Technical Report

Enhanced Race-To-Halt: A Leakage-Aware Energy Management Approach for Dynamic Priority Systems

Muhammad Ali Awan

Stefan M. Petters

HURRAY-TR-110602

Version:

Date: 06-20-2011

Enhanced Race-To-Halt: A Leakage-Aware Energy Management Approach for Dynamic Priority Systems

Muhammad Ali Awan, Stefan M. Petters

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

With progressing CMOS technology miniaturization, the leakage power consumption starts to dominate the dynamic power consumption. The recent technology trends have equipped the modern embedded processors with the several sleep states and reduced their overhead (energy/time) of the sleep transition. The dynamic voltage frequency scaling (DVFS) potential to save energy is diminishing due to efficient (low overhead) sleep states and increased static (leakage) power consumption. The state-of-the-art research on static power reduction at system level is based on assumptions that cannot easily be integrated into practical systems. We propose a novel enhanced race-to-halt approach (ERTH) to reduce the overall system energy consumption. The exhaustive simulations demonstrate theeffectiveness of our approach showing an improvement of up to 8 % over an existing work.

Enhanced Race-To-Halt: A Leakage-Aware Energy Management Approach for Dynamic Priority Systems

Muhammad Ali Awan   Stefan M. Petters
CISTER Research Unit, ISEP-IPP Porto, Portugal
{maan,smp}@isep.ipp.pt

***Abstract*—With progressing CMOS technology miniaturization, the leakage power consumption starts to dominate the dynamic power consumption. The recent technology trends have equipped the modern embedded processors with the several sleep states and reduced their overhead (energy/time) of the sleep transition. The dynamic voltage frequency scaling (DVFS) potential to save energy is diminishing due to efficient (low overhead) sleep states and increased static (leakage) power consumption. The state-of-the-art research on static power reduction at system level is based on assumptions that cannot easily be integrated into practical systems. We propose a novel enhanced race-to-halt approach (ERTH) to reduce the overall system energy consumption. The exhaustive simulations demonstrate the effectiveness of our approach showing an improvement of up to 8 % over an existing work.**

I. Introduction

Embedded devices are designed to perform a set of functions and interact with their environment. Typical examples of such systems are cars, satellites or mobile phones. Real-time (RT) embedded systems have additional timing constraints, which are required to be met on top of functional aspects for the overall system to be considered correct.

Beyond the real-time constraints many embedded systems are also limited in the energy supply. Such power constraints are induced by the battery powered mobile devices or those with limited or intermittent power supply (e.g. solar cells). The most prominent hardware features available to system designers to use energy efficiently are DVFS and sleep states. With the CMOS technology scaling, the leakage power has become a significant factor in overall power consumption. Leakage power consumption is due to sub-threshold current that flows through the transistors. The technology scaling resulted in the leakage power often dominating the dynamic power [1].

The latest embedded processors are equipped with several sleep states and also decrease the overhead of transitioning into such a sleep state. Using DVFS is complex [2] and may not be optimal in terms of energy consumption, as DVFS increases the execution time and thus the leakage energy. Instead, race-to-halt followed by a sleep state is emerging as a candidate superior to DVFS in energy management [1].

This work was financed by FEDER funds (EU) through the Operational Programme 'Thematic Factors of Competitiveness' - COMPETE, by National Funds (PT) through FCT - Portuguese Foundation for Science and Technology and the ARTEMIS-JU, under the projects RePoMuC (PTDC/EIA-EIA/112599/2009) and RECOMP (ARTEMIS/0202/2009).

The increase in computing power also leads to a progressive integration of functionality in single devices. For example, a current mobile phone combines applications of soft real-time character (e.g. base station communication) with such of best-effort character (e.g. SMS). Modern cars integrate safety-critical components (e.g. airbag) with comfort functionality. Additionally the different system components and software modules are potentially provided by different third party suppliers. Consequently such mixed criticality systems require temporal and functional isolation not only to protect critical applications from less critical ones, but also as a means to identify the offending application in the case of a misbehaving system.

Most current research is based on one or more of the following very strong assumptions: a simplistic power and processor model for DVFS, a negligible time/energy overhead of entering and leaving a sleep state or the requirement of additional hardware to run the proposed approach [2]. This paper reduces the restrictive assumptions that prevent the practical use of state-of-the-art research. It assumes non-negligible time/energy overhead of sleep transition and does not require specialized hardware.

The major contributions are as follows. 1) An energy efficient slack management approach to accumulate the execution slack. 2) A simple method to calculate the break-even-time offline for different sleep states. 3) An enhanced race-to-halt (ERTH) algorithm to minimize the leakage energy consumption for the mixed-critically uniprocessor systems, while avoiding the impractical assumptions made. 4) Some improvements are also made in the procrastination approach (LC-EDF) to reduce the computational time and implementation complexity.

Section II discuss the limitations of the state-of-the-art followed by a system model in Section III. The break-even-time and schedulability is discussed in Section IV. The following section presents the slack management algorithm. The ERTH algorithm is given in section VI. Section VII discusses the offline and online overhead of ERTH against LC-EDF. The simulations results and conclusions are given in Sections VIII and IX respectively.

II. Related Work

To reduce the leakage power, Lee et al. [3] addressed the procrastination scheduling for periodic hard real-time systems and proposed Leakage Control EDF (LC-EDF) and Leakage Control Dual Priority (LC-DP) algorithms. They maximized the idle interval by delaying the busy period to increase the duration of the sleep state. An external specialized hardware (ASIC or FPGA) is assumed to implement the algorithm. Irani et al. [4] proposed offline and online algorithms for power saving while considering shutdown in combination with DVFS. Although the combination of shutdown and DVFS has its merits, but their work is not applicable in real systems due to a number of very restrictive assumptions in terms of the power model. Niu and Quan's [5] scheduling technique also integrates DVFS and shutdown to minimize the overall energy consumption based on the latest arrival time of jobs. However, this algorithm cannot be used online due to extensive analysis overhead. Previously, Jejurikar et al. [6] integrated the DVFS with LC-EDF, to minimized the total power consumption. The critical speed η_{crit} is estimated that determines the lower bound on the processor frequency to minimize the energy consumption per cycle. Nevertheless, they did not relax on the requirement of additional hardware and the power model used in their approach is also very simple.

Jejurikar et al. [7] showed that procrastination under LC-DP originally proposed by Lee et al. [3] may cause some of the tasks to miss their deadline. They proposed improvements in the original algorithm with an integration of DVFS. However, they adopted the same assumptions of [3] with simplistic power model. Later on Chen and Kuo [8] showed that an approach given in [7] still might lead to some tasks missing their deadlines. They proposed a two phase algorithm that estimates the execution speed and procrastination interval offline and predicts turn off/on instances online. Their work also assumes a very simplistic power and processor model.

The work of Jejurikar and Gupta [9] reclaims the execution slack generated due to the difference between worst-case execution time (WCET) and actual execution time. They use LC-EDF and DVFS to minimize the overall energy consumption. This algorithm follows the same assumption made by previous work [3], [6], [7]. Chen and Thiele [10] proposed leakage-aware DVFS scheduling, where tasks execute initially with decelerating frequencies to accumulate the slack to initiate sleep state and towards the end execute with accelerating frequencies to reduce the dynamic power consumption. However, it still relies on a simplistic power model. The system-level power management algorithm developed by Devadas and Aydin [11] for frame-based embedded systems addresses the interplay of DVFS and device power management (DPM). While the approach is very promising, the simple power model and the restriction of the frame-based tasks need further work.

