Power-Conscious Scheduling for Real-Time Embedded Systems Design

Youngsoo Shin‡, Kiyoung Choi§, and Takayasu Sakurai‡
‡Center for Collaborative Research and Institute of Industrial Science,
University of Tokyo, Tokyo 106-8558, Japan
§School of Electrical Engineering and Computer Science,
Seoul National University, Seoul 151-742, Korea

Abstract

Power efficient design of real-time embedded systems based on programmable processors becomes more important as system functionality is increasingly realized through software. We address a power optimization method for real-time embedded applications on a variable speed processor. The method combines off-line and on-line components. The off-line component determines the lowest possible maximum processor speed while guaranteeing deadlines of all tasks. The on-line component dynamically varies the processor speed or bring a processor into a power-down mode to exploit execution time variations and idle intervals. Experimental results show that the proposed method obtains a significant power reduction across several kinds of applications.

Keywords – Real-time, scheduling, operating system, processor, low-power, CMOS

1 Introduction

Recently, power consumption has been a critical design constraint in the design of digital systems due to widely used portable systems such as cellular phones and PDAs, which require low power consumption with high speed and complex functionality. The design of such systems often involves reprogrammable processors such as microprocessors, microcontrollers, and DSPs in the form of off-the-shelf components or cores. Furthermore, an increasing amount of system functionality tends to be realized through software, which is leveraged by the high performance of modern processors. As a consequence, reduction of the power consumption of processors is important for the power-efficient design of such systems.

Recognizing the need to reduce the power consumption of processors, a number of methods have been proposed at the hardware and software levels. The methods at the software level can be loosely classified into power-aware compilation techniques [1], [2], [3] and Operating System (OS) directed power management techniques. The importance of latter approach increases recently because OS is recognized to play a central role in power management of overall system components.

Broadly, there are two kinds of methods to reduce power consumption of processors in OS level. The first is to bring a processor into a power-down mode, where only certain parts of the processor such as the clock generation and timer circuits are kept running. Another method is to use a *variable speed processor* (VSP), which can change its speed by varying the clock frequency along with the supply voltage when the required performance on the processor is lower than the maximum.

Reducing power consumption of processors is fundamentally equivalent to exploiting *idle intervals* of processors. Thus, we should first identify sources of idle intervals to efficiently reduce the power dissipated by processors. Our approach is strongly motivated by the fact that there are several kinds of sources for idle intervals in a schedule of a real-time task set. Especially in case of a *priority-based preemptive scheduling*, which is one of the most widely

used scheduling methods for real-time systems, we identify three kinds of sources. The first one occurs when a system is not tightly designed for a given processor, meaning that there is room for design change or improvement; introducing some more tasks, replacing certain tasks with their version-ups, using other processors with lower performance, and so on. Even if the system is tightly-designed, there are still idle intervals in case of fixed-priority scheduling which are strongly dependent upon the relative values of the periods of the tasks comprising the system; the second source of idle intervals. The third one is from run-time variation of execution time of each task, that is, the execution time of each task in run-time is not constant due to data-dependent computation, over-estimation of worst-case execution time, and so on. Each of these will be elaborated in more detail in section 3.

To exploit these idle intervals for low-power, we propose a power optimization method for real-time embedded applications on a *VSP with a power-down mode*. The proposed method consists of two components: *off-line component based on real-time analysis of a task set* that exploits the first source of idle intervals and *on-line component based on priority-based real-time scheduling* that exploits both the second and the third sources. Specifically, for a given real-time task set, we first compute the lowest possible maximum processor speed such that at least one of deadlines are violated if the processor is running below that speed. With the maximum speed of the VSP set to the computed value, we then dynamically varies the speed of the VSP or bring the VSP into a power-down mode to exploit execution time variation of each task and idle intervals present in the schedule. Note that all kinds of idle intervals can be exploited by on-line component only [4]. However, we show that combined off-line and on-line components bring about more power-saving.

The remainder of the paper is organized as follows. In the next section, we review related work, which focuses on the reduction of power consumption of processors. In section 3, we present the system model for power optimization, off-line component, and on-line

component. In section 4, experimental results are presented to evaluate the proposed method. Finally, a conclusion follows in section 5.

2 Related Work

2.1 Power-Down Modes

In most embedded systems, a processor often waits for some events from its environment, wasting its power. To reduce the waste, modern processors are often equipped with various levels of power modes. In the case of the PowerPC 603 processor [5], there are four power modes (Full On, Doze, Nap, and Sleep), which can be selected by setting the appropriate control bits in a register. Each mode is associated with a level of power saving and delay overhead. In the conventional approach employed in most portable appliances, a processor enters power-down mode after it stays in an idle state for a predefined time interval. Since the processor still wastes its energy while in the idle state, this approach fails to obtain a large reduction in energy when the idle interval occurs frequently but its length is short. In [6], [7], the length of the next idle period is predicted based on a history of processor usage. The predicted value becomes the metric to determine whether it is beneficial to enter power-down modes or not. This method focuses on event-driven applications such as user-interfaces where the latency caused by the mismatch between the predicted value and the actual value can be tolerated. However, an exact value or a lower bound are needed instead of a predicted value for the next idle period when the power-down modes are to be applied in a hard real-time system.

