2013 16th Euromicro Conference on Digital System Design

Effective Online Power Management with Adaptive Interplay of DVS and DPM for Embedded Real-time System

Gang Chen
TU Munich, Germany
cheng@in.tum.de

Kai Huang
TU Munich, Germany
huangk@in.tum.de

Jia Huang
fortiss GmbH, Germany
huang@fortiss.org

Christian Buckl
fortiss GmbH, Germany
buckl@fortiss.org

Alois Knoll
TU Munich, Germany
knoll@in.tum.de

Abstract—**Effective power management is an important design concern for modern embedded systems. In this paper, we present an effective framework to integrate both DVS and DPM to optimize the overall energy consumption. We propose an online algorithm to determine the optimal operating frequency and mode transition of a processor based on the runtime workload. Our algorithm runs in $O(n)$ time, where n is the number of the events stored in the system buffer. A feasibility analysis is also presented, which serves as a criteria for setting the system buffer as well as runtime schedulabilty check. The evaluations with specifications of two commercial processors show that our algorithm is more energy-efficient compared to existing schemes in the literature.**

I. Introduction

Energy efficiency has become one of the major goals in embedded system design. Using appropriate power management techniques, the lifetime for battery-operated systems could be extended and the heat dissipation could be decreased, lowering the requirement for expensive packaging and cooling technology. Power consumption of processors mainly comes from dynamic power consumption due to switching activity and static power consumption due to the leakage current [12]. In micrometer CMOS technology, dynamic power dominates power consumption of processors. As CMOS technologies are shifting toward sub-micron domains, the static power increases exponentially and becomes comparable or even greater than dynamic power.

To reduce dynamic power and static power consumption, two main mechanisms can be employed, i.e., Dynamic Voltage Frequency Scaling (DVS) and Dynamic Power Management (DPM), respectively. DVS reduces the dynamic power consumption by dynamically adjusting voltage and frequency of a processor. The disadvantage of this technique is the lack of means to reduce static power consumption. On the other hand, DPM reduces the static power by switching the processor to a sleep mode with low static power consumption. The limitations of the pure DPM are: (1) dynamic power consumption is not considered when making the DPM-related decisions; (2) mode-switching in processor causes additional energy and latency penalty. Actually, it is worthwhile to switch the processor to sleep mode only when the idle interval is longer than a certain threshold called break-even time.

DVS and DPM outperform each other in different workload and architecture configurations [8]. Concerning DVS, lower frequencies result in lower dynamic power consumption, which however prolong the task execution time and shorten idle intervals. Therefore, DVS techniques in general limit the opportunities of reducing static power. On the other hand, although running the system at higher frequencies can create longer sleep intervals and reduce more static power, DPM will cost more dynamic power and mode-switch overheads. In principle, DVS and DPM counteract each other with respect to energy reduction. The motivation of our work is to integrate DVS and DPM and find the best trade-off to reduce the total energy consumption.

This paper presents a framework that considers the trade-off between DPM and DVS and minimizes the overall energy consumption for hard real-time tasks. The core of the framework is an online algorithm that dynamically determines the optimal processor speed and activation time instants based on the runtime workload. Our algorithm is in $O(n)$ complexity while it can take into account the break-even time and available frequency bounds of the processor. By considering the integration of DVS and DPM, our framework yields significant energy reduction compared to the related work in the literature, using specifications from two commercial processor cores. We also provide a schedulability analysis of our algorithm, which serves as a criteria for setting the system buffer as well as runtime schedulabilty check.

The rest of paper is organized as follows: Section II reviews related work in the literature. Section III presents basic models and the definition of studied problem. Section IV describes our algorithm and the corresponding feasibility analysis. Power management scheme based on our algorithm is depicted in Section V. Experimental evaluation is presented in Section VI and Section VII concludes the paper.

II. Related Work

A lot of researchers have explored DVS or DPM on real-time embedded system [1], [2], [9], [11], [22]. However, only a few have considered the combination of both DPM and DVS. Chen and Kuo [4] propose to execute tasks at a certain speed and control the procrastination of real-time tasks. By accumulating the execution of the procrastinated real-time tasks in a busy interval, the processor can switch into sleep mode to reduce the energy. The work in [7] presents an algorithm on the interplay DVS and DPM to optimize the system-wide energy for frame-based real-time embedded applications. The algorithm requires that the period of an application is equal to its deadline. Based on the concept of device forbidden regions(DFRs), an algorithm is proposed in [6] to determine the optimal processor speed as well as switching decisions of a system to reduce the system-wide energy for periodic real-time tasks. Taking a set of well-known existing(DVS and DPM) policies, authors in [3] employ machine-learning mechanisms to adaptively determine a suitable policy according to the runtime workload.

Most of the above research on integration of DVS and DPM requires special patterns of event arrivals, such as periodic real-time events [4], or frame-based events with a period equal to its deadline [6], [7]. However, in practice, this precise timing information of event arrivals might not be known in advance, since the arrival time information depends on many nonfunctional factors, e.g., environmental impacts. To model the irregular events, Real-time Calculus (RTC) is proposed by Thiele et al. [19]. In [16], Maxiaguine et al. apply Real-time Calculus within DVS context and compute a safe frequency at periodic intervals with a predefined length to prevent buffer overflow of a system. Huang et al. [10] and Lampka et al. [13] proposed online DPM by predicting a tighter trace bound with different prediction algorithms, procrastinating the processing of arrived events as late as possible. The schedulability for online DVS algorithms is explored when the event arrivals are constrained by a given upper arrive curve in [5], [17]. In this paper, we propose a power management framework on the interplay of DVS and DPM for event streams with irregular arrival patterns. In this framework, a online algorithm, namely Optimal Workload-Aware Algorithm (OWAA), is proposed to determine an optimal point of switching the processor from sleep mode to active mode with an optimal active frequency. Arrival-curve model is adopted to predict the worst-case idle interval to decide whether the processor is switched into sleep mode.

III. Models and Problem

This section describes the system models used in this paper, which is the basis of our work.

A. Power Model

The power model in [5] is adopted in this article. The power consumption can be described as follows:

$$P(f) = P_{sta}+H(P_{ind}+P_d) = P_{sta}+H(P_{ind}+C_{ef}f^{\gamma}) \quad (1)$$

where P_{sta}, P_{ind}, and P_d represent static power, frequency-independence power, and frequency-dependence power, respectively. H is set to 1 and 0 when the processor is in active and sleep modes. Thus, P_{sta} can be seen as the sleep power. Moreover, C_{ef} and γ respectively refer to the switch capacitance and the dynamic power exponent, which are system-dependent constants. As the existence of P_{ind}, there is a critical frequency $f_{crit} = \sqrt[\gamma]{\frac{P_{ind}}{C_{ef}(\gamma-1)}}$, which indicates that executing at frequency f_{crit} is more energy-efficient than frequencies lower than f_{crit}.