The common assumptions made in state-of-the-art are simple, convex power model, a continuous spectrum of available frequency/voltage, negligible time/energy overhead of frequency/voltage switch/sleep transitions, external specialized hardware to run the proposed algorithm. These assumption cannot easily be integrated into practical systems and subsequently severely limit the practical relevance of the proposed work.

III. System Model

We assume a sporadic task model, with l independent tasks $\mathrm{T} = \{\tau_1, \tau_2, \cdots, \tau_l\}$. A task τ_i is described by $\langle C_i, D_i, T_i\rangle$, where C_i is the worst-case execution time (WCET), D_i the relative deadline and T_i the minimum inter-arrival time. The independent tasks are allocated a periodic budget A_i and release as a sequence of jobs $j_{i,m}$. Each job $j_{i,m}$ has a deadline $d_{i,m}$, a budget $a_{i,m}$, a release time $r_{i,m}$ and an actual execution time $\hat{c}_{i,m}$.

We use the Rate-Based Earliest Deadline first (RBED) framework [12], which provides temporal isolation via enforced budgets $a_{i,m}$ associated with each job $j_{i,m}$. This temporal isolation allows the mixing of hard, soft and best-effort type applications. The allocations of budget for soft real-time (SRT) and best-effort (BE) tasks may be less than or equal to WCET ($A_i \leq C_i$). For hard real-time (HRT) tasks the budget is equal to the WCET ($A_i = C_i$), to ensure the timely completion of all jobs. The scheduler pre-empts every job when it has used up its allocated budget $a_{i,m}$. Thus a job exceeding its budget cannot affect the overall schedulability.

We assume N sleep states in our system, where each sleep state n is characterized with a power P_n, a transition overhead of t_n and a break-even time (BET) t_n^e. The time t_n includes transition time to sleep t_n^s, as well as the wake-up time t_n^w i.e. $t_n = t_n^s + t_n^w$. Top speed and idle mode power are represented as P_{Ts}, P_{Ls} respectively.

IV. Sleep Interval Limits

In the context of our approach the break-even time t_n^e is defined as follows.

Definition 1. *The break-even time t_n^e is the time interval during which energy consumption of that sleep state n becomes equal to the energy consumption of system running at a certain frequency set-point without a load (in idle mode).*

The interval t_n^e includes the transition delays (t_n^s, t_n^w) and also compensates for the energy lost during transition phase of the sleep state. In order to estimate t_n^e, let's consider Figure 1. The overhead of the idle mode transition is assumed negligible. We assume power consumption remains steady in idle and top speed mode. However, we also assume the power consumption of the sleep state n

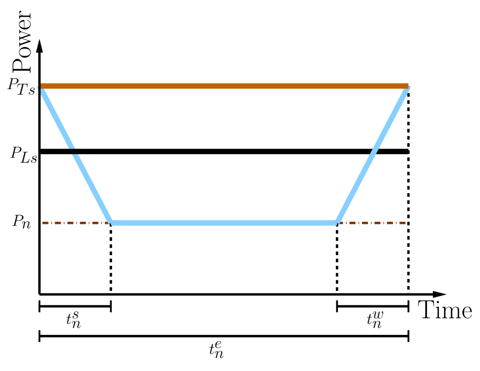

Figure 1. Break-Even Time t_n^e estimation

varies linearly during transition phase and remains uniform during sleep interval as shown in Figure 1. The area under the curve gives the energy consumption. According to our definition by equating the energy consumption of both curves, we get t_n^e as given in Equation 1. A more accurate model for t_n^e is possible and would be a function of e.g. the current DVFS state, type of the task (CPU intensive or memory intensive), etc. For the purpose of this paper, the approach presented here is sufficient, but does not prevent at all the use of more sophisticated models.

$$t_n^e = \frac{(t_n^s + t_n^w)(P_{Ts} - P_n)}{2(P_{Ls} - P_n)} \tag{1}$$

The definition of t_n^e implies that the system will save energy if a sleep state is initiated for more than t_n^e. Therefore t_n^e gives a lower bound for the desired sleep interval. In the running system a threshold value greater than t_n^e will be chosen as a minimum sleep interval. Intuitively it is preferable to choose fewer, but longer sleep intervals, when compared to more frequent sleep transitions for small intervals. While the overhead of the sleep transition has a major impact on the lower bound of the sleep interval, the schedulability of the system enforces an upper bound of the sleep interval. We define this upper bound on the sleep interval as static limit t_l for which the system can, under certain conditions (explained later in Section VI) be enforced to stay in a sleep state without causing deadline misses.

Definition 2. *The static limit t_l describes the maximum time interval for which the processor may be enforced in a sleep state without causing any applications to miss their deadlines under worst-case assumptions.*

The schedulability analysis of the EDF on uniprocessor [13], [14] is given in Theorem 1. However overall demand bound function for a T can be represented as $dbf_{\mathrm{T}}(L) \overset{\mathrm{def}}{=} max_{L_0} df(L_0, L_0 + L)$ following the definition of Rahni et al. [15].

Theorem 1. *A synchronous periodic task set* T *is schedulable under EDF if and only if, $\forall L \in L^*$, $df(0, L) \leq L$, where L is an absolute deadline and L^* is the first idle time in the schedule.*

Therefore formally the static limit is defined by exploiting the demand bound function dbf. Assuming Theorem 1, a static limit for sleep threshold t_l is given in Equation 2.

$$\forall L \in L^*, \qquad t_l = \min\left(L - dbf(L)\right) \tag{2}$$

Where L is an absolute deadline and L^* is the first idle time in the schedule.

Theorem 2. *Initiating sleep state for the static limit t_l does not violate the EDF schedule if and only if*

$$\forall L \in L^*, \qquad dbf(L) + t_l \leq L$$

Where L is an absolute deadline and L^* is the first idle time in the schedule.

Proof: The sleep interval t_l can be interpreted as highest priority task in the system. In an EDF scheduled system it is equivalent to a task with deadline equal to the shortest relative deadline of any task in the system. As such the $dbf^*(L)$ is increased over dbf(L) by t_l, following the definition in Equation 2 it follows that $dbf^*(L) \leq L$. ■

V. Slack Management Algorithm

The processing time not used in a system is called slack. This slack may be categorized in two types, dynamic and static slack. The static slack exists due to spare capacity available in the system schedule. This spare capacity occurs as the system is loaded less than what can be guaranteed by the schedulability tests.

The dynamic slack occurs due to difference between worst-case assumptions made in the offline analysis and the actual online behavior of the system. It is further divided into two components based on two different worst-case assumptions. The first worst-case assumption made is that each job of a task will execute for its WCET C_i. Most of the jobs in the real system finish execution earlier than their $a_{i,m}$ or C_i. Thus we term the first component of the dynamic slack as *execution slack* $\hat{S}_{i,m}$, which is generated by the difference in C_i and $\hat{c}_i$.

Similarly, the system is analyzed with the second assumption that each job of a sporadic task will be released as soon as possible i.e. released periodically with the minimum inter-arrival time. However, for truly sporadic tasks that rarely happens in hard real-time systems. Jobs of a sporadic tasks are released with a variable delay bounded by the minimum inter-arrival time. The slack generated due to sporadic delays is called *sporadic slack* and forms the second component of the dynamic slack. Naturally, all of the dynamic slack is generated online.

