Adaptive Power Management for Real-Time Event Streams

Kai Huang, Luca Santinelli*, Jian-Jia Chen, Lothar Thiele, Giorgio C. Buttazzo*
Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
Email: {firstname.lastname}@tik.ee.ethz.ch
* Real-Time Systems Laboratory, Scuola Superiore Sant'Anna of Pisa, Italy
* Email:{luca.santinelli,giorgio}@sssup.it

Abstract— Dynamic power management has become essential for battery-driven embedded systems. This paper explores how to efficiently and effectively reduce the energy consumption of a device (system) for serving multiple event streams. Considering two different preemptive scheduling, i.e., earliest deadline first and fixed priority, we propose new method to adaptively control the power mode of the device according to historical arrivals of events. Our method can not only tackle arbitrary event arrivals but also provide hard real-time guarantees with respect to both timing and backlog constraints. Simulation results are presented as well to demonstrate the effectiveness of our approach.

Keywords: Adaptive Power Management, Energy Minimization, Real-Time Event Streams, Real-Time Calculus.

I. Introduction

Power management with energy efficiency considerations has been an important design issue, especially for battery-driven embedded devices to extend their battery life-time. Dynamic power consumption due to switching activities and static power consumption due to the leakage current are two major sources of power consumption of a CMOS circuit [9]. For micrometer-scale semiconductor technology, the dynamic power dominates the power consumption of a processor. However, as the CMOS technology is scaling downward aggressively to the deep sub-micron domain, the leakage power consumption increases exponentially and is comparable to or even more than the dynamic power dissipation.

This paper explores how to apply dynamic power management (DPM) to reduce the energy consumption for *hard* real-time embedded systems by changing the mode of a device. We consider a device with *active*, *standby*, and *sleep* modes with different power consumptions, and a controller decides when to change the power mode of the device. Intuitively, the device can be switched to the sleep mode to reduce the power consumption when it is idle. This switching operation, however, has two concerns. On one hand, the sleep period should be long enough to recuperate the mode-switch overhead. On the other hand, to cope with the burstiness of event arrivals, the reserved time for serving the burst events must be sufficient to prevent deadline violation of events and backlog overflow of the system when activating the device again later on.

To cope with these two concerns, we propose online algorithms in [7]. Trying to be optimistic for the controller, events are handled only when they really arrive. Our algorithms adaptively predict the next moment for mode switch by considering both historical and future event arrivals, and procrastinate the buffered and future events as late as possible. In this paper, we extend the basic algorithms in [7] for multiple event streams with different characteristics. In particular, a more realistic backlog model is adopted, i.e., distributed backlogs for each event steam. We consider two different preemptive scheduling, i.e., earliest deadline first (EDF) and fixed priority (FP), and develop means to guarantee the timing and backlog constraints for the given event streams.

The rest of this paper is organized as follows: We review the related work in the next section. Section III and IV present our system model and basics of our analysis, respectively. We present our solutions in Section V. Simulations results are presented in Section VI. Section VII concludes the paper.

II. Related Work

Dynamic power management (DPM) with clock gating or voltage gating can be applied to change the device power mode, e.g., to a sleep mode, to consume less (static/leakage) power. For devices with the sleep mode, Baptiste [2] proposes an algorithm based on dynamic programming to control when to turn on/off a device for aperiodic real-time events with the same execution time. For multiple low-power modes, Augustine et al. [1] determine the mode that a processor should enter for aperiodic real-time events and propose a competitive algorithm for online use. Swaminathan et al. [12] explore dynamic power management of real-time events in controlling shutting down and waking up system devices for energy efficiency. To aggregate the idle time for energy reduction, Shrivastava et al. [11] propose a framework for code transformations. By considering platforms with both DPM and dynamic voltage scaling (DVS), Chen and Kuo [3] propose to execute tasks at a certain speed (mostly at the critical speed) and to control the procrastination of real-time events. By turning the device to the sleep mode, the execution of the procrastinated real-time events is aggregated in a busy interval to reduce energy consumption. Heo [6] et al. explore how to integrate different power management policies in a server farm.

Most of the above approaches require either precise information of event arrivals, such as periodic real-time events [3], or aperiodic real-time events with known arrival time [2, 1, 8]. However, in practice, the precise timing information of event arrivals might not be known in advance since the arrival time depends on many factors. When the precise timing of event arrivals is unknown, to our best knowledge, the only known approaches are to apply the online algorithms proposed by Irani

978-1-4244-5767-0/10/$26.00 2010 IEEE

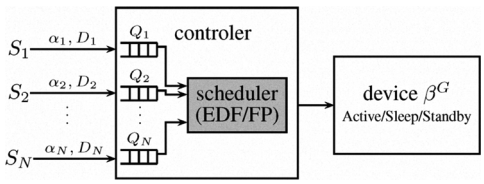


Fig. 1: The abstract system model of the studied problem.

et al. [8] and Augustine et al. [1] to control when to turn on the device. However, since the online algorithms in [1, 8] greedily stay in the sleep mode as long as possible without referring to incoming events in the near future, the resulting schedule might make an event miss its deadline. Such algorithms are not applicable for hard real-time systems.