Considering the overhead of switching the processor between active mode and sleep mode, the processor break-even time T_{BET} indicates the minimum time length that the processor should stay at sleep mode. If the interval at which the processor can stay at sleep mode is smaller than T_{BET}, the mode-switch mode overheads are larger than the energy saving. Therefore, mode-switch is not worthy. The break-even time T_{BET} can be defined as follows:

$$T_{BET} = \max{(t_{sw}, \frac{E_{sw} - P_s \cdot t_{sw}}{P_i - P_s})} \quad (2)$$

Where t_{sw} and E_{sw} denote the total state transition time and energy overhead, respectively. P_i and P_s respectively represent the idle power and sleep power.

B. Event Model

This paper considers events that could come irregularly. To model such events, the concept of arrival curve, which originated from Network Calculus [15] and Real-time Calculus(RTC)[14], [19], is adopted. In this context, a cumulative function $R(s,\, t)$ is used to describe an event stream in a system, which is defined as the number of events arriving in the time interval $[s,\, t)$. In *any* time interval of length Δ, event stream R can be characterized by an arrival curve $\alpha(\Delta)$:

$$R(t+\Delta) - R(t) \leq \alpha(\Delta),\, \forall\, t, \Delta \geq 0 \quad (3)$$

with $\alpha(\Delta) = 0$ for $\Delta \leq 0$.

Arbitrary event arrival patterns are specified by means of arrival curves [19]. In this paper, we just consider one event stream, assuming the events in stream have the same execution time C at maximum frequency f_{max} and the same related deadline D. Besides, at each time instant t, a trace of past events could be used to predict a tight bound $\mu(\Delta, t)$ for event stream R. See [10], [13] for details.

C. Problem Statement

Given are a processor P, an event stream R characterized by arrival curve α, and an energy model E. We are to design a power management scheme, that should decide when to conduct the system mode-switch (sleep, idle, or active) and how to schedule the frequency, such that (a) all the events can complete before their deadlines, (b) the frequencies produced by scheduler are within processors' available frequency bounds, and (c) the overall energy consumption of processors is minimized.

Note that in this work, we focus on the processor whose overhead for frequency scaling is small enough and can be neglected compared to the worst case execution time C. Therefore, we don't consider the processor's overhead for frequency scaling.

IV. Our Algorithm

In order to interplay DVS and DPM, a scheduler in general has to compute: (a) the switching time instant and the active frequency to turn the processor from sleep mode to active mode, (b) the time instant and the value of the frequency to

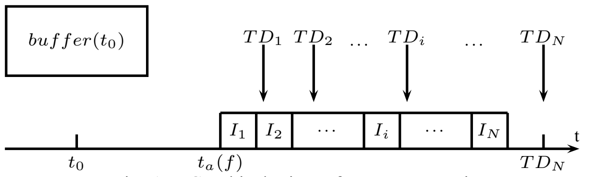


Fig. 1. Graphical View of Event Processing

scale the frequency when in active mode, and (c) when to turn the processor back to sleep mode. Therefore, we have to deal with activation decision, frequency scaling decision, and deactivation decision. This section presents an online algorithm that computes the activation decisions when the processor is in sleep mode and frequency scaling decisions when the processor is in active mode. System deactivation is considered in Section V-A. Combining DVS and DPM, our algorithm looks for the minimum energy consumption based on the runtime workload. The algorithm is triggered when the runtime workload is updated and it renews the previous computed activation and frequency scaling decisions. The algorithm determines an optimal execution frequency f as well as an activation time instant t_a by which the overall energy consumption is minimized upon the current workload, taking into account the maximum speed that the processor can provide. Analysis is also presented to depict the feasibility and complexity of the algorithm.

A. Optimal Workload-Aware Algorithm

Fig. 1 illustrates the event processing. Assume at current time instant t_0, the set of unprocessed events, $e_1, e_2, \ldots, e_N$, is denoted as $buffer(t_0)$. Note that all events are sorted by their arrival time in system buffer, and the event with earliest arrival time is processed first. a_i, D, and C_i are respectively the arrival time, relative deadline, and remaining worst-case execution time of event e_i. C is the worst-case execution time, thus $C_i \leq C$. $T_a(f)$ denotes the latest activation time of the processor, under which all events in $buffer(t_0)$ are guaranteed to satisfy their deadlines, using frequency f. I_i represents the interval of processing event e_i, which can be computed by $I_i = \frac{C_i}{f}$, while TD_i represents the absolute deadline of e_i. To guarantee all events in $buffer(t_0)$ can be processed before their deadlines, the active time $T_a(f)$ and the frequency f should meet the following constraint:

$$T_a(f) + \frac{\sum_{j=1}^{i} C_j}{f} \leq a_i + D, \forall i \in N \quad (4)$$

Thus the latest activation time $T_a(f)$ can be calculated based on $buffer(t_0)$ and frequency f:

$$T_a(f) = D + \min_{i=1}^{N} (a_i - \frac{\sum_{j=1}^{i} C_j}{f}) \quad (5)$$

The above equation shows that the time instant for activating the processor and the operating frequency f are correlated. The slower frequency at which the processor runs, the earlier the processor needs to be turned on. For brevity, we define:

Def. 1: $T_a(f, k)$ represents the activation time instant when the processor runs at frequency f and $a_i - \frac{\sum_{j=1}^{i} C_j}{f}$ with $i = 1, ..., N$ gets its minimum value at $i = k$. Then, $T_a(f, k) = D + (a_k - \frac{\sum_{j=1}^{k} C_j}{f})$.

When the processor turns on at $T_a(f, k)$ and runs at frequency f, event e_k completes right on its deadline.

According to the power model, the energy consumption of processing all events in the system buffer is:

$$E(k, f) = P(f)\frac{\sum_{j=1}^{N} C_j}{f} + P_{sta/idle}(T_a(f, k) - t_0) \quad (6)$$

where, at time instant t_0, $P_{sta/idle}$ is set to sleep power P_{sta} when the processor is in sleep mode and idle power P_{idle} when in active mode.