2.2 Scheduling on a Variable Speed Processor

It is a well-known fact that power consumption in CMOS circuits can be decomposed into two parts: static and dynamic. The dynamic power consumption, which is a dominant

factor, is described by

$$P_{dynamic} = a \cdot f \cdot C_L \cdot V_{dd}^2, \tag{1}$$

where a is the expected number of transitions per cycle, called switching activity, f is the clock frequency, C_L is the average load capacitance, and V_{dd} is the supply voltage. The reduction of V_{dd} is the most effective way to reduce the power consumption as expected in (1). However, reducing V_{dd} leads to an increase in circuit delay, denoted by t_d, which can be approximated by

$$t_d = k\frac{V_{dd}}{(V_{dd} - V_t)^\alpha}, \tag{2}$$

where k is a constant, V_t is the threshold voltage, and α is a constant satisfying $1 < \alpha < 2$. A digital system designed with a fixed supply voltage (V_{dd}) works at a fixed speed and then can be made idle if the computational demand is less than the maximum. If the supply voltage is lowered dynamically to the lowest value satisfying the required speed constraint of the system as exhibited by (2), less power would be consumed. This kind of *adaptive* scaling of the supply voltage was exploited in self-timed circuits [8] and DSP systems [9]. Recently, the same mechanism was adapted to a microprocessor architecture [10], [11]. For example, [11] reports a processor based on the ARM microprocessor core, where the operating voltage is set by a feedback loop which compares the current and target frequencies.

A scheduling method to reduce power consumption of a VSP was first proposed in [12] and was later extended in [13]. The basic method is that short-term processor usage is predicted from a history of processor utilization. From the predicted value, the speed of the processor is set to the appropriate value. Because latency exists when the prediction fails, these methods cannot be applied to real-time systems.

Static scheduling methods for real-time systems were proposed in [14], [15], [16]. The underlying model of their approaches is a set of tasks with a single period. When periods of tasks are different from each other, which is the conventional model employed in real-time system design, we can transform a problem by taking the LCM (Least Common Multiple)

of tasks' periods as a single period and treating each instance of the same task occurring within the LCM a a different task. This can cause a practical problem because we require excessively large memory space to save a statically computed schedule, whereas the size of memory is one of the design constraints in a typical embedded system. Furthermore, LCM becomes excessively large when periods of tasks are mutually prime. Another problem is that a schedule is computed based on the assumption that a fixed amount of execution time is required for each task. As a result, the full potential of power saving cannot be obtained when variations of execution time exist.

A dynamic scheduling method, called Average Rate Heuristic (AVR), was also proposed in [14] with the same model as in the static version. Associated with each task is its *average-rate requirement*, which is defined by dividing its required number of cycles by its time frame (deadline − arrival time). At any time t, the AVR sets the speed of a processor to the sum of average-rate requirements of tasks whose time frame includes t. Among available tasks, AVR resorts to the earliest deadline policy [17] to choose a task. Because average-rate requirements are computed statically with fixed numbers of execution cycles, the same problem occurs when variations of execution time exist.

3 Power Optimization Method

3.1 System Model

For a processor model, we assume a VSP similar to [11]. The *reference clock frequency*, denoted as f_{ref}, and the *reference supply voltage*, denoted as V_{ref}, of the VSP is 100 MHz and 3.3 V, respectively. The clock frequency can be varied from 100 MHz down to 8 MHz with a step size of 1 MHz. The supply voltage is 3.3 V for 100 MHz clock and, for lower clock frequency, follows (2). We assume that there is only one power-down mode available. The average power consumed by the processor when it is in power-down mode is 5% of the fully active mode and it takes 10 clock cycles to return from the power-down mode

to the fully active mode. The processor model described above is only for the purpose of simulation which is to be presented in section 4. Therefore, our method can be applied for other processor models, for example of a processor with only two speed levels [18], though the result of power saving may be different.

In a typical real-time embedded application, there are many periodic tasks that share hardware resources. To ensure that each task satisfies its timing constraint, the execution of tasks should be coordinated in a controlled manner. This is often done via *priority-based preemptive scheduling algorithm*. There are two kinds of algorithms based on priority assignment: fixed-priority (or static-priority) algorithms such as rate-monotonic (RMS) [17] and deadline-monotonic (DMS) [19] and dynamic-priority algorithms such as earliest deadline first (EDF) [17]. A priority-based scheduling is quite simple to implement in most kernels, and it typically requires little if any extra hardware support. Also, there are many analytical methods to check the schedulability of the system.

The real-time embedded application is modeled as a set of tasks, $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, which are numbered in order of decreasing priority in case of fixed-priority scheduling (FPS). The parameters of τ_i include its period (the minimum inter-arrival time between successive requests in case of a sporadic task) T_i, deadline D_i, and worst case execution time (WCET) C_i. A task set is called *feasible* if deadline of each task is satisfied at all times. Note that C_i is measured or estimated [20] when the VSP is running in reference speed (f_{ref} and V_{ref}).