To model such irregular events, Maxiaguine et al. [10] apply Real-Time Calculus within the DVS context and compute a safe frequency at periodical interval with predefined length to prevent buffer overflow of a system. Recently, Chen et al. [4] explore the schedulability for online DVS scheduling algorithms when event arrivals are constrained by a given upper arrival curve. In contrast to these closest approaches, we focus on DPM and our adaptation points are dynamic and vary according to the actual arrivals of events. Furthermore, we focus on multiple event-stream scenarios where event streams with different periods can be tackled with both earliest-deadline-first and fixed-priority scheduling.

III. System Models and Problem Definition

System Model We consider a device controlled by a controller which handles event arrivals and controls the power mode of the device to serve the arrived events. The device has three power modes, namely *active, standby,* and *sleep* modes. The power consumption in the sleep mode is P_σ. To serve an event, the device must be in the active mode with power consumption P_a, in which $P_a > P_\sigma$. Once there is no event to serve, the device can enter the sleep mode. However, switching from the sleep mode to the active mode and back takes time, denoted by $t_{sw,on}$ and $t_{sw,sleep}$, and incurs energy overhead, denoted by $E_{sw,on}$ and $E_{sw,sleep}$, respectively. To prevent the device from frequent mode switches, the device can also stay in the standby mode. The power consumption P_s in the standby mode, by definition, is no more than P_a and is more than P_σ. We assume that switching between the standby mode and the active mode has negligible overhead, the same assumption as in [17, 16].

Event streams with different properties arrive to the controller. Suppose that there are N event streams in a given set $\mathcal{S}$. To buffer incoming events of each stream S_i in set $\mathcal{S}$, the controller maintains a separate backlog of size Q_i. Buffering more than Q_i events incurs a backlog overflow and causes a controller failure. We assume that Q_i is given. Deciding Q_i for a given global backlog constraint is not considered in this paper. An abstract model of our system is shown in Fig. 1, where the controller could be the operating system and the device could be an I/O peripheral device, for instance. Parameters α, D, and β^G in Fig. 1 will be introduced next.

Event Model To model irregular arrival of events, we adopt the arrival curves $\bar{\alpha}(\Delta) = [\bar{\alpha}^u(\Delta), \bar{\alpha}^l(\Delta)]$ from Real-Time Calculus [13], in which $\bar{\alpha}_i^u(\Delta)$ and $\bar{\alpha}_i^l(\Delta)$ are the upper and lower bounds on the number of arrival events for a stream S_i in any time interval of length Δ, respectively. For instance, for an event stream with period p, jitter j, and minimal inter arrival distance d, the upper arrival curve is $\bar{\alpha}^u(\Delta) = \min\{\lceil \frac{\Delta+j}{p} \rceil, \lceil \frac{\Delta}{d} \rceil\}$. The concept of arrival curves unifies many other timing models of event streams. Analogous to arrival curves that provide an abstract event stream model, a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ defines an abstract resource model which provides an upper and lower bounds on the available resources in any time interval Δ. Please refer to [14] for details.

Note that an arrival curve $\bar{\alpha}_i(\Delta)$ specifies the number of events of stream S_i whereas a service curve $\beta(\Delta)$ specifies the available amount of time for execution, for interval length Δ. Therefore, $\bar{\alpha}_i(\Delta)$ has to be transformed to $\alpha_i(\Delta)$ to indicate the amount of computation time required for the arrived events in intervals. Suppose that the execution time of any event in stream S_i is w_i. The transformation can be done by $\alpha_i^u = w_i\bar{\alpha}_i^u$, $\alpha_i^l = w_i\bar{\alpha}_i^l$ and back by $\bar{\alpha}_i^u = \alpha^u/w_i$, $\bar{\alpha}_i^l = \alpha^l/w_i$ thereof. Moreover, to satisfy the real-time constraint, the response time of an event in event stream S_i must be no more than its specified relative deadline D_i, where the response time of an event is its finishing time minus the arrival time of the event. On the arrival of an event of stream S_i at time t, the absolute deadline is $t + D_i$.

Problem Definition This paper explores how to effectively minimize the energy consumption to serve a set $\mathcal{S}$ of N event streams by DPM. Intuitively, energy saving can be obtained by a) turning the device to the sleep mode when no event to process, and b) staying at the sleep mode as long as possible. However, switching from/to the sleep mode incurs overhead. As a result, there is a break-even time T_{BET} defined as:

$$\max\Big\{t_{sw,\,on} + t_{sw,\,sleep},\ \frac{E_{sw,\,on} + E_{sw,\,sleep}}{P_s - P_\delta}\Big\}.$$

In the case of a sleeping interval is shorter than T_{BET}, the mode-switch overhead is more than the energy consumption of staying in the standby mode. Turning the device to the sleep mode, therefore, is not worthwhile. Prolonging the sleep mode, on the other hand, might make current and future events violate their timing constraints or incur backlog overflow.

We say that a scheduling decision is *feasible* if it is always possible to meet the timing and backlog constraints for any event traces constrained by an arrival curve. An algorithm is *feasible* if it always generates feasible scheduling decisions.