With Eqns. (1), (5), and (6), the energy consumption $E(k, f)$ can be computed as:

$$\begin{aligned} E(k, f) &= (P_{sta} + P_{ind} + C_{ef} f^{\gamma})\frac{\sum_{j=1}^{N} C_j}{f} \\ &\quad + P_{sta/idle}(D + a_k - \frac{\sum_{j=1}^{k} C_j}{f} - t_0) \\ &= C_{ef}\sum_{j=1}^{N} C_j f^{\gamma-1} + ((P_{sta} + P_{ind})\sum_{j=1}^{N} C_j \\ &\quad - P_{sta/idle}\sum_{j=1}^{k} C_j)\frac{1}{f} + P_{sta/idle}(D + a_k - t_0) \end{aligned}$$

For a specific event e_k, $P_{sta/idle}(D + a_k - t_0)$ can be seen as a constant in $E(k, f)$. As $\gamma \geq 1$, frequency that minimizes $E(k, f)$ can be found by analyzing its first-order derivative. Optimal frequency $f_{opt}(k)$ for event e_k can be given as:

$$f_{opt}(k) = \sqrt[\gamma]{\frac{(P_{sta} + P_{ind})\sum_{j=1}^{N} C_j - P_{sta/idle}\sum_{j=1}^{k} C_j}{C_{ef}(\gamma - 1)\sum_{j=1}^{N} C_j}} \quad (7)$$

By analyzing its first-order derivative, the following properties hold for any $\epsilon \geq 0$.

Prop. 1: $\forall f, f > f_{opt}(k), E(k, f_{opt}(k)) \leq E(k, f) \leq E(k, f + \epsilon)$

Prop. 2: $\forall f, f < f_{opt}(k), E(k, f_{opt}(k)) \leq E(k, f) \leq E(k, f - \epsilon)$

Further more, we can proof that $E(k, f)$ is strictly convex to f only when $\gamma \geq 2$ holds by analyzing its second-order derivative.

For event e_k, there is a corresponding *accepted frequency range* $[f_l(k), f_u(k)]$, which makes $a_i - \frac{\sum_{j=1}^{i} C_j}{f}$ get its minimum value at event e_k. The accepted range for e_k can be determined as follows.

$$a_k - \frac{\sum_{j=1}^{k} C_j}{f} \leq a_i - \frac{\sum_{j=1}^{i} C_j}{f}, i \neq k$$

For the case of $i \geq k + 1$, there is $a_i \geq a_k$, then

$$f \geq \frac{\sum_{j=1}^{i} C_j - \sum_{j=1}^{k} C_j}{a_i - a_k}$$

For the case of $i \leq k - 1$, there is $a_i \leq a_k$, then

$$f \leq \frac{\sum_{j=1}^{i} C_j - \sum_{j=1}^{k} C_j}{a_i - a_k}$$

To compute the accepted frequency range $[f_l(k), f_u(k)]$ for event e_k, the *accepted frequency function* AF(e_k,e_i) between e_k and e_i is defined as follows:

Def. 2: $AF(e_k, e_i) = \begin{cases} +\infty & a_i = a_k \wedge i > k \\ -\infty & a_i = a_k \wedge i < k \\ \frac{\sum_{j=1}^{i} C_j - \sum_{j=1}^{k} C_j}{a_i - a_k} & \text{otherwise} \end{cases}$

Thus, with the frequency bounds $[f_{min},\ f_{max}]$ that processor can provide, the accepted range $[f_l^*(k),\ f_u^*(k)]$ for event e_k can be calculated as:

$$f_l^*(k) = \begin{cases} \max_{i=k+1}^{N} AF(e_k, e_i) & 1 \le k \le N-1 \\ f_{min} & k = N \end{cases} \tag{8}$$

$$f_u^*(k) = \begin{cases} \min_{i=1}^{k-1} AF(e_k, e_i) & 2 \le k \le N \\ f_{max} & k = 1 \end{cases} \tag{9}$$

When the processor turns on at $T_a(f, k)$ and runs at frequency f, event e_k completes right on its deadline $a_k + D$. Consider the worst case of turning on system at current time instant t_0. To handle the worst case, the frequency f should be no less than the *gate frequency* $f_g(t_0,\ k)$.

$$f_g(t_0, k) = \frac{\sum_{j=1}^{k} C_j}{a_k + D - t_0} \tag{10}$$

Correspondingly, the accepted frequency bound $[f_l(k),\ f_u(k)]$ should be adjusted to:

$$f_l(k) = \max\left(f_l^*(k), f_g(t_0, k), f_{min}\right) \tag{11}$$

$$f_u(k) = \min\left(f_u^*(k), f_{max}\right) \tag{12}$$

The case $f_l(k) > f_u(k)$ indicates that $T_a(k, f)$ is not acceptable for event e_k. Let $\overline{f}(k)$ denote the frequency that minimizes the energy consumption when $T_a(k, f)$ is accepted in event e_k. From Prop. 1 and Prop. 2, $\overline{f}(k)$ can be determined by considering three possible scenarios:

$$\overline{f}(k) = \begin{cases} f_{opt}(k) & \text{if } f_l(k) \le f_{opt}(k) \le f_u(k) \\ f_l(k) & \text{if } f_{opt}(k) \le f_l(k) \le f_u(k) \\ f_u(k) & \text{if } f_l(k) \le f_u(k) \le f_{opt}(k) \end{cases} \tag{13}$$

The above derivation finds an optimal frequency $\overline{f}(k)$ for each event e_k in the unprocessed workload. $\overline{f}(k)$ is the local optimal frequency when $T_a(k, f)$ is accepted in event e_k. To determine the global optimal frequency, it is essential to consider all the local optimal frequency for every acceptable event currently stored in the system buffer and compare their energy consumption to select a global optimal accepted event e_{opt}. Thus, at time instant t_0, the global optimal frequency $\overline{f}_{gopt}$ and corresponding activation time $\overline{t_a}$ can be calculated as follows:

$$\overline{f}_{gopt} = \overline{f}(e_{opt}) \tag{14}$$

$$\overline{t_a} = D + a_{e_{opt}} - \frac{\sum_{j=1}^{e_{opt}} C_j}{\overline{f}_{gopt}} \tag{15}$$

where $e_{opt} = \{k | E(k, \overline{f}(k)) = \min_{i=1}^{N} E(i, \overline{f}(i)), k = 1, , N\}$.

Algo. 1 traces the current workload and generates an optimal decision $(\overline{f_{gopt}},\ \overline{t_a})$ which minimizes the energy consumption. Algo. 1 is triggered when workload is changed, i.e., at the arrival of every new event (Lines 1–9) and completion of processing for every event (Lines 10–14). The `update_accepted_frequency` routine (Line 6, Line 11) is used to refresh the accepted frequency in Eqns. (11) and (12) for all events in the backlog. The `select_optimal_frequency` routine computes the local optimal frequency $\overline{f}$ of each event using Eqn. (13), then selects the optimum using Eqns. 14 and 15. The complexity of `update_accepted_frequency` and `select_optimal_frequency` is $O(N)$ (see Section IV-C). In our approach, Algo. 1 is used to generate optimal activation decisions when the processor is in sleep mode and frequency scaling decisions when the processor is in active mode.

