Power Optimization of Real-Time Embedded Systems on Variable Speed Processors

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. This paper presents 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 according to the status of task set in order 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.

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, especially in operating system (OS) level, is important for the power-efficient design of such systems.

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. In [1], 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. 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. A scheduling method for event-driven applications to reduce power consumption of a VSP was proposed in [2]. There are also several scheduling methods for real-time systems [3]. Because a fixed amount of execution time is assumed for these methods, the full potential of power saving cannot be obtained when variations of execution time exist.

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 2.

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 section 2, we present the system model for power optimization, off-line component, and on-line component. In section 3, experimental results are presented to evaluate the proposed method. Finally, a conclusion follows in section 4.

2 Power Optimization Method

2.1 System Model

For a processor model, we assume a VSP similar to [5]. 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

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

3.3 V for 100 MHz clock and, for lower clock frequency, follows

$$t_d = k\frac{V_{dd}}{(V_{dd}-V_t)^\alpha}, \quad (1)$$

where t_d is the circuit delay, k is a constant, α is a constant satisfying $1 < \alpha < 2$, and V_t is the threshold voltage. 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 3. Therefore, our method can be applied for other processor models, for example of a processor with only two speed levels, though the result of power saving may be different.

The real-time embedded application is scheduled according to *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) [6] and deadline-monotonic (DMS) [7] and dynamic-priority algorithms such as earliest deadline first (EDF) [6]. 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 when the VSP is running in maximum 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. 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. 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 1(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 1(b). It is noted that the task set scheduled in Figure 1(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 the figure. When some task instances are completed earlier than their WCETs, there are more idle intervals as shown in Figure 1(c). □

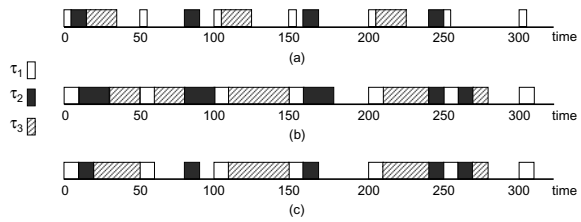

Figure 1. A schedule for the example task set. (a) When tasks always run at their WCETs. (b) When tasks always run at their WCETs on a processor with the speed lowered by half. (c) When the execution times of some task instances are smaller than their WCETs.

2.2 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* [6] 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. Lehoczky et al. [8] shows that the analysis is needed only at discrete time points, called *scheduling points*. The set of time points for task τ_i is defined by

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

when $T_i = D_i$. If D_i is different from T_i, (2) can be modified as

$$S_i' = (S_i - \{t | t \in S_i, t > D_i\}) \cup \{D_i\}. \quad (3)$$

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

$$\sum_{k=1}^{i} C_k \lceil \frac{t}{T_k} \rceil \leq t. \quad (4)$$

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

$$\sum_{k=1}^{i} \frac{1}{\eta_{i,j}} C_k \lceil \frac{S_{i,j}}{T_k} \rceil = S_{i,j}, \quad (5)$$

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

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

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

$$\eta = \max_i \eta_i. \quad (7)$$

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[1]

$$f_{max} = \eta f_{ref}. \quad (8)$$

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 [6]. Thus, η is straightforward to compute because it is equal to the processor utilization, given by

$$\eta = \sum_{\forall \tau_i \in \tau} \frac{C_i}{T_i}. \quad (9)$$

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 (9) 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 (9), 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 (2), 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 (7), which yields

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

Thus, we can reduce the maximum speed by as much as half or can use the processor with half performance (see Figure 1(b)). □

2.3 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 rely on a power-efficient version of priority-based real-time scheduing method [4], which we call `lpps` for brevity.

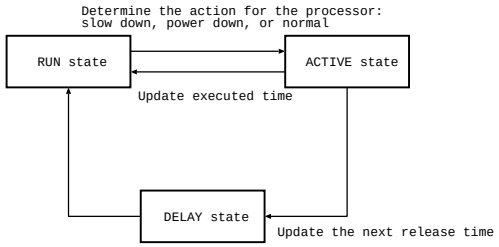

Figure 2. Diagram of task state transition.

It is based on the implementation model of priority-based scheduling in the kernel [9]. The priority-based scheduling can be implemented by maintaining 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 periods and are waiting for their next periods 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. These processes can be described by a diagram of task state transition shown in Figure 2, where each arc is annotated with actions required for `lpps`.

During the state transition from RUN to ACTIVE as shown in Figure 2, we determine the action required for the processor: lower the speed of the VSP, bring the processor to a power-down mode, or execute in full speed. We take the first two actions when run queue is empty. Specifically, we vary the speed of the VSP when there is only one task that needs the processor (when active task is present but run queue is empty) and its required execution time is less than its allowable time frame[2], and bring the processor to a power-down mode when there is no task that needs the processor (when all tasks reside in delay queue). With these run-time actions taken on the processor, idle intervals, which arise during run-time, can be exploited for power reduction.

3 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 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 [10]. We also compare the result with that of [4].

[1] Actually, 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.

[2] min(active_task.deadline, delay_queue.head.release_time) − current_time.

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

We collect four applications for experiments: an `avionics` task set [11], an `ins` [12], a `flight_control` [13], and a `cnc` machine controller [14]. 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 $m = \frac{\text{BCET}+\text{WCET}}{2}$, where BCET indicates the best case execution time, and standard deviation of $\sigma = \frac{\text{WCET}-\text{BCET}}{6}$[3]. Then, the BCET is varied from 10% to 100% of the WCET for each task.

First, f_{max} and V_{max} are obtained for each application using (7) and (9), 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.

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 3, 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 1). 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.

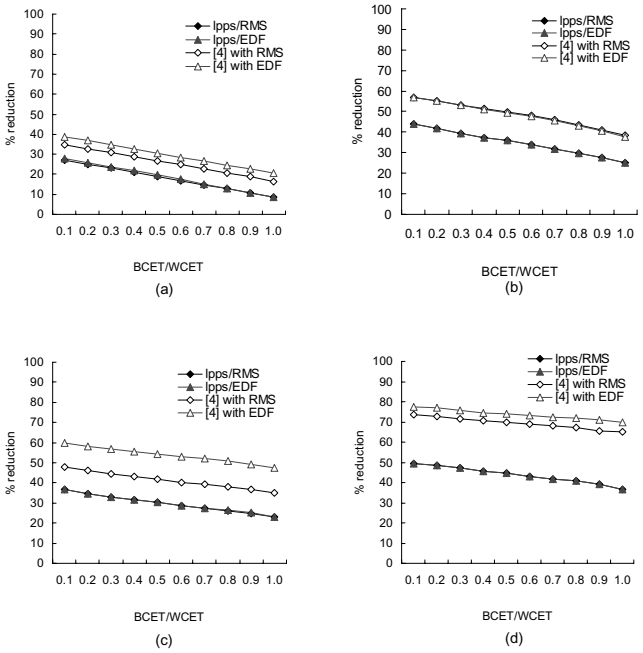

Figure 3. Simulation results of (a) `avionics`, (b) `ins`, (c) `flight_control`, and (d) `cnc`.

4 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.

[3] In a random Gaussian distribution, the probability that a random variable x takes on a value in the interval $[m-3\sigma, m+3\sigma]$ is approximately 99.7%. Thus, if we set WCET to be equal to $m+3\sigma$, almost all generated values fall between BCET and WCET. Let $m+3\sigma =$ WCET and solving for σ with the help of equation for m, we get equation for σ. After the generation of execution times, we apply clamping operation so that the generated value does not exceed WCET.

References

[1] 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.

[2] 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.

[3] 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.

[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] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.