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

Youngsoo Shin†, Kiyyoung 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 brings 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, the 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 \( f_{\text{ref}} \), and the reference supply voltage, denoted as \( V_{\text{ref}} \), 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
\[
t_d = k \frac{V_{dd}}{(V_{dd} - V_t)\alpha}
\]

where \( t_d \) is the circuit delay, \( k \) is a constant, \( \alpha \) 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 \( \tau_i \) 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 \( (f_{ref} \text{ and } V_{max}) \).

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 \( \tau_2 \) were to take a little longer to complete, \( \tau_3 \) 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).

\begin{table}[h]
\centering
\caption{An example task set}
\begin{tabular}{|c|c|c|c|c|}
\hline
\( T_i \) & \( D_i \) & \( C_i \) & Priority \\
\hline
\tau_1 & 50 & 50 & 5 & 1 \\
\tau_2 & 80 & 80 & 10 & 2 \\
\tau_3 & 100 & 100 & 20 & 3 \\
\hline
\end{tabular}
\end{table}

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 \( \tau_i \) is defined by
\[
S_i = \{kT_j | j = 1, 2, \ldots; k = 1, \ldots, \left\lfloor \frac{T_i}{D_i} \right\rfloor \},
\]

if \( T_i = D_i \). If \( D_i \) is different from \( T_i \), (2) can be modified as
\[
S_i = \{S_i - \{\eta t > D_i \}} \cup \{D_i\}.
\]

\( \eta t \) can be scheduled without violating its deadline, if there exist one or more scheduling points \( t \in S_i \), which satisfy
\[
\sum_{k=1}^{i} C_k \left( \frac{t}{T_k} \right) \leq t.
\]

Now, it is assumed that elements of \( S_i \) are sorted in ascending order. \( S_{i,j} \) is defined as the \( j \)th element of \( S_i \) that is, \( j \)th scheduling point of \( \tau_i \). Thus, for each scheduling point \( S_{i,j} \), \( \tau_i \) just meets its scheduling point if it satisfies
\[
\sum_{k=1}^{i} \frac{1}{n_{i,j}} C_k \left( \frac{S_{i,j}}{T_k} \right) = S_{i,j},
\]

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

$$\eta_{i,j} = \frac{\sum_{k=1}^{i} C_k \left[ \frac{S_{i,k}}{T_k} \right]}{S_{i,j}}. \quad (6)$$

Because $T_k$ is schedulable if it completes its execution before or at any scheduling points and the minimum possible speed scaling factor is needed for $T_k$ for minimum power consumption, speed scaling factor for $T_k$, denoted by $\eta_k$, is given by $\eta_k = \min_{i,j} \eta_{i,j}$. 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 $\eta$, is given by

$$\eta = \max \eta_k. \quad (7)$$

Note that if $\eta$ 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

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

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, $\eta$ is straightforward to compute because it is equal to the processor utilization, given by

$$\eta = \frac{\sum_{T_k \in \tau} C_k}{T_k}. \quad (9)$$

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, $\eta$ 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

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

We compute $\eta$ using (7), which yields

- $\eta_1 = \min \left( \frac{C_1}{T_1} \right) - 0.4$
- $\eta_2 = \min \left( \frac{C_1 + C_2}{T_2} \right) - 0.25$
- $\eta_3 = \min \left( \frac{C_1 + C_2 + C_3}{T_3} \right) - 0.5$
- $\eta = \max(\eta_1, \eta_2, \eta_3) - 0.5$

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 scheduling method [4], which we call 1pps for brevity.

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 1pps.

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 $\eta_{ref}$ 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 $\text{min}(\text{active\_task\_deadline, delay\_queue\_head\_release\_time}) - \text{current\_time}$. 

---

**Figure 2. Diagram of task state transition.**
Table 2. Maximum frequency and voltage computed for each application. $f_{ref} = 100$ MHz and $V_{ref} = 3.3$ V.

<table>
<thead>
<tr>
<th>Application</th>
<th>FPS</th>
<th>EDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avionics</td>
<td>3.1 V</td>
<td>3.0 V</td>
</tr>
<tr>
<td>ins</td>
<td>2.9 V</td>
<td>2.2 V</td>
</tr>
<tr>
<td>Flight Control</td>
<td>2.7 V</td>
<td>2.0 V</td>
</tr>
</tbody>
</table>

We collect four applications for experiments: an avionics task set [11], an ins [12], a flight control [13], and 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 $m = \frac{BCET}{WCET}$, where BCET indicates the best case execution time, and standard deviation of $\sigma = \frac{WCET - BCET}{3}$. 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 each other.

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.

References