To minimize energy consumption while guaranteeing the feasibility of a task set, we first determine the lowest possible speed such that the task set is feasible if the VSP is running in that speed entirely, and will be infeasible if running in lower speed. This can be done with off-line method as illustrated in the next subsection. Note that worst-case scenario (all tasks execute in WCET at all times) must be assumed in off-line method. However, during operation of the system, the execution time of each task frequently deviates from its WCET, sometimes by a large amount. In many cases, the possibility of a task running at

its WCET is usually very low. Furthermore, the complex architecture of modern processors (pipeline, instruction cache, data cache, and so on) makes the static estimation of WCET difficult thereby resulting in over-estimation of WCET. As examples of this variation in execution time, Figure 1 shows the ratio between the best-case execution time (BCET) and WCET obtained from [21] for a number of applications.

Figure 1 goes here

These execution time variation cannot be exploited with off-line method alone. Furthermore, with fixed-priority scheduling, there are still idle intervals remained even if the VSP is running in the lowest possible speed entirely. To exploit these execution time variation and idle intervals, we use an on-line method, where we dynamically vary the speed of the VSP or bring the VSP into a power-down mode according to the status of the task set.

Example 1 Consider the three tasks given in Table 1. Rate monotonic priority assignment is a natural choice because periods (T_i) are equal to deadlines (D_i). Priorities are assigned in row order as shown in the fifth column of the table (lower value means higher priority). Assume all tasks are released simultaneously at time 0. A typical schedule, which assumes that tasks run at their WCETs (C_i), is shown in Figure 2(a). If the speed of the processor is lowered by half or if the processor with half performance is used meaning that C_i is doubled, the schedule becomes as shown in Figure 2(b). It is noted that the task set scheduled in Figure 2(b) just meets its feasibility. For example, if τ_2 were to take a little longer to complete, τ_3 would miss its deadline at time 100. Even though the system is tightly constructed, there are still idle intervals, as can be seen in Figure 2(b). When some task instances are completed earlier than their WCETs, there are more idle intervals as shown in Figure 2(c). These idle intervals are sources of power reduction by on-line method. □

Table 1 goes here

Figure 2 goes here

3.2 Computation of Maximum Speed

For a given task set, in order to determine the lowest possible maximum processor speed (thus the *lowest possible maximum clock frequency*, denoted as f_{max}, and the *lowest possible maximum supply voltage*, denoted as V_{max}), the analysis of the schedulability of the task set is required. We first present the approach for fixed-priority algorithms and then the approach for dynamic-priority algorithms.

The schedulability analysis for fixed-priority scheduling is based on the *critical instant theorem* [17] which says that if a task meets its deadline whenever the task is requested simultaneously with requests for all higher priority tasks, then the deadline will always be met for all task phasings. This implies that it is needed to perform the analysis from time 0 up to LCM of all task periods under the assumption that all tasks are requested simultaneously at time 0. This again requires the analysis to be performed in the continuous time interval. Lehoczky et al. [22] shows that the analysis is actually needed only at discrete time points instead of continuous time interval. The set of time points, called *scheduling points*, for task τ_i is defined by

$$S_i = \{kT_j | j = 1, 2, \ldots, i; k = 1, \ldots, \lfloor \frac{T_i}{T_j} \rfloor\}, \tag{3}$$

when $T_i = D_i$. If D_i is different from T_i, (3) can be modified as

$$S_i' = (S_i - \{t | t \in S_i, t > D_i\}) \cup \{D_i\}. \tag{4}$$

τ_i can be scheduled without violating its deadline, if there *exist* one or more scheduling points $t \in S_i$, which satisfy

$$\sum_{k=1}^{i} C_k \lceil \frac{t}{T_k} \rceil \leq t. \tag{5}$$

Note that the left hand side of the inequality represents the cumulative demands on the processor imposed by $\tau_1, \tau_2, \ldots, \tau_i$.

Now, it is assumed that elements of S_i are sorted in ascending order. $S_{i,j}$ is defined as the jth element of S_i, that is, jth scheduling point of τ_i. Thus, for each scheduling point

$S_{i,j}$, τ_i *just* meets its scheduling point if it satisfies

$$\sum_{k=1}^{i} \frac{1}{\eta_{i,j}} C_k \lceil \frac{S_{i,j}}{T_k} \rceil = S_{i,j}, \tag{6}$$

where $\eta_{i,j}$ is *speed scaling factor* for τ_i at $S_{i,j}$. For example, $\eta_{i,j} = \frac{1}{2}$ means that the speed of the processor is reduced by half thus execution times of tasks are doubled. Solving for $\eta_{i,j}$ gives

$$\eta_{i,j} = \frac{\sum_{k=1}^{i} C_k \lceil \frac{S_{i,j}}{T_k} \rceil}{S_{i,j}}. \tag{7}$$