The execution slack and static slack are managed explicitly in our approach. Nevertheless, effect of the sporadic slack is considered implicitly. Our slack management approach is based on the basic principles of [16]. The execution slack $\hat{S}_{i,m}$ received from previous tasks at time instant t is represented by the tuple $S_t = \langle S_{t \cdot s}, S_{t \cdot d} \rangle$, where $S_{t \cdot s}$ corresponds to effective slack size and $S_{t \cdot d}$ corresponds to the absolute deadline of the slack. During idle mode, the system consumes available slack [17]. We use only a single container for the slack management, as keeping several containers for the slack at different priority levels would add extra online computational overhead.

To preserve the schedulability, we assume tasks with higher priority can only pass slack to the tasks of same or lower priority. When a system receives execution slack from a higher priority task, the slack deadline $S_{t \cdot d}$ is updated accordingly ($S_{t \cdot d} = max\{S_{t \cdot d}, d_{i,m}\}$), while the slack received from the previous task $S_{t' \cdot s}$ is added to $S_{t \cdot s}$ ($S_{t \cdot s} += S_{t' \cdot s}$). On every scheduling event, the slack priority is compared against the current job priority. If the slack S_t has a higher or equal priority than the current job to be executed, the actual budget $a_{i,m}$ of the current job $j_{i,m}$ is incremented by $S_{t \cdot s}$; i.e. $a_{i,m} += S_{t \cdot s}$. The slack management in our approach will not pass slack to BE tasks, as BE tasks are likely to consume the slack and we want to retain that for energy management purposes. The advantage of this approach is that slack generated at different priority levels are eventually accumulated implicitly with a very simplistic and transparent approach using just one container to hold the slack. The complete slack management algorithm is given in Algorithm 1.

Algorithm 1 Slack Managment

```
1:  **On Every Scheduling Event**
2:  **if** ($S_{t \cdot d} \leq d_{i,m}$) **then**
3:      $a_{i,m} += S_{t \cdot s}$
4:      $S_{t \cdot d} = 0$
5:      $S_{t \cdot s} = 0$
6:  **end if**
7:  **Slack Update On Job Completion**
8:  $S_{t \cdot s} += a_{i,m}$
9:  $S_{t \cdot d} = max(S_{t \cdot d}, d_{i,m})$
10: **if** ($Ready\ Queue\ Empty$) **then**
11:     Consume slack $S_t$ first
12: **end if**
```

VI. ERTH Algorithm

Three different principles are defined to initiate a sleep mode. ERTH does not initiate a sleep state for less than static limit t_l.

Principle 1:
If the task to execute in the system is HRT or SRT and the available S_t is less than t_l, slack is added to $a_{i,m}$ of the $j_{i,m}$. The system performs a race-to-halt with the likelihood of generating more slack in future. If $((S_{t \cdot s} \geq t_l) \&\& (S_{t \cdot d} \leq d_{i,m}) \&\& (HRT \| SRT))$ is true, a timer is initialized with $t_l - t_n^w$ and the sleep state is initiated until the timer expires.

Theorem 3. *If the next job to execute in the ready queue $j_{i,m}$ is of type HRT or SRT, while the execution slack has a size greater than or equal to the static limit ($S_{t \cdot s} \geq t_l$) and the slack deadline is less than or equal to the absolute deadline $d_{i,m}$ of the HRT or SRT job $j_{i,m}$ ($S_{t \cdot d} \leq d_{i,m}$), then system can initiate a sleep state for a static limit of t_l without violating EDF schedulability.*

Proof: Suppose the available S_t is considered as a task τ_{sleep} with a budget and deadline equal the t_l and $S_{t \cdot d}$ respectively. We need to prove τ_{sleep} is schedulable without an interruption in the presence of T. For this we split the potentially affected jobs of T in two parts which we address serarately: 1)$\forall \tau_i$ not released yet. 2)$\forall \tau_i$ released but in ready queue.
Case 1: ($\forall \tau_i$ not released yet)
This can be proven by contradiction. Suppose, the system schedule a τ_{sleep} for t_l and there is a synchronous arrival of all the tasks not yet released, and some of the τ_i missed their deadline. However from Theorem 2, all the τ_i in the system can be delayed for an interval of t_l without any deadline misses, which is a contradiction. Therefore all the τ_i not released yet will meet their respective deadline.
Case 2)$\forall \tau_i$ released and in ready queue.
Due to the condition expressed in Principle 1, task τ_{sleep} has a deadline earlier than any τ_i in the ready queue. This task can be scheduled before its deadline and thus τ_{sleep} will not affect the τ_i in the ready queue. Hence we proved tasks in both cases do not violate the schedule, thus theorem holds. ∎

Principle 2:
In case of the task to execute being of BE type, we use Equation 3 and Equation 4 below to evaluate the possible sleep interval. This result in two main advantages. Firstly, the sleep interval estimated φ is always equal to or greater than t_l ($\varphi \geq t_l$) as we assume that φ is computed when $S_{t \cdot s} \geq t_l$. Secondly, it is useful, when t_l is very small (i.e. high system utilization). We do not assume knowledge about the previous release times of tasks, therefore a worst-case situation is assessed with Equation 4. The worst-case situation is when all higher priority tasks $(d_{i,m} \leq S_{t \cdot d})$ arrive just after system has initiated a sleep state. Jobs with a deadline after the deadline of the current job will not be affected, as the slack has a shorter or equal deadline to the current job. One way to visualize the working of Equation 4 is through demand bound function. Assume a synchronous arrival of all higher priority tasks at time instant t and compute the demand bound function within an interval of $[t, S_{t \cdot d}]$. The minimum gap ϱ is the one that has the smallest distance between budget demand and utilization bound.

The minimum gap identified by ϱ using Equation 4 does not relate to the schedulability of the lower priority tasks, as it may also contain the processing time reserved for those tasks. The amount of available slack in the system gives us an exact upper bound on the sleep duration. To avoid more complex schedulability checks we cannot use more than the available slack even if the available gap ϱ is greater than the slack $\varrho \geq S_{t \cdot s}$. Conversely, if the gap ϱ is less than the system slack $\varrho < S_{t \cdot s}$, than the sleep interval of $S_{t \cdot s}$ would be obviously some higher priority task. Therefore φ finds the minimum between the available slack and the smallest-gap identified by ϱ ensuring overall system schedulability. Once a gap φ in the schedule is identified, the timer is set for an interval of $\varphi - t_n^w$.