Algorithm 1 Online Optimal Workload-Aware Algorithm.

```
procedure ($\overline{f_{gopt}}$,$\overline{t_a}$)=BUFFER-AWARE-ALGO(signal $s$,buffer,event)
 1: if $s = event\_arrival$ then
 2:     if $get\_buffer\_state = full$ then
 3:         throw_execption
 4:     else
 5:         add_event_from_buffer(buffer,event)
 6:         update_accepted_frequency(s,buffer)
 7:         ($\overline{f_{gopt}}$,$\overline{t_a}$)=select_optimal_frequency(buffer)
 8:     end if
 9: end if
10: if $s = event\_completion$ then
11:     update_accepted_frequency(s,buffer)
12:     remove_event_from_buffer(buffer,event)
13:     ($\overline{f_{gopt}}$,$\overline{t_a}$)=select_optimal_frequency(buffer)
14: end if
```

B. Algorithm Feasibility Analysis

This section presents a schedulability analysis of our algorithm. We prove that our algorithm can guarantee that (a) all events in the system backlog can be processed within their deadlines; (b) under certain condition, the chosen frequency $\overline{f_{gopt}}$ does not exceed the maximal available frequency f_{max} and the activation time $\overline{t_a}$ is no earlier than at current time instant t_0.

Lem. 1: Turning on the processor at time instant $\overline{t_a}$ with frequency $\overline{f}_{gopt}$ computed by Algo. 1 will cause no deadline violation of the unprocessed events in the system buffer.

Proof: With Eqns. (15) and (5), we can get:

$$\overline{t_a} = D + \min_{i=1}^{N}\left(a_i - \frac{\sum_{j=1}^{i} C_j}{\overline{f}_{gopt}}\right) \le D + \left(a_i - \frac{\sum_{j=1}^{i} C_j}{\overline{f}_{gopt}}\right).$$

Thus

$$\overline{t_a} + \frac{\sum_{j=1}^{i} C_j}{\overline{f}_{gopt}} \le D + a_i, \forall e_i \tag{16}$$

holds. The left side of Eqn. (16) is the completion time of e_i and the right side is the absolute deadline of e_i. The inequality shows that all events complete no later than their deadlines. ■

Algo. 1 is triggered only when the workload is changed, i.e., at the time instant of the arrival and completion of an event. Thus, the feasibility analysis can be divided into two cases, i.e., new event arrival and event completion scenarios. Consider the feasibility of the worst case of our algorithm, i.e.,

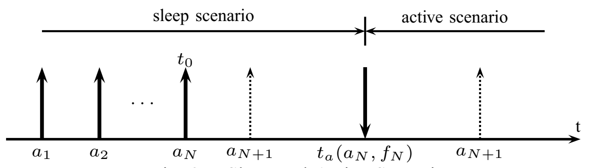


Fig. 2. Sleep and Active Scenarios

all events in the system buffer can be completed within their deadlines when the processor turns on at current time instant t_0 and executes at f_{max}. The feasibility of the worst case can guarantee that, at any time instant when the scheduling decision is re-computed, the resulted activation time will be no earlier than the current time and the computed frequency is no larger than f_{max}.

Lem. 2: At the time instant of event completion t_0, Algo. 1 is feasible.

Proof: Let t_p be the last updating time instant before t_0 and f_p be the updating frequency computed at t_p. According to Lem. 1, at time instant t_p, all events in $buffer(t_p)$ can be safely processed under f_p. During the interval $(t_p,\, t_0]$, the workload is decreased due to event completion. It is apparent that running system at f_p and activating system at current time instant t_0 can guarantee the remaining events to complete before their deadlines. As $f_{max} \geq f_p$, the worst case is also feasible. ■

Lem. 3: Define $buf(t)$ as the number of events stored in the system buffer at time instant t and a_N as the arrival time instant of N^{th} event. If $buf(a_N) \leq \left\lfloor \frac{Df_{max}}{C} \right\rfloor$ holds, Algo. 1 can find a feasible frequency and active time at updating time instance a_N.

Proof: We divide the proof into two cases, i.e., new event arrives before and after activation time t_a. Lets call these two cases sleep scenario and active scenario , respectively. Let $t_a(t,\, f)$ denote the updated activation time at time instant t. Fig. 2 shows that at time instant a_N, the system buffer contains N events with arrival time $a_1, ..., a_N$. Assume the arrival of new event e_N triggers the processor to update the feasible frequency f_N and new activation time $t_a(a_N,\, f_N)$. a_{N+1} denotes the arrival time instant for next new event. Sleep scenario is the case of $a_{N+1} < t_a(a_N, f_N)$ while active scenario is the case of $a_{N+1} \geq t_a(a_N, f_N)$. What we need to prove is that, for both cases, the condition of $buf(a_{N+1}) \leq \left\lfloor \frac{Df_{max}}{C} \right\rfloor$ can guarantee that Algo. 1 is feasible at updating time instant a_{N+1}.

- sleep scenario($a_{N+1} < t_a(a_N, f_N)$):

For all events e_i $(i = 1, ..., N)$, assume $a_i - \frac{\sum_{j=1}^{i} C_j}{f_{max}}$ gets its minimal value at event e_p. According to Eqn. (5) and $f_N \leq f_{max}$, we have:

$$t_a(a_{N+1}, f_{max}) = \min(t_a(a_N, f_{max}), D + a_{N+1} - \frac{\sum_{j=1}^{N+1} C_j}{f_{max}}) \tag{17}$$

$$\begin{aligned} t_a(a_N, f_{max}) &= D + a_p - \frac{\sum_{j=1}^{p} C_j}{f_{max}} \geq D + a_p - \frac{\sum_{j=1}^{p} C_j}{f_N} \\ &\geq D + \min_{k=1}^{N}(a_k - \frac{\sum_{j=1}^{k} C_j}{f_N}) = t_a(a_N, f_N) \\ &> a_{N+1} \end{aligned} \tag{18}$$

As $buf(a_{N+1}) \leq \left\lfloor \frac{Df_{max}}{C} \right\rfloor$ and $C_j \leq C$ hold, we can get:

$$D + a_{N+1} - \frac{\sum_{j=1}^{N+1} C_j}{f_{max}} \geq D + a_{N+1} - \frac{buf(a_{N+1})C}{f_{max}} \geq a_{N+1} \tag{19}$$

With Eqns. (17), (18), and (19), we have $t_a(a_{N+1},\, f_{max}) \geq a_{N+1}$ which shows that at the next updating time instant a_{N+1}, we can always activate the processor not earlier than a_{N+1} when the processor runs at the maximal frequency.