Because τ_i is schedulable if it completes its execution before or at *any* scheduling points and the minimum possible speed scaling factor is needed for τ_i for minimum power consumption, *speed scaling factor* for τ_i, denoted by η_i, is given by

$$\eta_i = \min_j \eta_{i,j}. \tag{8}$$

In order to get a feasible task set, all tasks are required to be schedulable. Thus, *speed scaling factor* for the task set, denoted by η, is given by

$$\eta = \max_i \eta_i. \tag{9}$$

Note that if η is larger than 1, the original task set is already infeasible meaning that it cannot be scheduled with fixed-priority scheduling even with f_{ref} and V_{ref}. Hence, f_{max} (correspondingly V_{max}) is obtained by

$$f_{max} = \eta f_{ref}. \tag{10}$$

In practice, we should take $\lceil \eta f_{ref} \rceil$ for f_{max} because discrete levels of frequencies are assumed. We also need clamping operation so that f_{max} falls between 8 MHz and 100 MHz.

For dynamic-priority scheduling, especially for EDF scheduling with $D_i = T_i$, a task set is feasible if and only if the *processor utilization* is less than or equal to 1 [17]. Thus, η is straightforward to compute because it is equal to the processor utilization, given by

$$\eta = \sum_{\forall \tau_i \in \tau} \frac{C_i}{T_i}. \tag{11}$$

It should be noted that there are no idle intervals meaning that the power consumption of the processor is minimized if the processor is running entirely in the speed obtained with (11) provided that fractional value is possible for f_{max}, and each task always execute in constant execution time of WCET. When $D_i < T_i$, we can use D_i instead of T_i in the denominator of the right hand side of equation (11), called *total density* in this case instead of processor utilization. Note that, however, η obtained in this way is conservative in that the task set is feasible with EDF if the total density is equal to or less than 1 but the opposite does not hold.

Example 2 Consider again the three tasks given in Table 1 with rate monotonic priority assignment. From equation (3), the set of scheduling points for each task is given by

$$S_1 = \{T_1\},\ S_2 = \{T_1, T_2\},\ S_3 = \{T_1, T_2, T_3\}.$$

We compute η using equations (7), (8), and (9), which yields

$$\begin{aligned}
\eta_1 &= \min(\frac{C_1}{T_1}) = 0.1, \\
\eta_2 &= \min(\frac{C_1 + C_2}{T_1}, \frac{2C_1 + C_2}{T_2}) = 0.25, \\
\eta_3 &= \min(\frac{C_1 + C_2 + C_3}{T_1}, \frac{2C_1 + C_2 + C_3}{T_2}, \frac{2C_1 + 2C_2 + C_3}{T_3}) \\
&= 0.5, \\
\eta &= \max(\eta_1, \eta_2, \eta_3) = 0.5.
\end{aligned}$$

Thus, we can reduce the maximum speed by as much as half or can use the processor with half performance (see Figure 2(b)). □

3.3 Low-Power Priority-Based Real-Time Scheduling

Even if the processor is running in the speed obtained with the method of the previous subsection, there are still idle intervals that arise from two sources (see Example 1). The first source is idle intervals *inherently present in fixed-priority scheduling* (thus it is not

the case with EDF) because of different period of each task. The second one is *run-time variation of execution time* of each task. In more specific, although constant execution time of WCET should be assumed in the method of the previous subsection, the execution time of each task in run-time is not constant due to data-dependent computation, over-estimation of WCET, and so on. To exploit these idle intervals, we propose a power-efficient version of priority-based real-time scheduling method, which we call `lpps` for brevity.

The basic mechanism of the proposed scheduling algorithm is based on the implementation model in [23], [24]. The scheduler maintains two queues, one called *run queue* and the other called *delay queue*. The run queue holds tasks that are waiting to run and the tasks in the queue are ordered by priority. The task that is running on the processor is called the *active task*. The delay queue holds tasks that have already run in their period and are waiting for their next period to start again. They are ordered by the time at which their release is due. When the scheduler is invoked, it searches the delay queue to see if any tasks should be moved to the run queue. If some of the tasks in the delay queue are moved to the run queue, the scheduler compares the active task to the task at the head of the run queue. If the priority of the active task is lower, a context switch occurs.

Because most information about the tasks is available through the queues and `lpps` depends on this information, the proposed scheduler can be implemented with a slight modification of the conventional scheduler. Figure 3 shows the pseudo code of the `lpps` scheduling algorithm. The code lines between L5 and L11 (except L9 to be explained shortly) conform to the behavior of the conventional scheduler. `lpps` works when the run queue is empty (L12). This is further divided into two cases: one where all tasks have completed their executions in each of their periods and are waiting for their next arrival times while residing in the delay queue (L13) and the other where all tasks *except* the active task have completed their execution (L16). In the first case, we can bring the processor into a power-down mode because there are no tasks that need it. Furthermore,

we know how long the processor will be idle because the task at the head of the delay queue is the first one that will require the processor (recall that the delay queue is ordered by the tasks' release times). This is the key ingredient of `lpps`. Thus, we set a timer to expire at the next release time of the task at the head of the delay queue and then put the processor into the power-down mode. Because, there is a delay overhead to wake up from the power-down mode, the timer actually should be set to expire earlier by that amount of delay (L14).