Theorem 4. *If the job to execute in the ready queue is of BE type and the execution slack is greater than or equal to the static limit ($S_{t \cdot s} \geq t_l$) with a deadline less than or equal to the absolute deadline of the BE job ($S_{t \cdot s} \leq d_{i,m}$), the system can initiate a sleep state for φ without violating any deadlines under EDF.*

$$\varphi = \min\left(S_{t \cdot s}, \varrho\right) \tag{3}$$

Where

$$\varrho = \min_{k,m \in V(S_{t \cdot d})} \left\{ g_{k,m} - \sum_{j \in V(d_{k,m})} \left\lfloor \frac{g_{k,m}}{T_j} \right\rfloor \times C_j \right\} \tag{4}$$

$$g_{k,m} = d_{k,m} - t \tag{5}$$

$$\mathbf{V(x)} = \{i : r_{i,m} \geq t \wedge d_{i,m} \leq x\} \tag{6}$$

Proof: In this case the available sleep interval is not defined offline. To prove that a system can afford to hold the sleep state for an interval of φ, we segregated the T into four different parts. The schedulabilty of each part is proven individually.
1)$\forall j_{i,m}$ not yet released and $d_{i,m} \leq S_{t \cdot d}$
2)$\forall j_{i,m}$ not released and $d_{i,m} > S_{t \cdot d}$
3)$\forall j_{i,m}$ is in the ready queue
4)$\forall j_{i,m}$ already completed
Let ϱ define the maximum available interval by which the higher priority jobs can be delayed at the current instant t. ϱ is computed by Equation 4 considering each deadline within an interval of $[t, S_{t \cdot d}]$. In principle it performs a limited demand-bound analysis for the defined interval to calculate the delay interval. Since there is a possibility to get a delay larger than the available $S_{t \cdot s}$, Equation 3 guarantees system is not delayed more than available slack capacity. Assume a sleep interval as a task τ_{sleep} with a deadline equal to $S_{t \cdot d}$. Equation 4 implies scheduling a τ_{sleep} for not more than ϱ does not affect the schedule of any $j_{i,m}$ that is not yet released and has a $d_{i,m} \leq S_{t \cdot d}$. Moreover, we restrict that τ_{sleep} will not execute for more than $S_{t \cdot s}$ with Equation 3. This ensures that any $j_{i,m}$ not released yet with $d_{i,m} > S_{t \cdot d}$ will not be affected. Equation 6 exclude all $j_{i,m}$ such that $d_{i,m} \leq t$. Similar to our previous Theorem 3, the schedualablity of $\forall j_{i,m}$ in the ready queue is not affected as well, as they have a deadline later than that of τ_{sleep}. Any jobs already completed, are obviously unaffected. As none of the task in T miss their deadline, hence theorem holds. ■

Principle 3:
t_l is computed offline for the worst-case scenario in the schedule. Therefore initiating a sleep state for $t_l - t_n^w$ during idle mode will not affect the schedulability in any circumstance. However, the system cannot prolong the sleep state beyond the static limit. While φ could be used to increase the sleep interval, it would also substantially increase the complexity of the algorithm.

Theorem 5. *If the system is idle it can initiate a sleep state for the static limit t_l without violating the EDF schedulability, assuming the available slack S_t that may be less than the static limit t_l is consumed first.*

Proof: The proof of the Theorem 5 follows the same reasoning as given for Theorem 2. ■

The sleep transition due to any of these principles restricts the system to wake up until the timer expires. This restriction applies to all higher priority tasks as well. We assume that all interrupts bar the timer interrupt are disabled on initiating a sleep state and re-enabled on completion of the sleep. In many CPUs separate interrupt sources can be used for this. As usual with such disabled interrupts, events occurring during the sleep interval are to be flagged in the interrupt controller for processing after the interrupts are re-enabled. The complete algorithm is given in Algorithm 2.

Algorithm 2 Energy Management Algorithm
```
1: if ($System\ Idle$) then
2:    Manage Slack($t_l$)
3:    Set Sleep Time($t_l$)
4: else if ($GetSlack(j_{i,m}) \geq t_l$) then
5:    if ($HRT/SRT\ Task$) then
6:       Manage Slack($t_l$)
7:       Set Sleep Time($t_l$)
8:    else if $BE\ Task$ then
9:       Compute $\varphi$
10:      Manage Slack($\varphi$)
11:      Set Sleep Time($\varphi$);
12:   end if
13: else
14:    Race-To-Halt
15: end if
16: Set Sleep Time($\eta$)
17: ∀ Sleep States $N$:$\eta \geq t_n^e$
18: Minimize $\{(\eta * P_n) + (t_n * (P_{Ts} - P_n))\}$
19: $Timer = \eta - t_n^w$
20: Mask and record interrupts
21: Get Slack($j_{i,m}$)
22: if $d_{i,m} \geq S_{t \cdot d}$ then
23:    return $S_{t \cdot s}$
24: else
25:    return 0
26: end if
27: Manage Slack($\eta$)
28: if ($\eta \leq S_{t \cdot s}$) then
29:    $S_{t \cdot s} -= \eta$
30: else
31:    $S_t = 0$
32: end if
```

VII. Offline vs Online overhead

We compare our approach with the procrastination approach (LC-EDF) as it is with its use of dynamic priorities closest to our work. In LC-EDF, the system enters a sleep mode whenever it is idle. While in the sleep state, on each higher priority (shorter deadline) task arrival, the algorithm recomputes the new procrastination interval for that task, unless the system cannot further procrastinate. The overhead of the algorithm depends on the number of idle intervals and tasks in the system. The complexity of the procrastination algorithm is $O(p^2)$, where p is the number of tasks in the system.

LC-EDF needs an external hardware to compute its algorithm, such as an ASIC or FPGA. The external hardware is needed as the system has to compute the algorithm on every task arrival while the processor is in a sleep state. Such external hardware obviously has its own energy cost and partly negates what LC-EDF is aiming to achieve. Otherwise the system would need to transition out of and back into a sleep state on each task arrival. Considering the transition overhead (energy and time cost) associated to each sleep, makes it impractical approach from implementation perspective.

The ERTH proposed the more effective power saving algorithm with lower complexity. We discuss the complexity of ERTH in the three different cases. Firstly, the Principle 1 just needs one comparison against the offline computed t_l to initiate the sleep state. Secondly, Principle 2 requires to compute φ in order to obtain the maximum available gap to initiate a sleep state. The major overhead is the computation of ϱ that could be done offline or online. The interval for computing ϱ is no more than the longest T_i in the τ. Thus the maximum available gaps can be computed offline for each deadline and sorted in an increasing order by time. The online overhead is to search the sorted array of maximum available gaps for each given interval, which can be done in $O(ln(p))$, where p is the number of intervals. The online overhead to compute φ depends on the number of jobs in an interval. This overhead is justified when compared to its energy saving and benefits of using it. The major advantage is that it is computed only once before initiating the sleep state. Thirdly, when the system is idle (Principle 3), we use t_l to initiate a sleep state, which does not generate any overhead.

Another advantage of ERTH is the existence of fixed sleep-interval at the sleep-state initialization instant. Once the sleep state is initiated, no matter how many tasks arrive during the sleep mode, the system will wake up after a defined limit (when timer expires). We have defined bounds that ensures the schedulability of the system. This simplifies the system implementation and no external hardware is needed to compute the algorithm. The system will have less interrupt overhead when compared with the LC-EDF, as interrupts are recorded and evaluated after the timer expires and are essentially processed in batch and thus leads to fewer pre-emptions of running tasks. Conversely, in LC-EDF, the system has to respond to each interrupt at the time of its arrival.

We have also proposed some modifications in the LC-EDF that simplified the implementation and reduced its computational time. The improved form is shown in Equation 7. The system utilization is normally known offline/or can be computed once on system mode change. Thus system just need to accumulate the δ_i while in procrastination mode.

$$\Delta_l = T_l \times \left(1 - \sum_{i=1}^{n} \frac{C_i}{T_i} - \sum_{\forall i \in \mathbf{W}(l)} \frac{\delta_i}{T_i}\right) \qquad (7)$$

where $\mathbf{W}(l)$ is the set of indices of all tasks in the ready queue, that re-evaluated the LC-EDF, from its last activation.