- active scenario $a_{N+1} \geq t_a(a_N, f_N)$:

Let t_p be the last updating time instant before a_{N+1}. Thus, $buf(a_{N+1}) = buf(t_p) + 1$. Assume f_p is the updating frequency at t_p. According to Lem. 1, at time instant t_p, all events in $buffer(t_p)$ can be safely processed under f_p. Consider the worst case of active scenario, i.e., the processor turns on at a_{N+1} and runs at f_{max}. Running processor at f_p and activating processor at current time instants a_{N+1} can guarantee the first $buf(t_p)$ events in $buffer(a_{N+1})$ to complete before their deadlines. Notes that f_p cannot guarantee the last event e_{N+1} to complete before its deadline. As $f_{max} \geq f_p$, the first $buf(t_p)$ events can also be safely processed under under f_{max}.

For the last event e_{N+1} in $buffer(a_{N+1})$, the completion time $TC(e_{N+1})$ under f_{max} can be computed as $TC(e_{N+1}) = a_{N+1} + \frac{\sum_{j=1}^{buf(a_{N+1})} C_j}{f_{max}} \leq a_{N+1} + D$ with $buf(a_{N+1}) \leq \left\lfloor \frac{Df_{max}}{C} \right\rfloor$. It shows that the worst case of active scenario is feasible if $buf(a_{N+1}) \leq \left\lfloor \frac{Df_{max}}{C} \right\rfloor$ holds.

From above cases, the lemma holds. ■

Lem. 3 shows that our algorithm can accumulate at most $\left\lfloor \frac{Df_{max}}{C} \right\rfloor$ unprocessed events in the system buffer. It offers a criterion for allocating the capacity of the system buffer. We can set the capacity of the system buffer as $\left\lfloor \frac{Df_{max}}{C} \right\rfloor$. If the system buffer overflows, it indicates that our online algorithm is not feasible. Otherwise, our algorithm is feasible. Based on Lem. 3, we provide a general feasible analysis for arbitrary arrival patterns.

Lem. 4: Given an event stream R characterized by α, Algo. 1 is feasible when $\alpha(D) \leq \left\lfloor \frac{Df_{max}}{C} \right\rfloor$ holds.

Proof: Assume at current time t_0, N events with arrival time $a_1, a_2, ..., a_N$ are stored in $buffer(t_0)$. According to deadline constraints, $a_1 + D \geq t_0 \geq a_n$. Thus, $buf(t_0) = R(a_n) - R(a_1) \leq \alpha(a_n - a_1) \leq \alpha(D)$. With $\alpha(D) \leq \left\lfloor \frac{Df_{max}}{C} \right\rfloor$ and Lem. 3, we know Algo. 1 is feasible. ■

C. Algorithm Complexity Analysis

The computational overhead of Algo. 1 comes from (a) refreshing the accepted frequency $[f_l(k),\, f_u(k)]$ (Eqns. (11) and (12)) and (b) computing the optimal decisions (Eqn. (14) and (15)).

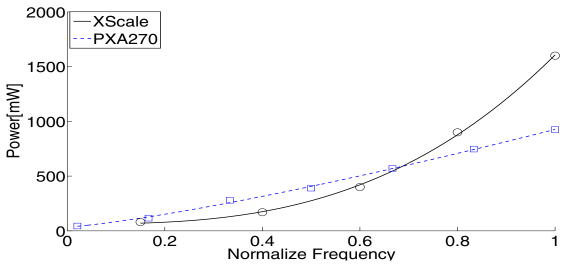


Fig. 3. Power Consumption Function

The accepted frequency $[f_l(k),\, f_u(k)]$ is refreshed at the point of adding new events to the system buffer or removing completed events from buffer. For brevity, let $[f_l^*(n,k),\, f_u^*(n,k)]$ be the accepted frequency bound determined by Eqns. (8) and (9) when buffer contains n events. For the case of adding new event e_{n+1} to buffer, $[f_l^*(n+1,k),\, f_u^*(n+1,k)]$ can be refreshed by the following iterations:

$$f_l^*(n+1,k) = \begin{cases} max(f_l^*(n,k), AF(e_k,e_{n+1})) & 1 \le k \le n \\ f_{min} & k = n+1 \end{cases} \tag{20}$$

$$f_u^*(n+1,k) = \begin{cases} f_u^*(n,k) & 1 \le k \le n \\ \min_{i=1}^{n} AF(e_i,e_{n+1}) & k = n+1 \end{cases} \tag{21}$$

$$s_u^*(n+1,k) = \begin{cases} s_u^*(n,k) & 1 \le k \le n \\ \mathrm{smin}_{i=1}^{n} AF(e_i,e_{n+1}) & k = n+1 \end{cases} \tag{22}$$

Note that $smin$ is the operation for getting the second minimum value in a group of numbers and $s_u^*(n,k)$ is used in case of removing completed event from the system buffer. $smin$ can be executed integrated with $\min$. Considering the *gate frequency* f_g and the processor frequency bound $[f_{min},\, f_{max}]$, the adjusted frequency could be determined by Eqns. (11) and (12). There are at most n iterations in `update_accepted_frequency` routine when a new event is added into the system buffer.For the case of removing completed events, `update_accepted_frequency` routine also needs at most n iterations. $[f_l^*(n-1,k),\, f_u^*(n-1,k)]$ could be similarly determined by the following iterations:

$$f_l^*(n-1,k-1) = f_l^*(n,k) \qquad k \ge 2 \tag{23}$$

$$f_u^*(n-1,k-1) = \begin{cases} s_u^*(n,k) & f_u(n,k) = AF(e_k,e_1) \& k \ge 3 \\ f_u^*(n,k) & f_u(n,k) \ne AF(e_k,e_1) \& k \ge 3 \\ f_{max} & k = 2 \end{cases} \tag{24}$$

From above analysis, it can be concluded that the complexity of update_accepted_frequency is $O(n)$, where n is the number of events stored in the system buffer.

From Eqns. (14) and (15), assuming there are n accepted events in the system buffer, computing the optimal frequency $\overline{f_{opt}}$ and activation time $\overline{t_a}$ also need $O(n)$ operations.

V. Power Management Strategy

A power management strategy involves solving the following two problems to save energy: (a) determine the activation time and the activation frequency when the processor is in sleep mode, and (b) determine the running frequency and the time to turn the processor back to sleep mode when the processor is in active mode. Meanwhile, the scheduler decisions must guarantee the real-time constraints and meet the frequency bounds.