Therefore, the problem studied in this paper is to decide a feasible schedule for a) *when to turn the device from the sleep mode to the active mode to serve events*, and b) *when to turn the device to the sleep mode to reduce the energy consumption.*

IV. Real-time Calculus Basics

To compute a safe interval for putting the device to sleep, we apply Real-Time Calculus [13] and Real-Time Interface [14]. Within this context, the device is said to provide guarantee output service $\beta^G(\Delta)$. Correspondingly, a stream S_i requests service demand $\beta^A(\Delta)$. For instance, to satisfy the required related deadline D_i, the service demand $\beta^A(\Delta)$ of stream S_i is

$$\beta^A(\Delta) = \alpha_i^u(\Delta - D_i). \tag{1}$$

To obtain a feasible scheduling of stream S_i on the device, the condition $\beta^G(\Delta) \geq \beta^A(\Delta)$ has to be fulfilled.

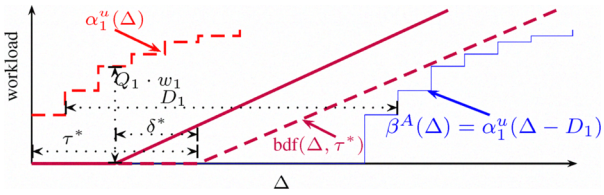


Fig. 2: An example of the bounded delay function for stream S_1, in which only part of the upper arrival curve $\alpha_1^u(\Delta)$ is presented for simplicity.

Bounded Delay We construct $\beta^G(\Delta)$ by a *bounded delay function* $\boldsymbol{bdf}(\Delta, \tau)$ which is defined as no service provided for at most τ units of time:

$$\boldsymbol{bdf}(\Delta, \tau) = \max\{0, (\Delta - \tau)\}, \forall \Delta \geq 0. \tag{2}$$

Moreover, the longest delay τ^* for providing a service guarantee for a given demand $\beta^A(\Delta)$ is defined as:

$$\tau^* = \max\{\tau : \boldsymbol{bdf}(\Delta, \tau) \geq \beta^A(\Delta), \forall \Delta \geq 0\}. \tag{3}$$

To prevent a backlog of size Q_i from overflow, this τ^* needs to be reduced by δ^* which is defined as

$$\delta^* = \max\{0, \min\{\delta : \alpha_i^u(\Delta) - \boldsymbol{bdf}(\Delta, \tau^* - \delta) \leq Q_i \cdot w_i, \forall \Delta\}\}. \tag{4}$$

With the computed τ^* and δ^*, if $\tau^* - \delta^*$ is larger than the break-even time T_{BET}, we can safely turn the device to the sleep mode while guaranteeing a feasible scheduling. Figure 2 depicts an example for the above analysis for one event stream.

Future Prediction with Historical Information As the scheduling decision is made online and depends on the actual arrivals of events, we keep the track of event arrivals in the past as a history. If a burstiness has been observed recently, one could predict that in the near future only sparse events will arrival due to the constraint of the arrival curve. Suppose t is the current time and $R_i(t)$ is the accumulated number of events of stream S_i in interval $[0, t)$. Δ^h is the history window of the controller, in which historical information for only Δ^h time units is retained. We define at time t the history curve as

$$H_i(\Delta, t) = \begin{cases} R_i(t) - R_i(t - \Delta), & \text{if } \Delta \leq \Delta^h, \\ R_i(t) - R_i(t - \Delta^h), & \text{otherwise.} \end{cases} \tag{5}$$

The maximal future event arrivals $\bar{\alpha}_i^u(\Delta, t)$ in the near future from time t to $t + \Delta$ is thereby bounded by

$$\bar{\alpha}_i^u(\Delta, t) \leq \inf_{\lambda \geq 0} \{\bar{\alpha}_i^u(\Delta + \lambda) - H_i(\lambda, t)\}. \tag{6}$$

Analogously, we denote at time t the set of unfinished events of S_i in the backlog Q_i as $\boldsymbol{E_i}(t)$. Suppose that those events in $\boldsymbol{E_i}(t)$ are indexed as $e_{i,1}, e_{i,2}, \ldots, e_{i,|\boldsymbol{E_i}(t)|}$ from the earliest to the latest deadline, where $|\boldsymbol{E_i}(t)|$ is the number of events in the backlog and $D_{i,j}$ is the absolute deadline of event $e_{i,j}$. We can model the service demand for those events in the $\boldsymbol{E_i}(t)$ as

$$B_i(\Delta, t) = w_i \cdot \begin{cases} (j-1), & D_{i,j} - t < \Delta \leq D_{i,j+1} - t, \\ |\boldsymbol{E_i}(t)|, & \Delta > D_{i,|\boldsymbol{E_i}(t)|} - t, \end{cases} \tag{7}$$

in which $D_{i,0}$ is defined as t for brevity.

V. Adaptive Dynamic Power Management

In this section, we present our online power management scheme. Subsection V.A presents an overview of our scheme and Subsection V.B sketches the solution for systems with one event stream as illustration. In Subsection V.C, we present in details our solution for multiple event streams with earliest-deadline-first (EDF) and fixed-priority (FP) scheduling.