VIII. Evaluation

A. Experimental Setup

In order to evaluate the effectiveness of our approach, we have implemented ERTH as well as the LC-EDF in a simulator. While not a fundamental requirement of our approach we assume implicit deadlines $D_i = T_i$ for our evaluation to show the effectiveness over the LC-EDF. It is obvious that $D_i > T_i$ leads to greater saving opportunities, but does not provide greater insights. All random numbers are taken from a uniform distribution and unless explicit values are given, random numbers are used for all assignments.

Task-set sizes |T|	$\{10, 50, 200\}$
Share of HRT/SRT/BE tasks $\xi = \{\xi_1, \xi_2\}$	$\{\langle 10\%, 30\%, 60\%\rangle, \langle 20\%, 40\%, 40\%\rangle\}$
Inter-arrival time T_i for RT tasks	$[30ms, 50ms]$
Inter-arrival time T_i for BE tasks	$[50ms, 1sec]$
Sporadic delay limit $\Gamma_x \in$	$\{0.1, 0.2\}$
Best-Case execution-time limit C^b	0.2
Sleep Threshold Ψ_x in	$\{1, 2, 5, 10, 20\}$

Table I
Overview of Simulator Parameters

To cover a wide range of different systems, different task sets are evaluated from a large number of fine grained small tasks (200) to a small number of coarse grained tasks (10). The two different share distributions ξ_1 and ξ_2 are applied to the number of tasks in a given class, as well as the overall utilization of the respective task classes. Moreover, utilization allocated to specific task classes is also distributed randomly among the tasks of the same class, where, for example, for ξ_2 two HRT tasks would share a total of 20% of the total system utilization claimed by all tasks. The actual individual utilization per task is generated such that the target share for each scheduling class is achieved. Starting from the utilization U_i and a minimum inter-arrival time T_i for each task according to the limits in Table I, the WCET of each task is demeed to be $C_i = U_i * T_i$.

Beyond those initial settings a two level approach is used for generating a wide variety of different tasks and subsequently varying jobs. Tasks are further annotated with a limit on the sporadic delay Δ_i^s in the interval $[0, \Gamma_x * T_i]$ and on the best-case execution time C_i^b in the interval $[C^b * C_i, C_i]$.

However, not only tasks vary in their requirements, the same task has also varying behavior dependent on system state and input parameters. This is modeled, by assigning each job $j_{i,m}$ an actual sporadic delay in the interval $[0, \Delta_i^s]$ and an actual execution time in the interval $[C_i^b, C_i]$.

As the break-even-time t_n^e does not save energy, we scale this with a factor $\Psi = \Psi_x * t_n^e$ called threshold, where $\Psi_{10} = 10$.

Overall system utilization is varied from 0.25 to 1 with an increment of 0.01. Thus in total 4560 task-sets are generated. For each task set, seed value of the random number generator is varied from 1 to 100. In total we simulated 456 thousand combinations of the above mentioned different parameters and each task set is simulated for 100 seconds.

As previously noted we have implemented both ERTH and the LC-EDF approach in our simulator. We have assumed the overhead of the ERTH and LC-EDF to be negligible. This is obviously a favorable treatment for LC-EDF as the time/energy overhead of the external specialized hardware is substantial. Our simulator takes into account the effect of the sleep state transition delays and its energy/time overhead is included in our power model.

The power model of our simulator is based on the Freescale PowerQUICC III Integrated Communications Processor MPC8536 [18]. The power consumption values are taken from its data sheet for different modes (Maximum, Typical, Doze, Nap, Sleep, Deep Sleep). As the transition overheads are not mentioned in their data sheet, we assumed the transition overhead for four different sleep states: The transition overhead of the typical mode is considered negligible. The overhead t_n^e for the four different sleep states are computed from Equation 1.

B. Results

All the parameters discussed in the experimental setup remain same, except were explicitly fixed in the experimental description or displayed as separate lines in the figures. The figures present results averaged over the 100 runs with different seed value as well as all different free parameters. As baseline we have simulated ERTH without the use of sleep states and denoted it as ERTH-WS. During execution ERTH-WS uses P_{Ts}, otherwise it consumes typical power P_{Ls}. All the results are normalized to the results of ERTH-WS.

We simulated two different scenarios. In scenario 1, we assume that $A_i = C_i$ for all task class (HRT, SRT, and BE task) and have used only one sporadic delay limit of $\Gamma_{0.1}$. In scenario 2, we assume BE tasks often overrun beyond their allocated periodic budget A_i. The mean of the BE tasks actual-execution-time distribution is set to 95% of the A_i in this scenario. However for HRT/SRT tasks, we assume $A_i = C_i$. The borrowing mechanism [16] is also integrated in scenario 2, where BE tasks may use their future budgets.

1) Scenario 1: For the first four experiments we set the minimum sleep threshold $\Psi = 1$. Figure 2 compares the total energy consumption of ERTH and LC-EDF, normalized to ERTH-WS for a task set of 200 tasks with a distribution of ξ_1. ERTH performs better than LC-EDF for all and in particular for higher utilizations. In LC-EDF, as the utilization increases, the maximum feasible idle interval (procrastination interval) estimated by the LC-EDF algorithm decreases. Thus LC-EDF cannot use at higher utilizations the more energy efficient sleep states with corresponding higher overhead t_n^e. However, our efficient slack management enables ERTH to use more efficient sleep states for accumulated slack S_t. ERTH saves energy at $U = 1$ due to the execution slack. For ξ_2 we observed a virtually identical behavior in terms of total energy consumption to that of ξ_1 in Figure 2.

The energy consumption for three different task sets is analyzed for both ERTH and LC-EDF. ERTH is insensitive to the number of tasks and the energy consumption does not vary for different task sets at same utilization. However, LC-EDF is susceptible to changes to the task-set size, as shown in Figure 3. The major reason of this variation in total energy consumption is the

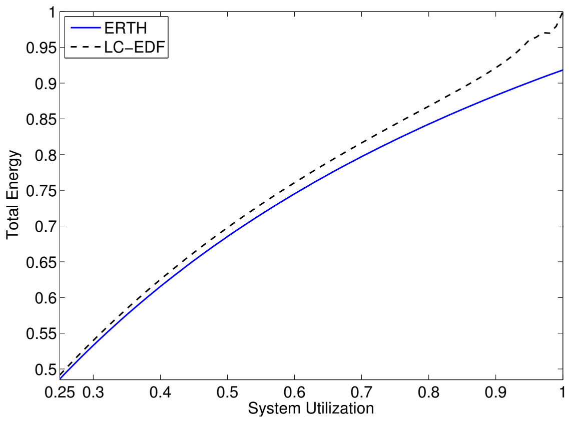

Figure 2. Total Energy Consumption (ξ_1 and $|\mathrm{T}| = 200$)

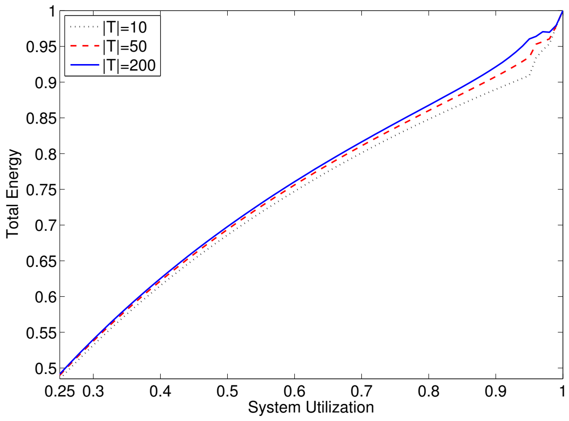

Figure 3. Effect of Task-Set Size Variation on LC-EDF (ξ_1)

algorithm, which computes the procrastination interval. As the procrastination interval is recomputed on every arrival of a job with deadline shorter than any of the currently delayed jobs, an increase in the number of tasks means a higher probability of recomputing the procrastination interval. Each recomputation includes a nominal shortening of the procrastination interval and increasing the virtual utilization in the process. Thus the procrastination interval decreases with an increase in the number of tasks.