A. System Deactivation

There are two scenarios that the system could switch to sleep mode, i.e., when the system buffer is empty and when the idle time interval $\overline{t_a} - t_0$ computed by Algo. 1 is longer than T_{BET}. In the following, we consider these two scenarios individually.

Empty system buffer: Here we use an online DPM system deactivation scheme based on the dynamic counter technique proposed in [13]. To satisfy the required deadline D and prevent system buffer with capacity Q from overflow, the minimum service demand can be computed as follows:

$$\beta(\Delta,t) = \max\left(\mu(\Delta,t) - Q, \mu(\Delta - D,t)\right) \tag{25}$$

where $\mu(\Delta,t)$ is the predicted bound of arrive curve determined by the dynamic counter. Then, the largest period τ that the processor can stay in sleep mode with predictive active frequency f_a can be computed as:

$$f_a(\Delta - \tau) \ge C\beta(\Delta,t) \tag{26}$$

If τ is larger than the break-even time T_{BET}, the processor should turn to sleep mode. Otherwise, the processor enters the idle mode. An interesting problem in this prediction is how to set the predictive frequency f_a. If f_a is set to the maximum frequency f_{max}, processor can greedily explore the opportunity for energy saving, but a higher speed is required when the processor turns back to active mode. Moreover, in some scenarios, e.g., when the idle power of the processor is relative small and close to the sleep power, energy saving obtained by switching the system into sleep mode might be very limited. This method would produce pessimistic results, as the dynamic power consumption is the dominating part. In our approach, the critical speed f_{crit} is selected as the predictive frequency to compute τ. The choice generates better results for both processors used in our experiment.

Idle time interval $\overline{t_a} - t_0 \ge T_{BET}$: Assume that at current updating time t_0, $(\overline{f_{gopt}},\overline{t_a})$ is the newly computed optimal frequency and mode-switch point. The workload at time instant t_0 is denoted as $w(t_0)$. Consider any interval $[t_0, t_0+\Delta)$, the number of events arrived in this interval is upper-bounded by $\mu(\Delta,t_0)$. The future event $e_{\mu(\Delta,t_0)+1}$ arrives later than the time instant $t_0 + \Delta$, i.e., $a_{\mu(\Delta,t_0)+1} \ge t_0 + \Delta$. Let $t_a(t,f)$ denote the active time with active frequency f at updating time instant t. To guarantee that the sleep time is larger than T_{BET} with arrival curve $\mu(\Delta,t_0)$, $t_a(a_{\mu(\Delta,t_0)+1}, f_{gopt})$ should be no less than $t_0 + T_{BET}$. According to Eqn. (5), the following constraint should be met for future event $e_{\mu(\Delta,t_0)+1}$.

$$\inf_{\Delta \le T_{BET}} (D + a_{\mu(\Delta,t_0)+1} - \frac{w(t_0) + (\mu(\Delta,t_0)+1)C}{f_{gopt}}) \ge t_0 + T_{BET} \tag{27}$$

With $a_{\mu(\Delta,t_0)+1} \geq t_0 + \Delta$, we have:

$$f_{gopt}(\Delta + D - \frac{w(t_0)}{f_{gopt}} - T_{BET}) \geq (\mu(\Delta, t_0) + 1)C \qquad \Delta \leq T_{BET} \tag{28}$$

If Eqn. (28) holds, the processor can turn to sleep mode. Otherwise, the processor enters idle mode.

B. Sytem Activation

Once the processor enters sleep mode, it should go back to active mode with an appropriate frequency at a later moment for event processing. Further analysis is needed to find the optimal operating frequency and switching moment. On the one hand, it is beneficial to turn on processor as late as possible for the sake of static power reduction. The side-effect is however a higher execution frequency later in the active mode. On the other hand, an early wake-up would require a lower operating frequency and therefore reduce the dynamic power consumption. To resolve this contradiction, Algo. 1 is used to find the optimal switching point and operating frequency for the current workload. Algo. 1 works in an workload-triggered manner. Optimal decision is re-calculated when the workload is updated, e.g., at new event arrival and event completion. In sleep mode, the decision is also updated on arrival of new events.

Based on the switching point and active frequency determined by Algo. 1, a further concern is that the processor should stay in sleep mode long enough to achieve power reduction (i.e., the sleep time exceeds the break-even time T_{BET}). As a result, Eqn. (10) should be adjusted as follows:

$$f_g(t_0, k) = \frac{\sum_{j=1}^{k} C_j}{a_k + D - max(t_0, t_{off} + T_{BET})} \tag{29}$$

where t_{off} is the time instant of turning off the processor.

Consider the scenario that a new event arrives at t_0, Algo. 1 generates the switch moment t_a and operating frequency f. If no further event arrives before t_a, processor is turned on at t_a with frequency f. Otherwise, the switching moment and execution frequency are updated at the time instant of the new event arrival. The feasibility of this activation strategy is already analyzed in Section IV-B (see Lem. 3).

C. Frequency Selection On Active Mode

When the processor enters active mode, Algo. 1 is used to compute the optimal frequency $\overline{f_{opt}}$ and active time $\overline{t_a}$ to process the current workload. Here Algo. 1 works in a workload-triggered manner. The frequency is updated at the point of new event arrival and event completion. When $\overline{t_a} - t_0 \geq \overline{T_{BET}}$ holds, a mode-switching decision is made. Otherwise, $\overline{f_{opt}}$ is applied immediately to process the events. When the system buffer is empty, the approach in Section V-A is applied to decide whether to switch the processor into sleep mode.

VI. Evaluation

This section provides experimental evaluations for the proposed framework. We implement our algorithms in MATLAB and evaluate the performance by comparing with three existing on-line algorithms in literature [5], [13].

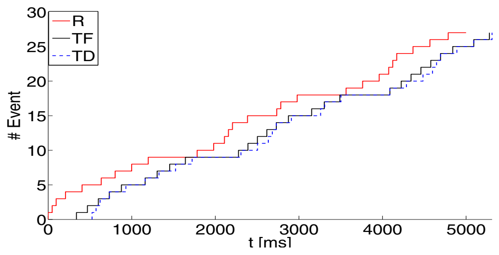


Fig. 4. Completion Time and Deadline

	S1	S2	S3	S4	S5	S6
P	198	102	283	239	148	114
J	387	70	269	222	91	13
d	48	45	58	65	78	0
C	35	11	45	38	20	15

TABLE I
PARAMETERS FOR SIX EVENT STREAMS IN [MS]

A. Experimental Setup

We adopt the PJD arrive pattern defined by parameters period p, maximum jitter j, and minimum event inter-arrival time d. The arrival curve for such models can be represented as $\alpha(\Delta) = min(\left\lceil \frac{\Delta+j}{p} \right\rceil, \left\lceil \frac{\Delta}{d} \right\rceil)$. The specification of six event streams adopted in [17] is listed in Tab. I. The RTC/RTS-toolbox [20] is used to generate the traces with different patterns. The worst-case execution time (WCET) at frequency f_{max} required for processing an event is also given in Tab. I. The WCETs for lower frequencies are normalized w.r.t the ratio to the f_{max}. The relative deadline is set to the period times a deadline factor η.