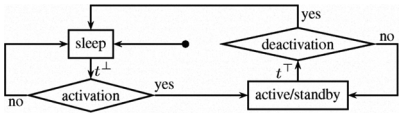


Fig. 3: The control flow of our approach.

A. Approach Overview

Our adaptive DPM scheme deals with *deactivation scheduling decisions* and *activation scheduling decisions* to decide mode switches safely and effectively. The control flow of our approach is illustrated in Fig. 3. For deactivation scheduling decisions, when the device is in the active mode and there is no event in the backlog, we develop an algorithm to decide whether the device has to change to the sleep mode instantly or it should keep in the standby mode for a while for serving incoming events in the near future. For the rest of the paper, time instants for deactivation decisions are denoted by $t^\top$.

For activation scheduling decisions, when the device is in the sleep mode and there is an event arriving or the sleep interval set by the controller expires, we use two different algorithms to decide whether the device has to change to the active mode instantly to serve events, or it should remain in the sleep mode for a while to aggregate more events to prevent from unnecessary mode switches. Time instances for activation decisions are denoted as $t^\perp$ for the rest of the paper.

B. Systems with One Event Stream

History-Aware Deactivation The History-Aware Deactivation (HAD) algorithm analyzes whether the device should be turned to the sleep mode from the active mode. The principle is to deactivate the device only when energy saving is possible. In the case of only one event stream S_1, a safe sleep interval of a feasible scheduling is obtained by $\tau^* - \delta^*$, applying $\bar{\alpha}_1^u(\Delta, t^\top)$ defined in (6) to (3) and (4). If this interval is larger than T_{BET}, the device is switched to the sleep mode at time $t^\top$.

Worst-Case-Greedy Activation The Worst-Case-Greedy (WCG) algorithm decides the *earliest* time when the device should change to the active mode for event processing. It conservatively assumes worst-case event arrivals and decides the earliest time to activate. If the worst case does not occur, the device is kept in the sleep mode for a longer period and an new activation moment is computed. The WCG algorithm works in a time-driven manner. Each time the predicted wakeup time $t^\perp$ comes, the wakeup decision is reevaluated based on the actually arrived events. We use the following formulas to derive the new wakeup time:

$$\beta^A(\Delta) = \alpha_1^u(\Delta - D_1, t^\perp) + B_1(\Delta, t^\perp), \tag{8}$$

$$\tau^\perp = \max\{\tau : \boldsymbol{bdf}(\Delta, \tau) \geq \beta^A(\Delta)\}, \tag{9}$$

$$\delta^\perp = \max\{0, \min\{\delta : \alpha_1^u(\Delta, t^\perp) - \boldsymbol{bdf}(\Delta, \tau^\perp - \delta) \leq (Q_1 - |\boldsymbol{E_1}(t^\perp)|) \cdot w_1, \forall \Delta\}\}. \tag{10}$$

If $\tau^\perp - \delta^\perp$ is larger than 0, the device can remain in the sleep mode and the next wakeup prediction is set to $t^\perp + \tau^\perp - \delta^\perp$.

Note that the first wakeup time is set to the arrival of the first event after the device is turned to the sleep mode. In this case, $B_1(\Delta, t)$ in (8) is 0 and $\boldsymbol{E_1}(t)$ in (10) is $\emptyset$ by definition.

Event-Driven Activation The Event-Driven-Greedy (EDG) algorithm computes the *latest* time that the device must be acti-

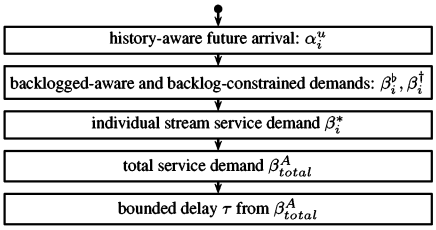


Fig. 4: Computing flow for scenarios of multiple event streams.

vated to satisfy the timing constraint. It optimistically assumes the least events and decides the latest turn-on time. The wakeup decision is reevaluated upon each event arrival until the predicted wakeup time hits. If the predicted wakeup time hits, the device has to be switched on immediately.

On the arrival of an event $e_{1,j}$ at time $t^\perp$, we choose the latest processing time $t' = t^\perp + D_1 - w_1$ as the reference time to compute the wakeup time. To precisely predict the burstiness after time t', the historical arrival and the backlog demand at time t' are redefined by appending the least event arrivals within interval $[t^\perp, t')$ constrained by $\alpha_1^l(\Delta)$ to those at time $t^\perp$:

$$H_1'(\Delta, t') = \begin{cases} \bar{\alpha}_1^l(\epsilon) - \bar{\alpha}_1^l(\epsilon - \Delta), & \Delta < \epsilon, \\ H_1(\Delta, t^\perp) + \bar{\alpha}_1^l(\epsilon), & \epsilon < \Delta < \Delta^h - T, \\ H_1(\Delta^h - \epsilon, t^\perp) + \bar{\alpha}_1^l(\epsilon), & \text{otherwise}, \end{cases} \quad (11)$$