To gauge the effect of task-set size with two different distributions (ξ_1, ξ_2), we have plotted the gain of ERTH over LC-EDF in Figure 4. As discussed, LC-EDF is sensitive to the task-set size while from our observations ERTH is not. Thus gain increases for larger task sets. As previously discussed the differences between different distributions ξ_1 and ξ_2 is small, but nevertheless observable in Figure 4. Generally the gains are larger with ξ_2 than those with ξ_1. In ξ_2 the number of BE decreases and RT tasks is increased. The BE tasks have larger periods as compared to RT tasks. In LC-EDF, the procrastination interval also depends on the period of the tasks as shown in Equation 7. More tasks with higher period, lead to a larger procrastination interval. Therefore with a decrease in BE tasks, performance of LC-EDF slightly degrades.

When the system is idle and the available interval is infeasible to initiate a sleep state, then it consumes typical power (idle power). The sleep state energy consumption shown in Figure 5 also includes the energy consumption

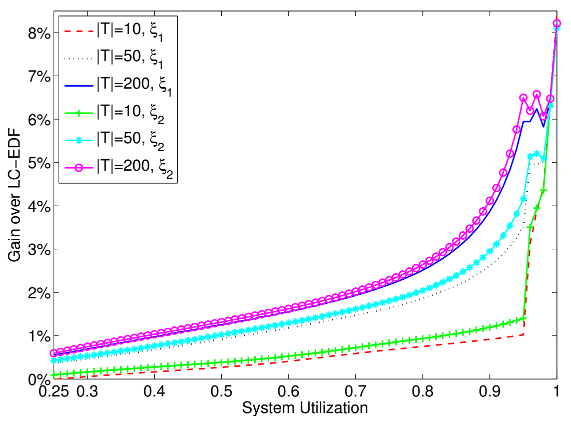

Figure 4. Gain of ERTH over LC-EDF for Different Task-Set Sizes

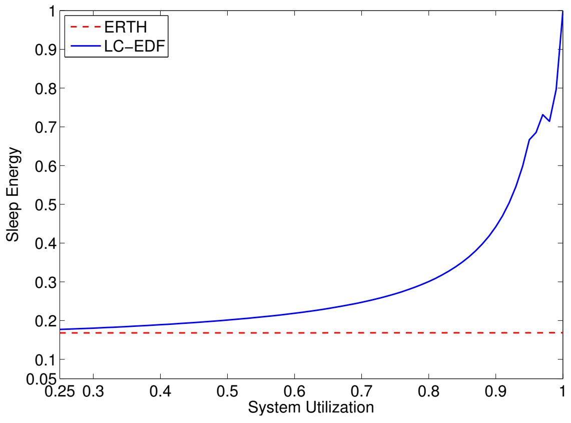

Figure 5. Sleep Energy Consumption (ξ_1 and $|\mathrm{T}| = 200$)

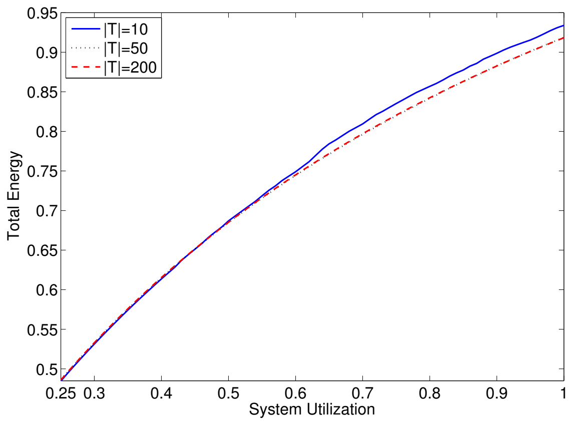

Figure 6. Effect of Extreme Threshold Ψ_{20} on Total Energy Consumption of ERTH under ξ_1

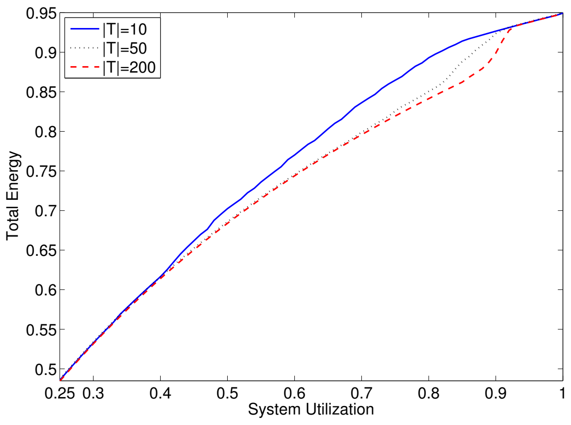

Figure 7. Effect of Extreme Threshold Ψ_{20} on Total Energy Consumption of ERTH under ξ_2

of the system when the system is idle and cannot use any sleep state. Figure 5 indicates, LC-EDF performance degrades with an increase in utilization. Among the set of available sleep states LC-EDF selects the single most efficient sleep state based on its maximum feasible idle interval. As the utilization increases, it cannot select the more efficient sleep states due to their higher transition delay t_n^e, thus the energy consumption increases. The efficient slack management is responsible for linearity of curve in ERTH, and show the stability of the proposed approach.

The effect of a higher threshold Ψ of t_n^e is studied using energy consumption of ERTH. Figure 6 and Figure 7 show the energy consumption of ERTH for a distribution of ξ_1 and ξ_2 respectively, with a threshold of Ψ_{20}. We also analyzed the results of other threshold values ($\Psi_{2,5,10}$) and found these scaling factors do not alter the energy consumption the corresponding curves overlap with that of Ψ_1. However Ψ_{20} is special case as it scales the t_n^e of some sleep states close to or above t_l and increases the dependency of ERTH on ξ and task-set size.

Firstly, consider the case of ξ_1 and Ψ_{20} (Figure 6). At a utilization of 0.6, the energy consumption of the task-set $|\mathrm{T}| = 10$ tasks is increasing. At higher utilizations t_l has decreased to such a degree, that it is less than some $\Psi_{20} * t_n^e$. This affects naturally the more efficient sleep states first. Consequently the system relies on Equation 3 which is utilized when the system switches to BE tasks. In a system with a higher number of tasks, the increased occurrences of BE tasks makes this more likely and allows for the use of more efficient sleep states. At distribution ξ_2 this behavior is more pronounced as depicted in Figure 7. This is partially motivated in the reduced share of BE tasks. The effect becomes so strong that at utilizations in excess of 0.8 even the larger task sets start to *loose* certain sleep states. In general we have observed that ERTH is not very sensitive to Ψ values except for very high values (i.e. 20) for which its performance decreases.