We use the hardware parameters from Marvell PXA270 [18] and Intel XScale [21] in our experiments. The voltage-frequency levels supported by PXA270 and Intel XScale are listed in Tab. II. From those discrete operating points, we use the least square curve fitting to obtain the frequency-power function in the form of $af^{\gamma} + b$ for each processor. In this case, the active power function of PXA270 and XScale can be written as $P_a(f) = 35.09 + 891.24f^{1.26}$ and $P_a(f) = 63.58 + 1543.28f^{2.87}$, respectively. The results are also depicted in Fig. 3. The idle power P_{idle}, sleep power P_{sta}, mode-switch energy overhead E_{sw}, mode-switch transition time overhead t_{sw}, and the critical speed f_{crit} of the two processors are listed in Tab. III.

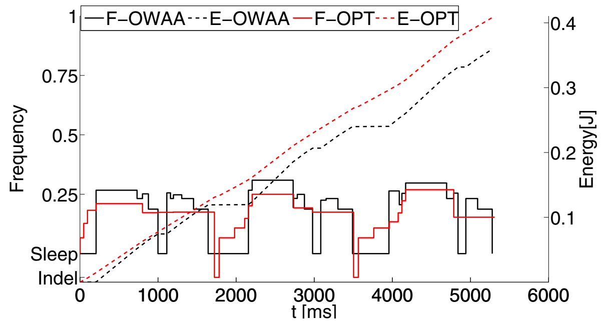
To evaluate the performance of our framework, we com-

PXA270	V (V)	1.55	1.45	1.35	1.25	1.15	0.9	0.85
	f (MHz)	624	520	416	312	208	104	13
	P (mW)	925	747	570	390	279	116	44.2
XScale	V (V)	1.8	1.6	1.3	1.0	0.75		
	f (MHz)	1000	800	600	400	150		
	P (mW)	1600	900	400	170	80		

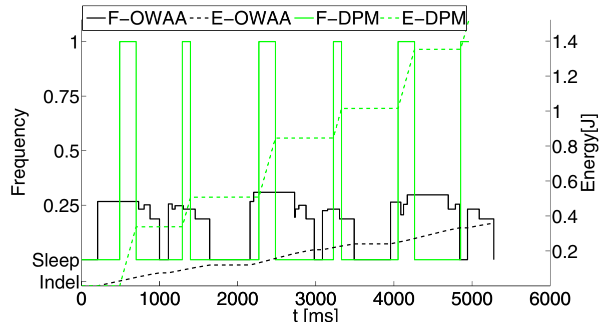
TABLE II
ACTIVE POWER CONSUMPTION FOR PROCESSORS PXA270 AND XSCALE

	P_{idle} (mW)	P_{sta} (mW)	E_{sw} (mJ)	t_{sw} (ms)	f_{cir}
PXA270	15.4	0.163	0.24	69.575	0.2211
XSale	40	0.8	0.5	85	0.2633

TABLE III
POWER PARAMETERS FOR PXA270 AND XSCALE



(a) OWAA vs DVS-OPT



(b) OWAA vs DPM-DNC

Fig. 5. Frequency Schedules and Energy Consumption

pared the energy consumption with the following algorithms:

•DPM-DNC: A latest online DPM algorithm for aperiodic tasks proposed in [13] which minimizes the energy consumption with workload prediction using dynamic counters.

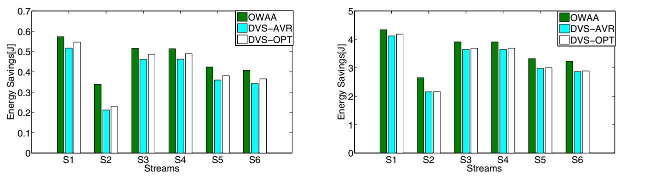
•DVS-AVR: The frequency decisions are performed by $\max(f_{min}, \sum_{e_i:a_i\leq t\leq d_i}\frac{C}{D})$, where a_i and d_i are respectively the arrival time and absolute deadline of the event e_i, and C and D are the worst-case execution time and the relative deadline [5].

•DVS-OPT: It makes the scheduling decision at time t by processing the event with the earliest deadline at speed $\max(f_{min}, \max_{e_j} \sum_{e_i:a_i\leq t, d_i\leq d_j}\frac{C_i(t)}{dj-t})$, where the event e_i is associated with arrival time a_i and absolute deadline d_i, and $C_i(t)$ is the remaining execution time at time t [5].

•OWAA: Power management scheme base on the optimal workload-aware algorithm (OWAA) proposed in Section V.

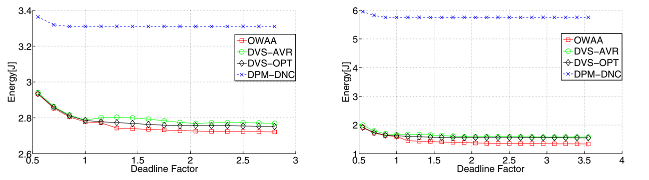
B. Simulation Result

Qualitative evaluation: We conduct the qualitative examinations on XScale processor for stream S_1. The experiment runs for five seconds. Fig. 4 shows the event completion time and



(a) PXA270 (b) XScale

Fig. 6. Energy Savings Compared to DPM-DNC



(a) PXA270 (b) XScale

Fig. 7. Energy Consumption of Different Deadline Settings Combining DVS and DPM

its deadline. The solid line TF and dashed line TD show the event completion curve and its deadline curve, respectively. As it can be seen, all events are completed before their deadlines.

Fig. 5 depicts frequency schedule and the resulting accumulative energy consumption produced by our scheme and DVS-OPT scheme where the prefix `F-` and `E-` represent the frequency schedule and the accumulative energy consumption of the schemes. As it can be seen, using the DVS-OPT scheme, the processor turns from idle state to active mode right after the new event arrives and runs with a low frequency. As the counterpart, our scheme first performs an appropriate delay to reduce the energy consumption and then selects a higher frequency to execute the tasks. From the accumulative energy consumption curves of both schemes (Fig. 5(a)), one can find out that our scheme exhibits better power efficiency than the DVS-OPT scheme during the entire time span.

