Multiple frequency levels may be assigned to a single task. In our approach, we assign at most two different frequencies for each task, and the highest frequency is always assigned to the second subtask. Our algorithm also targets dynamic scheduling (EDF) while Gruian restricts his approach to fixed-priority static scheduling. Dual speed scheduling was also proposed in two other approaches. First, Zhang *et al.* switch the processor speed between high and low whenever non-preemption blocking occurs among tasks that share resources [23]. Second, Lee *et al.* assume an architecture model where only two physical speed levels exist [9]. Our approach considers a more general case where multiple frequency and voltages levels are chosen by subsequent jobs of the same task or even different tasks, although for a single job, only two speeds are used. Last-chance scheduling without energy considerations goes back at least to Chetto and Chetto [3]. We ap-

ply this philosophy in a DVS context. We develop a novel variant based on task splitting with exactly two parts. Such a dual-subtask approach aggressively reduces power consumption if the first subtask is fully utilized while the second subtask never executes. Our feedback approach triggers this behavior, which is superior to Gruian's step-wise increase of frequencies using stochastic approach.

8. Conclusion

This paper presents a novel scheduling approach combining DVS with feedback control schemes, which extends EDF in a most aggressive manner. The technique relies strictly on operating system support to implement both the real-time scheduler and the feedback controller. Our contributions include techniques for preemption handling and feedback control for hard real-time systems with dynamically fluctuating workload characteristics, *i.e.*, when execution times of a task's jobs vary significantly. A feedback scheme is applied on the system with different workloads. The online complexity of our algorithm is O(n) for n tasks. The feedback technique makes the system capable to select the right frequency and voltage settings, so that energy consumption is significantly reduced. It also guarantees the timing constraints of hard real-time tasks, so that no tasks ever miss their deadlines. For predictable fluctuating execution time patterns, our feedback DVS scheme is able to adapt to dynamically fluctuating workloads better than previous work and saves up to 29% additional energy. The scheme is not sensitive to any particular workload characteristics, *i.e.*, the execution time patterns, and is capable of scaling for task sets with different number of tasks. Directions for future work include the assessment of the algorithm under real embedded environments and the investigation of the impact that different PID parameters have together with systematic approaches in parameter tuning.

References

[1] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and agressive scheduling techniques for power-aware real-time systems. In *IEEE Real-Time Systems Symposium*, Dec. 2001.

[2] A. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power cmos digital design. In *IEEE Journal of Solid-State Circuits, Vol. 27, pp. 473-484.*, April, 1992.

[3] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, Oct. 1989.

[4] A. Dudani, F. Mueller, and Y. Zhu. Energy-conserving feedback edf scheduling for embedded systems with real-time constraints. In *ACM SIGPLAN Joint Conference Languages, Compilers, and Tools for Embedded Systems (LCTES'02) and Software and Compilers for Embedded Systems (SCOPES'02)*, pages 213–222, June 2002.

[5] F. Gruian. Hard real-time scheduling for low energy using stochastic data and dvs processors. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'01*, Aug 2001.

[6] F. Gruian and Kuchcinski. Lenes: task scheduling for low-energy systems using variable voltage processors. In *Proceedings of ASP-DAC*, 2001.

[7] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the 1998 international symposium on Low power electronics and design*, pages 197–202. ACM Press, 1998.

[8] D. Kang, S. Crago, and J. Suh. A fast resource synthesis technique for energy-efficient real-time systems. In *IEEE Real-Time Systems Symposium*, Dec. 2002.

[9] Y.-H. Lee and C. M. Krishna. Voltage-clock scaling for low energy consumption in fixed-priority real-time systems. *Real-Time Syst.*, 24(3):303–317, 2003.

[10] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. of the Association for Computing Machinery*, 20(1):46–61, Jan. 1973.

[11] Y. Liu and A. K. Mok. An integrated approach for applying dynamic voltage scaling to hard real-time systems. In *Proceedings of the ninth IEEE Real-Time and Embedded Technology and Applications Symposium*, May 2003.

[12] C. Lu, J. Stankovic, G. Tao, and S. Son. Design and evaluation of a feedback control edf scheduling algorithm. In *IEEE Real-Time Systems Symposium*, Dec. 1999.

[13] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst.*, 23:85–126, 2002.

[14] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, pages 156–63, 2002.

[15] M. Mächtel and H. Rzehak. Measuring the Influence if Real-Time Operating Systems on Performance and Determinism. *Control Eng. Practice*, 4(10):1461–1469, 1996.

[16] R. Minerick, V. W. Freeh, and P. M. Kogge. Dynamic power management using feedback. In *Proceedings of Workshop on Compilers and Operating Systems for Low Power*, 2002.

[17] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Symposium on Operating Systems Principles*, 2001.

[18] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. *Technical report,Delft University of Technology*, 2000.

[19] S. Saewong and R. Rajkumar. Practical voltage-scaling for fixed-priority rt-systems. In *Proceedings of the ninth IEEE Real-Time and Embedded Technology and Applications Symposium*, May 2003.

[20] D. Shin, J. Kim, and S. Lee. Intra-task voltage scheduling for low-energy hard real-time applications. In *IEEE Design and Test of Computers*, March 2001.

[21] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Int'l Conf. on Computer-Aided Design*, 2000.

[22] J. Wegener and F. Mueller. A comparison of static analysis and evolutionary testing for the verification of timing constraints. *Real-Time Systems*, 21(3):241–268, Nov. 2001.

[23] F. Zhang and S. T. Chanson. Processor voltage scheduling for real-time tasks with non-preemptable sections. In *IEEE Real-Time Systems Symposium*, Dec. 2002.