The effect of a higher threshold on LC-EDF is observed in Figure 8, with $|\mathrm{T}| = 50$ and ξ_1. LC-EDF use a single sleep state for each utilization and Ψ pair. Each hump in the line for the same threshold refers to a switch to a different sleep state. The vertical shift of Ψ_{20} at $U = 0.25$ shows an unavailability of the most efficient sleep state from the start. Ψ_{20} scales the t_n^e of the most efficient sleep state larger than the maximum feasible idle period. The effect of Ψ is also observed in conjugation with the number of tasks. We selected a threshold of Ψ_{10} for Figure 9, nevertheless same effect holds for different Ψ values. It is evident that tasks set with a smaller number of tasks perform better even at a higher threshold, which affirms LC-EDF strong dependency on task-set size.

2) Scenario 2: In this case BE tasks may overrun from their allocated periodic budget A_i. Both ERTH and LC-

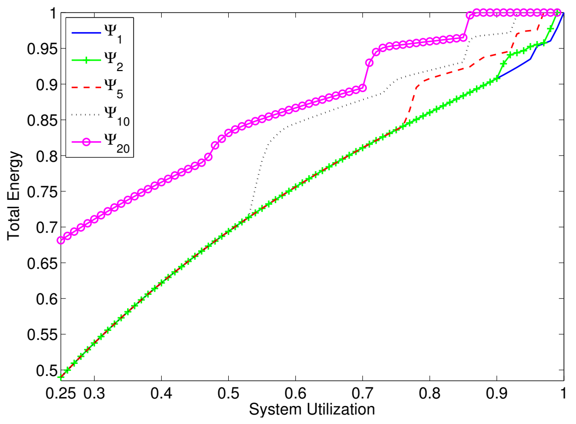

Figure 8. Effect of Threshold Change on Total Energy Consumption of LC-EDF ($|\mathrm{T}| = 50$ and ξ_1)

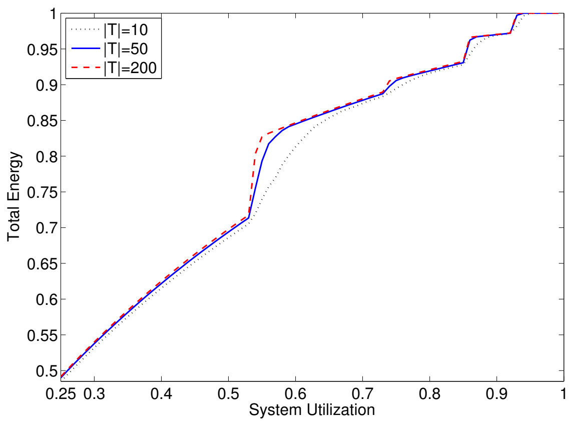

Figure 9. Effect of Elevated Threshold over Task-Set Size in LC-EDF (ξ_1 and Ψ_{10})

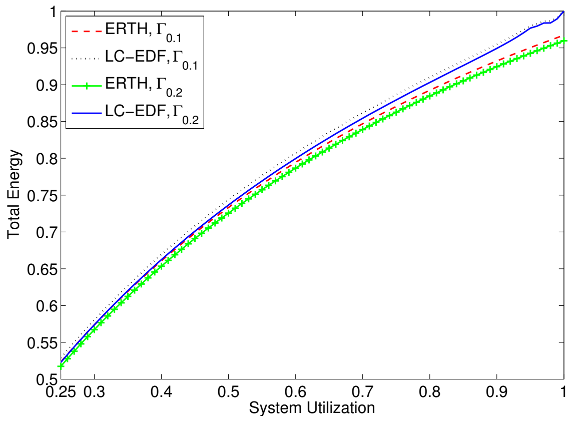

Figure 10. Total Energy Consumption in Two Sporadic Delay Limits $\Gamma_{0.1}$ and $\Gamma_{0.2}$ ($|\mathrm{T}| = 200$ and ξ_1)

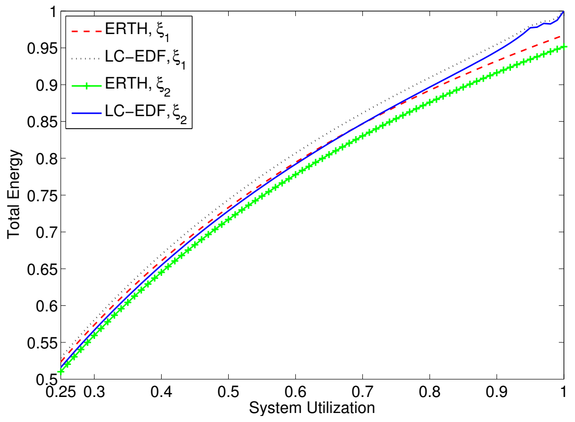

Figure 11. Total Energy Consumption with Two Distributions ξ_1 and ξ_2 ($|\mathrm{T}| = 200$ and $\Gamma_{0.1}$)

EDF have been extended to allow for the borrowing of budget from the future job releases of the same task. While it was of little consequence in scenario 1 it has to be noted that in ERTH, execution slack is not allocated to BE tasks for two reasons. Firstly, in our slack management, it reduces the priority of the execution slack, by extending its deadline, as the borrowing mechanism extends the deadline of the overrun job by one period from its current deadline. Therefore, if slack is allocated to such job, the deadline of the slack is also extended with it, its priority reduces and thus most RT tasks wont be able to utilize the slack. Secondly, the complexity of Equation 3 also increases, as the algorithm needs to compute for increased number of jobs when the deadline of the slack is longer.

The total energy consumption of the ERTH and LC-EDF is compared for two different sporadic delay limits ($\Gamma_{0.1}, \Gamma_{0.2}$) in Figure 10 for $|\mathrm{T}| = 200$ and ξ_1. $\Gamma_{0.1}$ and $\Gamma_{0.2}$ have a sporadic delay limit of 10% and 20% of T_i respectively. The widening of the sporadic delay limit means, that we are injecting more sporadic slack into the system. As sporadic slack is dealt with implicitly, the energy consumption of ERTH with $\Gamma_{0.2}$ scales down when compared to $\Gamma_{0.1}$. LC-EDF follows the same reasoning, however at higher utilization; extra sporadic slack does not help to save energy, as LC-EDF cannot use more efficient sleep states. Hence, while ERTH performs generally better than LC-EDF, at higher utilizations ERTH and LC-EDF depart extensively. The same reasoning holds for ξ_2 with the scaling down effect, due to the reason given below.

We observed the energy consumption of two different distributions (ξ_1,ξ_2) with $|\mathrm{T}| = 200$ and $\Gamma_{0.1}$ in Figure 11. In ξ_2 percentage of the BE tasks in a task-set is reduced to 40% and consequently also reduced borrowing, which provides more slack for energy management. Thus in Figure 11, the energy consumption scales down for ξ_2 compared to ξ_1. Nevertheless ERTH outperforms LC-EDF in both distributions (ξ_1,ξ_2), even with the borrowing mechanism integrated. The energy consumption scales down, when the same experiment is done with $\Gamma_{0.2}$ due to extra sporadic slack in the system.