$$B_1'(\Delta, t') = w_1 \cdot \begin{cases} (j-1), & D_{1,j} - t' < \Delta \le D_{1,j+1} - t'; \\ \mathcal{E}, & \Delta > D_{1,\mathcal{E}} - t', \end{cases} \quad (12)$$

where $\epsilon = t' - t^\perp$ and $\mathcal{E} = |\boldsymbol{E_1}(t^\perp)| + \bar{\alpha}_1^l(\epsilon)$. With the refined historical information and backlog demand, we can again apply (2) to compute a new wakeup alarm for event $e_{1,j}$:

$$\alpha_1^u(\Delta, t') = w_1 \cdot \Big(\inf_{\lambda \ge 0} \{\bar{\alpha}_1^u(\Delta + \lambda) - H_1(\lambda, t)\} \Big), \quad (13)$$

$$\beta^A(\Delta) = \alpha_1^u(\Delta - D_1, t') + B_1'(\Delta, t'), \quad (14)$$

$$\tau^\perp = \max\{\tau : \boldsymbol{bdf}(\Delta, \tau) \ge \beta^A(\Delta)\}, \quad (15)$$

$$\delta^\perp = \max\{0, \min\{\delta : \alpha_1^u(\Delta, t') - \boldsymbol{bdf}(\Delta, \tau^\perp - \delta) \le (Q_1 - |\boldsymbol{E_1}(t^\perp)| - \bar{\alpha}_1^l(\epsilon)) \cdot w_1, \forall \Delta\}\}. \quad (16)$$

If $t^\perp + \tau^\perp - \delta^\perp$ is earlier than the previous prediction, the predicted wakeup time is set to $t^\perp + \tau^\perp - \delta^\perp$. Otherwise, the previous prediction remains.

With this approach, both DPM schemes, i.e., HAD-WCG and HAD-EDG, provide feasible scheduling which guarantees the deadline constraint of any event as well as the backlog constraint at any time for one event-stream system. The detailed algorithms and the proofs are referred to [7].

C. Multiple Event Streams

To tackle multiple-stream scenarios, the key is to harness the scheduling impact at every reevaluation of the mode-switch decision. The basic approach is depicted in Fig. 4. Unlike systems with one event stream where the bounded delay is applied directly to the service demand of a stream, we compute the individual service demand of every stream, denoted as β_i^*, then derive the total service demand, denoted as β_{total}^A, according to a given scheduling policy thereof. With the computed β_{total}^A, the bounded delay is applied to calculate the feasible sleep interval. This approach is affected for all HAD, WCG, and EDG algorithms. Because of the similarity and limited space, we present the solution for the EDG algorithm only. In this paper, we provide solutions for earliest-deadline-first (EDF) and

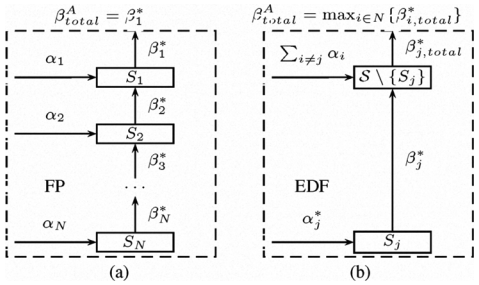


Fig. 5: Total service demand calculation for FP and EDF scheduling.

fixed-priority (FP) scheduling. Note that the refinements of the history curve and backlog demand in (11) and (12) can be applied to individual stream, denoted as H_i' and B_i' for briefness.

FP Scheduling For fixed-priority scheduling, without loss of generality, the event streams $S_1, S_2, \ldots, S_N$ are ordered according to their priorities, where the priority of stream S_i is higher than that of S_k when $k > i$. Streams can thereby be modeled as an ordered chain according to their priorities and a lower priority stream can only make use of the resource left from a higher priority stream. To compute the service demand of a higher priority stream, a backward approach is applied by considering the service demand from the directly lower priority stream, as shown in Fig. 5 (a). The service demand of stream S_N at time $t' = t^\perp + D_1 - w_1$ is

$$\beta_N^*(\Delta, t') = \max\{\beta_N^\flat(\Delta, t'), \beta_N^\dagger(\Delta, t')\}, \text{where} \quad (17)$$

$$\beta_N^\flat(\Delta, t') = \alpha_N^u(\Delta - D_N, t') + B_N'(\Delta, t'), \quad (18)$$

$$\beta_N^\dagger(\Delta, t') = \alpha_N^u(\Delta, t') - \big(Q_N - |\boldsymbol{E_N}(t^\perp)| - \bar{\alpha}_N^l(t' - t^\perp)\big) \cdot w_N, \quad (19)$$

$$\alpha_N^u(\Delta, t') = w_N \cdot \Big(\inf_{\lambda \ge 0} \{\bar{\alpha}_N^u(\Delta + \lambda) - H_N(\lambda, t')\} \Big). \quad (20)$$