Figure 3 goes here

In the second case, we can control the speed of the processor because there is just one task (the active task) to execute and the processor will be available solely for that task until the minimum of the deadline of the active task and the release time of the task at the head of the delay queue. The amount of time that will be needed by the active task equals its WCET less its already executed time. This can be obtained when a task is preempted because of a request for a task with higher priority during its execution (L8). When this occurs, we get the executed time of the task from the timer (L9) that is based on an external clock, which is independent of the variation of processor's speed. Note that we assume the execution of the whole task takes its WCET because at the time of scheduling we have no information whether it will take less than WCET or not. When the active task completes its execution, the scheduler gets the control and increases the speed of the processor to the maximum to prepare for the next arrival of tasks (L1 through L4). This involves a delay for raising the supply voltage and subsequently the clock frequency. Thus, the active task actually should complete its execution earlier by an amount equal to this delay. Considering all these factors, we obtain the ratio of the processor speed needed for the active task to the full speed (L17). From the computed ratio, we find an appropriate clock frequency (L18). In practice, only discrete levels of frequencies are available, and among them we should select a frequency larger than or

equal to the computed one to guarantee the timing constraints. All these processes are illustrated in the following example.

Example 3 Consider Figure 2(b), that is, the same task set in Table 1 with C_i doubled. At time 160 when a request for τ_2 arrives, the status of queues and the information associated with each task are shown in Figure 4(a). For simplicity of illustration, assume that the delay required to wake up from the power-down mode and that required to change the speed of a processor are all 0. Because the run queue is empty with the active task of τ_2, the scheduler computes the desired ratio of speed that yields $\frac{20-0}{200-160} = 0.5$ (see L17 of Fig. 3). Thus, we can slow down the processor by half. Now, assume that the instance of τ_2 started at time 160 executes at the lowered speed, but completes its execution at time 180 instead of 200, meaning that it executes in half its WCET. At this time, the status of queues becomes that of Fig. 4(b). Because all tasks reside in the delay queue, the scheduler brings the processor into a power-down mode (see L14 and L15 of Fig. 3) with the timer set to the next arrival time of τ_1 (200). □

Figure 4 goes here

4 Experimental Results

To evaluate the proposed method, we perform simulations with several examples and compare the average power consumption with the proposed method against that with the conventional priority-based scheduling. In the conventional priority-based scheduling, the processor is assumed to execute NOP (no operation) instructions, when it is not being occupied by any tasks. The average power consumed by a NOP instruction is assumed to be 20% of that consumed by a typical instruction [25]. We also compare the result with that of [4].

We collect four applications for experiments: an `avionics` task set [26], an `ins` [24], a `flight_control` [27], and a `cnc` machine controller [28]. The first three examples are

mission critical applications and the last one is a digital controller for a CNC machine, which is an automatic machining tool that is used to produce user-defined workpieces. For each task comprising an application, three timing parameters (T_i, D_i, and C_i) are given. Because the statistics of the actual execution times of instances of the tasks are not available, it is assumed that the execution time of each instance of a task is drawn from a random Gaussian distribution with mean of $m = \frac{\text{BCET+WCET}}{2}$ and standard deviation of $\sigma = \frac{\text{WCET−BCET}}{6}$, where WCET= C_i. Then, the BCET is varied from 10% to 100% of the WCET for each task. This ensures that almost all generated values fall between BCET and WCET because the probability that a random variable x takes on a value in the interval $[m - 3\sigma, m + 3\sigma]$ of a random Gaussian distribution is approximately 99.7%. If we set WCET to be equal to $m + 3\sigma$ and solve for σ with the help of equation for m, we get equation for σ. After the generation of execution time, we apply clamping operation so that the generated value does not exceed WCET.

First, f_{max} and V_{max} are obtained for each application using equations (9) and (11), which are summarized in Table 2. Clearly, they are smaller with EDF than with FPS, because EDF sets the lower bound for f_{max} and V_{max}. In case of `ins`, f_{max} with FPS is very close to that with EDF meaning that very high processor utilization is possible even with FPS. This is because most periods of tasks in `ins` is *harmonic*, that is, period of each task is divisible with each other.

Table 2 goes here

Next, with the maximum speed of the VSP set to the corresponding value shown in Table 2, each task set is simulated with `lpps`. The results are shown in Figure 5, where `lpps/RMS` indicates that RMS is used for basic scheduling algorithm of `lpps` and `lpps/EDF` similarly for EDF. The vertical axis indicates average power reduction with each method compared to the conventional priority-based scheduling (see Figure 2). Note that the power gain from off-line method is independent on the horizontal axis because worst-case scenario

is assumed in that method. The power gain from on-line method increases as the BCET gets smaller (variation of execution time gets larger). This is because the chances both for dynamically varying the speed of the VSP and for bringing the VSP into a power-down mode increases as the variation of execution times increases. The largest gain is obtained in `cnc`. This can be understood from Table 2 because `cnc` can be operated in the lowest speed, meaning that its processor utilization in reference speed is the lowest. Compared to on-line method alone, we can obtain more power saving with combined off-line and on-line methods.