Figure 12 analyzes the gain of ERTH over LC-EDF for different sporadic delay limits ($\Gamma_{0.1}$, $\Gamma_{0.2}$), with three task sets ($|\mathrm{T}| \in \{10, 50, 200\}$ and ξ_2. Though both approaches implicitly manage sporadic slack, ERTH performs slightly better than LC-EDF, especially at higher utilization for large task sets. This small gain of $\Gamma_{0.2}$ over $\Gamma_{0.1}$ shows efficient implicit use of sporadic slack in ERTH. We have also explored this with the distribution ξ_2 and the results indicated that at utilizations close to 1, the performance gain of ERTH is slightly less for all different task sets; about 0.5% when compared to Figure 12. For smaller utilizations the difference is less pronounced. This is a

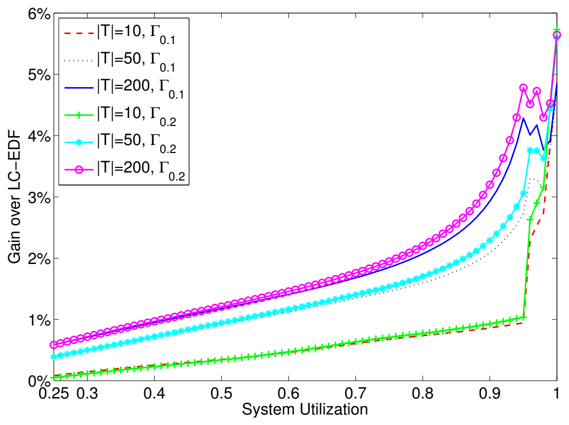

Figure 12. Gain of ERTH over LC-EDF (ξ_2)

function of the reduced number of BE tasks in ξ_2 and the consequently smaller amount of borrowing.

Figure 12 illustrates the gain of ERTH over LC-EDF for the distribution ξ_1 and $\Gamma_{0.1}$ and comparing that to Figure 4 one can notice the reduced gains returned when borrowing. Generally, the gain of scenario 2 compared to scenario 1 is less at higher utilizations, but approximately the same at lower utilizations. The gain rises exponentially in Figure 4, Figure 12 after $U = 0.8$ for large task sets.

The sleep state energy consumption of the scenario 2 is approximately similar to the scenario 1. Similarly, the higher threshold effect in scenario 2 is also identical to scenario 1 for LC-EDF and ERTH, but energy consumption, scales up linearly in scenario 2. The scaling up effect is due to an increased in execution-time requirement of the BE tasks that habitually overrun. For different combinations of ξ and Γ, a scaling up effect occurs in following ascending order $(\xi_2, \Gamma_{0.2})$, $(\xi_2, \Gamma_{0.1})$, $(\xi_1, \Gamma_{0.2})$ and $(\xi_1, \Gamma_{0.1})$. An increase in sporadic delay limit injects more sporadic slack to the system (therefore saving more energy) and vice versa. Similarly more borrowing consumes extra energy and vice versa. Thus with the highest sporadic delay limit and minimum borrowing $(\xi_2, \Gamma_{0.2})$ the energy consumption is least in scenario 2, whilst with least sporadic delay limit and most borrowing $(\xi_1, \Gamma_{0.1})$ energy consumption is maximized for different threshold values ($\Psi_1, \Psi_2, \Psi_5, \Psi_{10}$ and Ψ_{20}) and task sets ($|\mathrm{T}| \in \{10, 50, 200\}$).

IX. Conclusions and Future Directions

We presented an efficient energy saving approach for dynamic priority systems based on sleep states and eliminated unrealistic assumptions made in state-of-the-art research. Furthermore we reduced the online complexity of the system when compared to the original proposed approach. Our approach exploits execution slack and static slack explicitly, as well as sporadic slack implicitly using an efficient slack management approach. For future research, we intend to extend the approach to multicore processors, as well as demonstrating its effectiveness on real hardware. While our approach computes the sleep interval assuming synchronous release of all higher priority tasks, the sleep interval could be improved exploiting knowledge of past releases of all tasks. The integration of DVFS is an additional opportunity to save more energy. Finally, we want to explore the effect of ERTH on the number of preemptions and thus improving on required reservations.

References

[1] E. Le Sueur and G. Heiser, “Dynamic voltage and frequency scaling: The laws of diminishing returns,” in *2010 HotPower*, (Vancouver, Canada), Oct 2010.

[2] S. M. Petters and M. A. Awan, “Slow down or race to halt: Towards managing complexity of real-time energy management decisions,” in *12th WTR*, (Gramado/RS, Brazil), May 2010. Work-in-Prog. Session.

[3] Y.-H. Lee, K. Reddy, and C. Krishna, “Scheduling techniques for reducing leakage power in hard real-time systems,” in *15th ECRTS*, pp. 105 – 112, jul. 2003.

[4] S. Irani, S. Shukla, and R. Gupta, “Algorithms for power savings,” in *14th SOSP*, (Baltimore, Maryland), pp. 37–46, Soc. for Industrial & Appl. Mathematics, 2003.

[5] L. Niu and G. Quan, “Reducing both dynamic and leakage energy consumption for hard real-time systems,” in *CASES*, (Washington DC, USA), pp. 140–148, ACM, 2004.

[6] R. Jejurikar, C. Pereira, and R. Gupta, “Leakage aware dynamic voltage scaling for real-time embedded systems,” in *41st DAC*, (San Diego, CA, USA), pp. 275–280, ACM, 2004.

[7] R. Jejurikar and R. Gupta, “Procrastination scheduling in fixed priority real-time systems,” in *LCTES’04*, (Washington DC, USA), 2004.

[8] J.-J. Chen and T.-W. Kuo, “Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor,” *SIGPLAN Notices*, vol. 41, pp. 153–162, June 2006.

[9] R. Jejurikar and R. Gupta, “Dynamic slack reclamation with procrastination scheduling in real-time embedded systems,” in *42nd DAC*, (Anaheim, California, USA), pp. 111–116, 2005.

[10] J.-J. Chen and L. Thiele, “Expected system energy consumption minimization in leakage-aware dvs systems,” in *ISLPED*, (Bangalore, India), pp. 315–320, ACM, 2008.

[11] V. Devadas and H. Aydin, “On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications,” in *8th EMSOFT*, (Atlanta, GA, USA), pp. 99–108, ACM, 2008.

[12] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson, “Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes,” in *24th RTSS*, (Cancun, Mexico), Dec 2003.

[13] S. K. Baruah, L. E. Rosier, and R. R. Howell, “Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor,” *J. Real–Time Syst.*, vol. 2, pp. 301–324, 1990.

[14] R. Pellizzoni and G. Lipari, “Feasibility analysis of real-time periodic tasks with offsets,” *J. Real–Time Syst.*, vol. 30, pp. 105–128, 2005.

[15] A. Rahni, E. Grolleau, and M. Richard, “Feasibility analysis of non-concrete real-time transactions with edf assignment priority,” in *16th RTNS*, Oct 2008.

[16] C. Lin and S. A. Brandt, “Improving soft real-time performance through better slack management,” in *26th RTSS*, (Miami, FL, USA), Dec 2005.

[17] S. M. Petters, M. Lawitzky, R. Heffernan, and K. Elphinstone, “Towards real multi-criticality scheduling,” in *15th RTCSA*, (Beijing, China), Aug 2009.

[18] FreeScale Semiconductor, *MPC8536E PowerQUICC III Integrated Processor Hardware Specifications*. Document Number: MPC8536EEC, Rev 3, Nov 2010.