To derive β_1^*, we have to compute the service bounds $\beta_{N-1}^*, \beta_{N-2}^*, \ldots, \beta_2^*$, sequentially. Suppose that β_k^* has been derived, the resource constraint is that the remaining service curve should be guaranteed to be no less than β_k^*, i.e.,

$$\beta_{k-1}^\sharp(\Delta) \ge \inf\{\beta : \beta_k^*(\Delta, t') = \sup_{0 \le \lambda \le \Delta} \{\beta(\lambda) - \alpha_{k-1}^u(\lambda, t')\}\} \quad (21)$$

By inverting (21), we can derive $\beta_{k-1}^\sharp$ as:

$$\beta_{k-1}^\sharp(\Delta) = \beta_k^*(\Delta - \lambda) + \alpha_{k-1}^u(\Delta - \lambda, t') \quad (22)$$
$$\text{where } \lambda = \sup\{\tau : \beta_k^*(\Delta - \tau, t') = \beta_k^*(\Delta, t')\}.$$

To guarantee the timing constraint of event stream S_{k-1}, we also know that β_{k-1}^* must be no less than its own demand. Therefore, we know that

$$\beta_{k-1}^*(\Delta) = \max\{\beta_{k-1}^\sharp(\Delta), \beta_{k-1}^\flat(\Delta, t'), \beta_{k-1}^\dagger(\Delta, t')\}, \text{where} \quad (23)$$

$$\beta_{k-1}^\flat(\Delta, t') = \alpha_{k-1}^u(\Delta - D_{k-1}, t') + B_{k-1}'(\Delta, t'), \quad (24)$$

$$\beta_{k-1}^\dagger(\Delta, t') = \alpha_{k-1}^u(\Delta, t') - \big(Q_{k-1} - |\boldsymbol{E_{k-1}}(t^\perp)| - \bar{\alpha}_{k-1}^l(t' - t^\perp)\big) \cdot w_{k-1}, \quad (25)$$

$$\alpha_{k-1}^u(\Delta, t') = w_{k-1} \cdot \Big(\inf_{\lambda \ge 0} \{\bar{\alpha}_{k-1}^u(\Delta + \lambda) - H_{k-1}(\lambda, t')\} \Big). \quad (26)$$

By applying (23) for $k = N-1, N-2, \ldots, 2$, the service demand β_1^* of stream S_1 is derived.

Based on this approach, the computed service demand for the highest priority stream S_1 can be also seen as the total service demand β_{total}^A for stream set $\mathcal{S}$ under the fixed-priority

scheduling. Therefore, the timing as well as backlog constraints for all streams in $\mathcal{S}$ can be guaranteed by the sleep interval τ^* with which $\boldsymbol{bdf}(\Delta, \tau^*)$ bounds β_1^*:

$$\tau^* = \max\{\tau : \boldsymbol{bdf}(\Delta, \tau) \geq \beta_1^*(\Delta), \forall \Delta \geq 0\}. \quad (27)$$

This leads to the following theorem:

Theorem 1 *The τ^* obtained by (27) is a feasible sleep interval at every reevaluation of the EDG algorithm and it guarantees the backlog and timing constraints for all streams in $\mathcal{S}$ under the fixed-priority scheduling, if the device provides again full service after τ^* time unit.*

EDF Scheduling For earliest-deadline-first scheduling, the total service demand β_{total}^A for all N streams can be bounded by the sum of their service demands. The β_{total}^A computed in this manner, however, is not sufficient to guarantee the backlog constraint of any stream in $\mathcal{S}$. When an event of a stream S_j is happened to have the latest deadline, events in any stream of $\mathcal{S} \setminus \{S_j\}$ will be assigned a higher priority. S_j will suffer from backlog overflow.

To compute a correct service demand to satisfy the backlog constraint for stream S_j, S_j has to be considered as the lowest priority. Similar back-forward approach is applied, as shown in Fig. 5 (b). Instead of tracing back stepwise, the service demand needed for higher-priority streams is the sum of all streams from $\mathcal{S} \setminus \{S_j\}$. Again, we present the revision of the EDG algorithm as an example. The service $\beta_j^\sharp$ to guarantee the lowest priority stream S_j should be more than the demand β_j^* of S_j, i.e.,

$$\beta_j^\sharp(\Delta) \geq \inf\{\beta : \beta_j^*(\Delta, t') = \sup_{0 \leq \lambda \leq \Delta}\{\beta(\lambda) - \sum_{i \neq j}^{N} \alpha_i^u(\lambda, t')\}\} \quad (28)$$

By inverting (28), we can derive $\beta_j^\sharp(\Delta)$ as:

$$\beta_j^\sharp(\Delta) = \beta_j^*(\Delta - \lambda, t') + \sum_{i \neq j}^{N} \alpha_i^u(\Delta - \lambda, t') \quad (29)$$

where $\lambda = \sup\{\tau : \beta_j^*(\Delta - \tau, t') = \beta_j^*(\Delta, t')\}$, and

$$\beta_j^*(\Delta, t') = \max\{\beta_j^\flat(\Delta, t'), \beta_j^\dagger(\Delta, t')\} \quad (30)$$

where $\beta_j^\flat$ and $\beta_j^\dagger$ are from (24) and (26). To guarantee the timing constraint of all higher-priority streams, we also know that $\beta_{j,total}^*$ must be no less than the demand of $\mathcal{S} \setminus \{S_j\}$ as well. Therefore, we know that at time $t' = t^\perp + D_j - w_j$,

$$\beta_{j,total}^*(\Delta) = \max\{\beta_j^\sharp(\Delta), \sum_{i \neq j}^{N} \beta_i^\flat(\Delta, t')\}, \quad (31)$$