Figure 5 goes here

5 Conclusion

In this paper, we propose a power optimization method for a real-time embedded application on a variable speed processor. The method consists of two components. First, we determine the lowest possible processor speed such that the task set is feasible if the processor is running in that speed entirely, and will be infeasible if running in lower speed. Then, to exploit execution time variation and idle intervals, we relies on low-power priority-based real-time scheduling, which dynamically varies the speed of the VSP or brings the processor into a power-down mode. Experimental results show that the proposed method obtains a significant power reduction across several applications.

References

[1] C. L. Su, C. Y. Tsui, and A. M. Despain, "Low power architecture design and compilation technique for high-performance processors," in *Proc. IEEE COMPCON*, Feb. 1994, pp. 209–214.

[2] M. T.-C. Lee, V. Tiwari, S. Malik, and M. Fujita, "Power analysis and minimization techniques for embedded DSP software," *IEEE Trans. on VLSI Systems*, vol. 5, no. 1, pp. 123–135, Mar. 1997.

[3] H. Tomiyama, T. Ishihara, A. Inoue, and H. Yasuura, "Instruction scheduling for power reduction in processor-based system design," in *Proc. Design, Automat. and Test in Europe Conference and Exhibition*, Feb. 1998, pp. 855–860.

[4] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proc. Design Automat. Conf.*, June 1999, pp. 134–139.

[5] B. W. Suessmith and G. Paap III, "PowerPC 603 microprocessor power management," *Communications of the ACM*, vol. 37, no. 6, pp. 43–46, June 1994.

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

[7] C. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. Int'l Conf. on Computer Aided Design*, Nov. 1997, pp. 28–32.

[8] L. S. Nielsen, C. Niessen, J. Sparso, and K. van Berkel, "Low-power operation using self-timed circuits and adaptive scaling of the supply voltage," *IEEE Trans. on VLSI Systems*, vol. 2, no. 4, pp. 391–397, Dec. 1994.

[9] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: an approach for energy efficient computing," in *Proc. Int'l Symposium on Low Power Electronics and Design*, Aug. 1996, pp. 347–352.

[10] T. Ishihara and H. Yasuura, "Power-Pro: Programmable power management architecture," in *Proc. Asia South Pacific Design Automat. Conf.*, Feb. 1998.

[11] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," in *Proc. Int'l Symposium on Low Power Electronics and Design*, Aug. 1998, pp. 76–81.

[12] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. USENIX Symposium on Operating Systems Design and Implementation*, 1994, pp. 13–23.

[13] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithms for dynamic speed-setting of a low-power CPU," in *Proc. ACM Int'l Conf. on Mobile Computing and Networking*, Nov. 1995, pp. 13–25.

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

[15] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable voltage core-based systems," in *Proc. Design Automat. Conf.*, June 1998, pp. 176–181.

[16] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proc. Int'l Symposium on Low Power Electronics and Design*, Aug. 1998, pp. 197–202.

[17] C. L. Liu and James W. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[18] Y. Lee and C. Krishna, "Voltage-clock scaling for low energy consumption in real-time embedded systems," in *Proc. Int'l Workshop on Real-Time Computing Systems and Applications*, 1999.

[19] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," in *Proc. IEEE Workshop on Real-Time Operating Systems and Software*, May 1991, pp. 133–137.

[20] S. Lim, Y. Bae, G. Jang, B. Rhee, S. Min, C. Park, H. Shin, K. Park, and C. Kim, "An accurate worst case timing analysis for RISC processors," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1994, pp. 97–108.

[21] R. Ernst and W. Ye, "Embedded program timing analysis based on path clustering and architecture classification," in *Proc. Int'l Conf. on Computer Aided Design*, Nov. 1997, pp. 598–604.

[22] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1989, pp. 166–171.

[23] D. Katcher, H. Arakawa, and J. Strosnider, "Engineering and analysis of fixed priority schedulers," *IEEE Trans. on Software Eng.*, vol. 19, no. 9, pp. 920–934, Sept. 1993.

[24] A. Burns, K. Tindell, and A. Wellings, "Effective analysis for engineering real-time fixed priority schedulers," *IEEE Trans. on Software Eng.*, vol. 21, no. 5, pp. 475–480, May 1995.

[25] T. Burd and R. Brodersen, "Processor design for portable systems," *Journal of VLSI Signal Processing*, vol. 13, no. 2/3, pp. 203–222, Aug. 1996.

[26] C. Locke, D. Vogel, and T. Mesler, "Building a predictable avionics platform in Ada: A case study," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1991.

[27] J. Liu, J. Redondo, Z. Deng, T. Tia, R. Bettati, A. Silberman, M. Storch, R. Ha, and W. Shih, "PERTS: A prototyping environment for real-time systems," Tech. Rep. UIUCDCS-R-93-1802, University of Illinois, 1993.