Fig. 5(b) compares our scheme and the DPM-DNC scheme. Using DPM-DNC, the processor executes at the maximum speed to shorten the execution time and greedily explore the idle interval to switch the processor into sleep mode. However, the accumulative energy consumption curves in Fig. 5(b) show that doing this is actually suboptimal from the system point of view, due to the high dynamic power consumption. Unlike DPM-DNC, our scheme considers the trade-offs between dynamic and static power consumptions and controls the processor to run at an appropriate lower frequency according to the current workload, resulting in a lower overall power consumption.

Quantitative comparison: First, we consider the overall energy consumption and compare the proposed scheme with the three existing schemes. We run the simulation for 20 seconds and set the deadline factor to $\eta = 1.6$. Fig. 6 shows the energy savings w.r.t DPM-DNC[1] on PXA270 and XScale processors for streams listed in Tab. I. As depicted in Fig. 6, the proposed OWAA scheme outperforms the other three schemes in both devices. On average, for the PXA270 processor, our scheme

[1]energy savings of x policy w.r.t y policy: EnergySaving = PC(y)-PC(x), where PC(x) is power consumption of x

achieves 462.3 mJ and 45.9 mJ energy savings w.r.t DPM-DNC and DVS-OPT respectively. For the XScale processor, our scheme saved 3559.8 mJ and 290.2 mJ respectively w.r.t DPM-DNC and DVS-OPT.

In the next set of experiments, we compares the overall power consumption of the four schemes running on both devices with varying deadline factor η. The simulation time is again 20s and stream S_1 is used as the input. Fig. 7 shows the results. As it can be seen, our online scheme that combines both DVS and DPM achieves the best results. For a small deadline factor ($\eta \leq 1$), the number of unprocessed events that our algorithm can aggregate is limited and the opportunity for DPM is low. In this case, the contribution of energy savings mainly comes from the dynamic frequency scaling. We can see from Fig. 7 that OWAA can still exhibit higher energy gain compared to the other three schemes. With increasing deadline factor, OWAA can achieve further static power saving by combining the advantages of DVS and DPM. When the deadline factor η is large enough, power consumption curves of all schemes become flat. The energy consumption on PXA270 stays constantly at 2.72 J, 2.75 J, 2.77 J, and 3.31 J, using OWAA, DVS-OPT, DVS-AVR, and DPM-DNC scheme, respectively. One observation from Fig. 7 is that the performance gap between DPM-DNC and the other three algorithms is relatively small for the case of PXA270. The main reason is that the power consumption function of PXA270 is almost linear to the frequency (recall that $\gamma = 1.26$) and the trade-off between DVS and DPM is not significant. For the case of XScale, the power consumptions figures are 1.33 J, 1.55 J, 1.58 J, and 5.75 J for OWAA, DVS-OPT, DVS-AVR, and DPM-DNC schemes respectively. Our OWAA scheme saves about 14.19% power consumption w.r.t DVS-OPT on XScale.

VII. Conclusion

This paper presents an efficient framework for online energy optimization that considers the interplay of DVS and DPM. The core of the framework is an online algorithm that dynamically determines the optimal processor speed and mode transitions based on the runtime workload. Our algorithm is in $O(n)$ complexity and it takes into account the break-even time and available frequency bounds of the processor.We also provides a guideline for configuring the system buffer size based on a feasibility analysis.Experimental results with specifications from two commercial processors show that our framework outperforms existing online power management approaches in the literature.

References

[1] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *RTSS 2001: Proceedings of the 2001 22th IEEE Real-Time Systems Symposium*, pages 95 – 105, 2001.

[2] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proceedings of the 2004 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 520 – 529, 2004.

[3] K. Bhatti, C. Belleudy, and M. Auguin. Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. In *Proceedings of the 2010 IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing*, pages 184 –191, 2010.

[4] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *ICCAD 2007: Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 289 –294, 2007.

[5] J.-J. Chen, N. Stoimenov, and L. Thiele. Feasibility analysis of on-line dvs algorithms for scheduling arbitrary event streams. In *RTSS 2009: Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, pages 261 –270, 2009.

[6] V. Devadas and H. Aydin. Dfr-edf: A unified energy management framework for real-time systems. In *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 121 –130, 2010.

[7] V. Devadas and H. Aydin. On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications. *IEEE Transactions on Computers*, 61(1):31 –44, 2012.

[8] G. Dhiman and T. Rosing. Dynamic power management using machine learning. In *ICCAD 2006: Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, pages 747 –754, 2006.

[9] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. Buttazzo. Adaptive power management for real-time event streams. In *Proceedings of the 2010 15th Asia and South Pacific Design Automation Conference(ASP-DAC)*, pages 7 –12, 2010.

[10] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. Buttazzo. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems*, 47:163–193, 2011. 10.1007/s11241-011-9115-z.

[11] C.-M. Hung, J.-J. Chen, and T.-W. Kuo. Energy-efficient real-time task scheduling for a dvs system with a non-dvs processing element. In *RTSS 2006: Proceeding of the 2006 27th IEEE International Real-Time Systems Symposium*, pages 303 –312, 2006.

[12] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *DAC 2004: Proceedings of 2004 41st ACM/IEEE Design Automation Conference*, pages 275 –280, 2004.

[13] K. Lampka, K. Huang, and J.-J. Chen. Dynamic counters and the efficient and effective online power management of embedded real-time systems. In *CODES+ISSS 2011: Proceedings of the 9th IEEE/ACM International Conference on the Hardware/Software Codesign and System Synthesis*, pages 267 –276, 2011.

[14] J. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[15] J.-Y. Le Boudec. Application of network calculus to guaranteed service networks. *IEEE Transactions on Information Theory*, 44(3):1087–1096, may 1998.

[16] S. Maxiaguine, A. Chakraborty and L. Thiele. Dvs for buffer-constrained architectures with predictable qos-energy tradeoffs. In *CODES+ISSS '05: Proceedings of the 2005 IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, pages 111 –116, 2005.

[17] S. Perathoner, K. Lampka, N. Stoimenov, L. Thiele, and J.-J. Chen. Combining optimistic and pessimistic dvs scheduling: An adaptive scheme and analysis. In *ICCAD 2010: Proceedings of the 2010 IEEE/ACM International Conference on Computer-Aided Design*, pages 131 –138, 2010.

[18] Marvell PXA270. `http://www.marvell.com/application-processors/pxa-family`

[19] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of the 2000 IEEE International Symposium on Circuits and Systems*, volume 4, pages 101 –104 vol.4, 2000.

[20] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[21] R. Xu, D. Mossé, and R. Melhem. Minimizing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Trans. Comput. Syst.*, 25(4), 2007.

[22] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of 36th Annual Symposium on Foundations of Computer Science*, pages 374 –382, 1995.