Applying (31) to each steam in $\mathcal{S}$, the service demand for each steam is computed. Because each stream could be the lowest priority in the worst case, only the maximum of them can be seen as the total service demand for stream set $\mathcal{S}$. Therefore, the timing and backlog constraints for $\mathcal{S}$ can be guaranteed by τ^* with which $\boldsymbol{bdf}(\Delta, \tau^*)$ bounds the maximum of individual streams:

$$\tau^* = \max\{\tau : \boldsymbol{bdf}(\Delta, \tau) \geq \max_{i \in N}\{\beta_{i,total}^*(\Delta)\}, \forall \Delta \geq 0\}. \quad (32)$$

This leads to the following result:

Theorem 2 *The τ^* computed by (32) is a feasible sleep interval at every reevaluation of the EDG algorithm and it guarantees the timing and backlog constrains for all streams in $\mathcal{S}$ under EDF scheduling, if the device provides again full service after τ^* time unit.*

TABLE I: Event stream setting according to [7].

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
period (msec)	198	102	283	354	239	194	148	114	313	119
jitter (msec)	387	70	269	387	222	260	91	13	302	187
delay (msec)	48	45	58	17	65	32	78	-	86	89
w (msec)	12	7	7	11	8	5	13	14	5	6

TABLE II: Power profiles for devices according to [5].

Device Name	P_a (Watt)	P_s (Watt)	P_σ (Watt)	t_{sw} (sec)	E_{sw} (mJoule)
IBM Microdrive	1.3	0.5	0.1	0.012	9.6

VI. Simulation Results

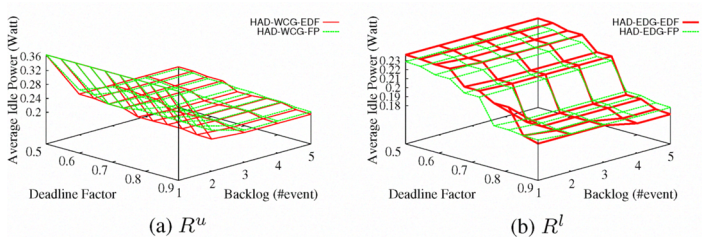
This section provides simulation results for the proposed method. The simulation is implemented in MATLAB using the RTC/RTS toolbox [15] and runs on a simulation host with Intel 1.6 *GHz* processor and 1 *GB* memory.

Simulation Setup We take the stream set studied in [7] for our case studies. The relative deadline D_i of an event stream S_i is defined by a *deadline factor* χ, i.e., $D_i = \chi * p_i$. Table I describes the parameters for generating the arrival curves of this stream set, where w is the worst-case execution time. We simulate scenarios of controlling an IBM Microdrive device, the power profiles of which is depicted in Table II. To trigger our simulation, we apply two traces with a time span of 10sec, denoted as R^u and R^l, imitating bursting and sparse event-arriving cases. Both R^u and R^l are generated by the RTS toolbox and compliant to arrival curve specifications.

We evaluate two schemes to control the device, i.e., the HAD-EDG and the HAD-WCG, applying both the EDF and the FP scheduling. Since all the schemes have the same energy consumption for event processing, we report the average idle power consumption which computed as the quotient of the sum of all the mode-switch overhead and the leakage energy consumption for the whole trace period divided by the time span of the trace. We also report the computation expense of these two schemes subject to different scheduling and traces. Due to the space limit, we plot the results for the EDF and FP scheduling within the same figures for all cases.

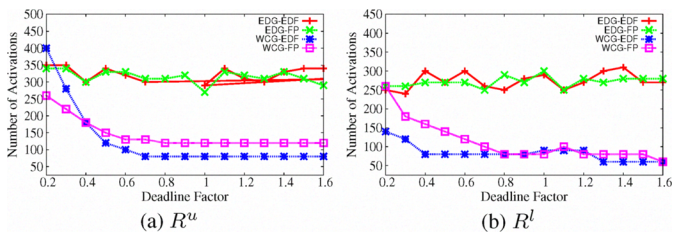
Simulation Result Firstly, we show the impact of our schemes according to different χ and backlog sizes. Due to the similarity, we only present the HAD-WCG scheme for trace R^U and HAD-EDG scheme for trace R^l. As shown in Fig. 6, the HAD-EDG and the HAD-WCG schemes reduce the average idle power consumption as χ and backlog size increases for both R^u and R^l cases. The reason is that we can procrastinate later the arrived events and accumulate more to process for each activation of the device with larger χ and backlog size. Both schemes are effective for the EDF and FP scheduling. Note that ideally the results of the two schemes should provide a same result for the same scheduling and trace. The deviation depicted in Fig. 6 is caused by the bounded delay approximation.