[28] 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 Real-Time Systems Symposium*, Dec. 1996.

List of Figures

1 The ratio between BCET and WCET for a number of applications. 22

2 A schedule for the example task set. (a) When tasks always run at their WCETs. (b) When tasks always run at their WCETs on a processor with the speed lowered by half. (c) When the execution times of some task instances are smaller than their WCETs. 23

3 Pseudo code of the `lpps` scheduler. 24

4 The status of queues and the information associated with each task (a) at time 160 and (b) at time 180. 25

5 Simulation results of (a) `avionics`, (b) `ins`, (c) `flight_control`, and (d) `cnc`. 26

List of Tables

1 An example task set 27

2 Maximum frequency and voltage computed for each application. $f_{ref} = 100$ MHz and $V_{ref} = 3.3$ V. 28

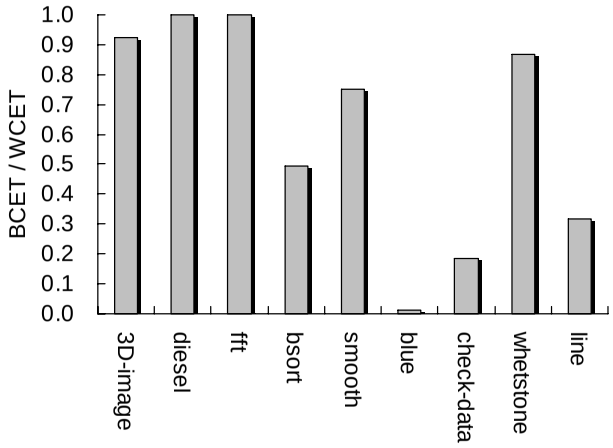

Figure 1. The ratio between BCET and WCET for a number of applications.

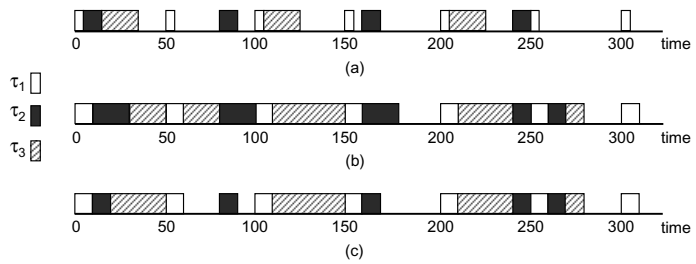

Figure 2. A schedule for the example task set. (a) When tasks always run at their WCETs. (b) When tasks always run at their WCETs on a processor with the speed lowered by half. (c) When the execution times of some task instances are smaller than their WCETs.

```
L1:   if current_frequency < maximum_frequency then
L2:       increase the clock frequency and the supply voltage to the maximum value;
L3:       exit;
L4:   end if
L5:   while delay_queue.head.release_time ≤ current_time do
L6:       move delay_queue.head to the run_queue;
L7:   end do
L8:   if run_queue.head.priority > active_task.priority then
L9:       set the active_task.executed_time;
L10:      context switch;
L11:  end if
L12:  if run_queue is empty then
L13:      if active_task is null then
L14:          set timer to (delay_queue.head.release_time − wakeup_delay);
L15:          enter power-down mode;
L16:      else
L17:          speed_ratio = Compute_speed_ratio();
L18:          find a minimum allowable clock frequency ≥ speed_ratio · max_frequency;
L19:          adjust the clock frequency along with the supply voltage;
L20:      end if
L21:  end if
```

Figure 3. Pseudo code of the `lpps` scheduler.

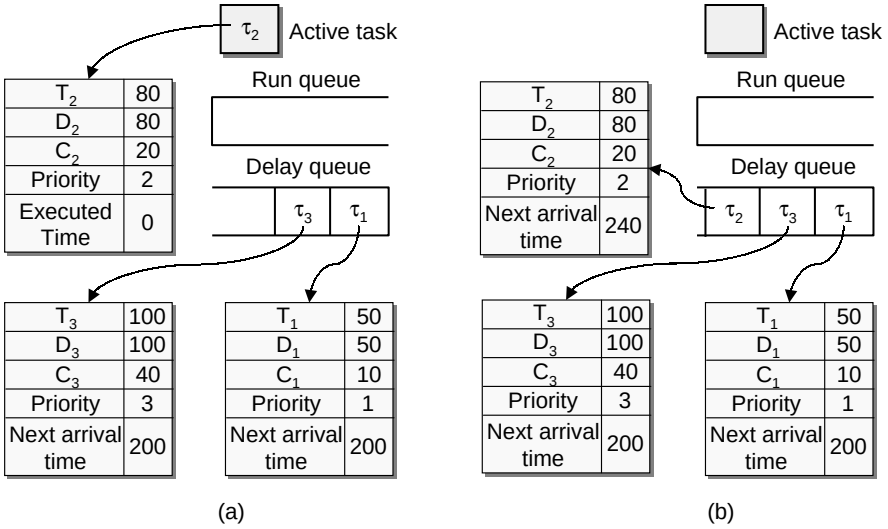

Figure 4. The status of queues and the information associated with each task (a) at time 160 and (b) at time 180.