Secondly, we show the impact of our algorithms to the controller. Fig. 7 shows the number of reevaluation of activation decision within the 10sec time span and Fig. 8 depicts the average computation time for each reevaluation. From Fig. 7, we can notice that the activation of the EDG algorithm is varied according to the traces while the WCG algorithm is affected heavenly by the deadline. Due to constraint of this stream set, the



(a) R^u (b) R^l

Fig. 6: Average idle power consumption for EDG-HAD and WCG-HAD schemes subjected to EDF and FP scheduling, where (a) and (b) apply traces R^u and R^l, respectively.



(a) R^u (b) R^l

Fig. 7: Number of activations according to the deadline factor χ, where (a) and (b) apply traces R^u and R^l, respectively.

EDG algorithm cannot outperform the WCG algorithm. But in other cases where events come sparsely, the EDG will perform better than the WCG algorithm.

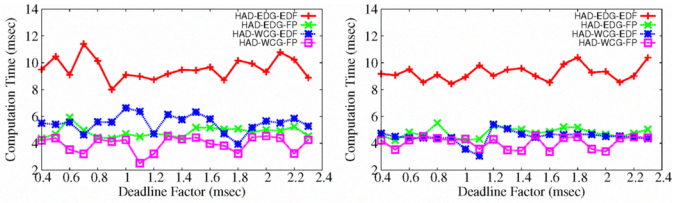
The average computation expensive of each reevaluation is depicted in Fig. 8. From the figure, we can conclude that our algorithms are efficient for both traces. The computation expenses of each activation and deactivation pair for all cases are within the range of millisecond and are acceptable to the task set in Table I. In general, the EDG algorithm and EDF scheduling are more expensive than the WCG algorithm and FP scheduling, respectively, which are confirmed with the definition in Section V. Another observation is that the computation expense is not neglectable for this stream set, which might harm the computation for the feasible sleep period. There are also means to tackle this problem, for instance, setting the computation overhead as a safe margin for the computed sleep period or putting the reevaluation itself as the highest priority events of the system. We do not elaborate them here, since they are not the focus of this paper.

VII. Conclusions

This paper explores how to apply dynamic power management to reduce the leakage power consumption for hard real-time embedded systems pertaining to both timing and backlog constraints. We propose algorithms to adaptively control the power mode of a device (system) based on the actual arrival of events, tackling multiple event streams with irregular event arrival patterns under both earliest deadline first and fixed priority preemptive scheduling. proof-of-concept simulation results demonstrate the effectiveness of approaches.

Acknowledgment

The work was partially supported by European Integrated Project SHAPES (grant no. 26825) under IST FET – Advanced Computing Architecture (ACA) and the European Community's Seventh Framework Programme FP7/2007-2013 project Predator (grant no. 216008).



(a) R^u (b) R^l

Fig. 8: Average computation time for an activation/deactivation pair of different schemes where R^u and R^l are applied to (a) and (b), respectively.

References

[1] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *45th Symposium on Foundations of Computer Science (FOCS)*, pages 530–539, Oct. 2004.

[2] P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: A polynomial time algorithm for offline dynamic power management. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithm (SODA)*, pages 364–367, 2006.

[3] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *Int'l Conf. on Computer-Aided Design (ICCAD)*, pages 289–294, 2007.

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

[5] H. Cheng and S. Goddard. Online energy-aware I/O device scheduling for hard real-time systems. In *Proceedings of the 9th Design, Automation and Test in Europe (DATE)*, pages 1055–1060, 2006.

[6] J. Heo, D. Henriksson, X. Liu, and T. Abdelzaher. Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS)*, pages 227–238, 2007.

[7] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Adaptive power management for real-time event streams. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS)*, 2009.

[8] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 37–46, 2003.

[9] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *ACM/IEEE Design Automation Conference (DAC)*, pages 275–280, 2004.

[10] A. Maxiaguine, S. Chakraborty, and L. Thiele. DVS for buffer-constrained architectures with predictable QoS-energy tradeoffs. In *the International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, pages 111–116, 2005.

[11] A. Shrivastava, E. Earlie, N. Dutt, and A. Nicolau. Aggregating processor free time for energy reduction. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, pages 154–159, 2005.

[12] V. Swaminathan and K. Chakrabarty. Pruning-based, energy-optimal, deterministic I/O device scheduling for hard real-time systems. *ACM Transactions in Embedded Computing Systems*, 4(1):141–167, 2005.

[13] L. Thiele, S. Chakraborty, and M. Naedele. Real-time Calculus for Scheduling Hard Real-time Systems. *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 4:101–104, 2000.

[14] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 243–252, Apr. 2006.

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

[16] C.-Y. Yang, J.-J. Chen, C.-M. Hung, and T.-W. Kuo. System-level energy-efficiency for real-time tasks. In *the 10th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 266–273, 2007.

[17] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *Proceedings of the 42nd ACM/IEEE Design Automation Conference(DAC)*, pages 628–631, 2005.