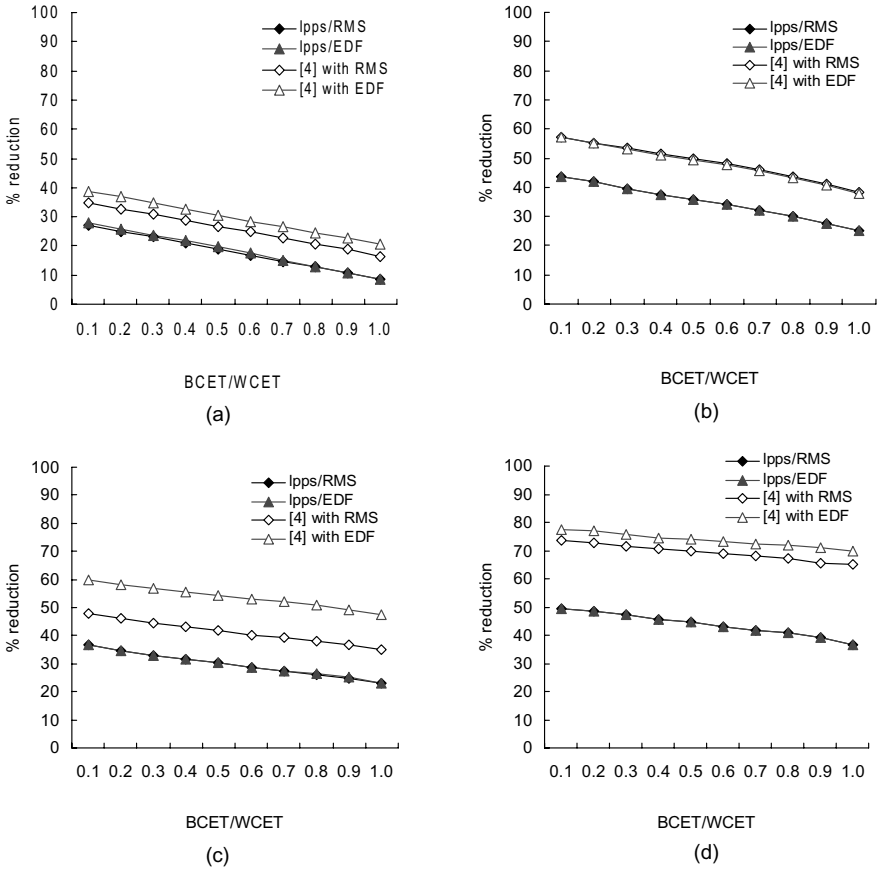

Figure 5. Simulation results of (a) `avionics`, (b) `ins`, (c) `flight_control`, and (d) `cnc`.

Table 1. An example task set

	T_i	D_i	C_i	Priority
τ_1	50	50	5	1
τ_2	80	80	10	2
τ_3	100	100	20	3

Table 2. Maximum frequency and voltage computed for each application. $f_{ref} = 100$ MHz and $V_{ref} = 3.3$ V.

	FPS		EDF	
	f_{max}	V_{max}	f_{max}	V_{max}
`avionics`	91 MHz	3.1 V	86 MHz	3.0 V
`ins`	75 MHz	2.7 V	74 MHz	2.7 V
`flight_control`	84 MHz	2.9 V	68 MHz	2.5 V
`cnc`	54 MHz	2.2 V	49 MHz	2.0 V

Biographies

Youngsoo Shin received the B.S., M.S., and Ph.D. degrees in electronics engineering from Seoul National University, Seoul, Korea, in 1994, 1996, and 2000, respectively. In 2000 he joined the Institute of Industrial Science, University of Tokyo, Tokyo, Japan, as a research associate. His research interests include various aspects of computer-aided design of VLSI circuits.

Kiyoung Choi received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1978 and the M.S. degree in electrical and electronics engineering from Korea Advanced Institute of Science and Technology, Tajon, Korea, in 1980. He received his Ph.D. degree in electrical engineering from Stanford University, USA, in 1989. From 1978 to 1983, he was with GoldStar Inc., Korea and from 1989 to 1991, he was with Cadence Design Systems, Inc., USA. He is currently an associate professor of the School of Electrical Engineering and Computer Science, Seoul National University, Seoul, Korea. His primary interests are in VLSI design and various aspects of computer-aided design.

Takayasu Sakurai received the B.S., M.S., and Ph.D. degrees in electronic engineering from University of Tokyo, Tokyo, Japan, in 1976, 1978, and 1981, respectively. In 1981 he joined Toshiba Corp., Japan. From 1988 to 1990, he was a visting researcher at University of California, Berkeley, doing research in the field of VLSI CAD. From 1990, back in Toshiba, he managed media processor and MPEG2 LSI designs. From 1996, he has been a professor at the Institute of Industrial Science, University of Tokyo, Tokyo, Japan, working on low-power and high-performance system LSI designs.