IPP HURRAY!

www.hurray.isep.ipp.pt

Technical Report

Online Intra-Task Device Scheduling for Hard Real-Time Systems

Muhammad Ali Awan

Stefan M. Petters

HURRAY-TR-120602

Version:

Date: 06-05-2012

Online Intra-Task Device Scheduling for Hard Real-Time 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

A large part of power dissipation in a system is generated by I/O devices. Increasingly these devices provide power saving mechanisms, inter alia to enhance battery life. While I/O device scheduling has been studied in the past for real-time systems, the use of energy resources by these scheduling algorithms may be improved. These approaches are crafted considering a very large overhead of device transitions. Technology enhancements have allowed the hardware vendors to reduce the device transition overhead and energy consumption. We propose an intra-task device scheduling algorithm for real time systems that allows to shut-down devices while ensuring system schedulability. Our results show an energy gain of up to 90% when compared to the techniques proposed in the state-of-the-art.

Online Intra-Task Device Scheduling for Hard Real-Time Systems

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

***Abstract*—A large part of power dissipation in a system is generated by I/O devices. Increasingly these devices provide power saving mechanisms, inter alia to enhance battery life. While I/O device scheduling has been studied in the past for real-time systems, the use of energy resources by these scheduling algorithms may be improved. These approaches are crafted considering a very large overhead of device transitions. Technology enhancements have allowed the hardware vendors to reduce the device transition overhead and energy consumption. We propose an intra-task device scheduling algorithm for real time systems that allows to shut-down devices while ensuring system schedulability. Our results show an energy gain of up to 90% when compared to the techniques proposed in the state-of-the-art.**

I. Introduction

Real-time (RT) embedded systems have to perform a set of functions while adhering to additional timing constraints. These systems interact with their environment and hence use I/O devices. Typical domains in which such systems are deployed includes avionics, automotive electronics and control systems. Besides the timing constraints many RT systems have limited or intermittent power supply. Therefore, energy efficiency is another important aspect that needs to be considered in the design process of such systems.

The demand of extra functionality on a single embedded system also results in an increased number of I/O devices. As I/O devices consume considerable amount of energy and are of particular concern in mobile systems, they often equipped with (a) power saving state(s) to minimise this energy consumption. A device can only operate in the active mode, and its transition into and out of sleep state incurs both time and energy overheads. Moreover, the request instant and access interval of the device can usually not be determined beforehand. In order to guarantee the temporal correctness of such RT system, the device transition delay to bring the device up from sleep needs to be taken into account.

Considering the uncertainty in the device usage instant during program execution, traditional device-scheduling algorithms made a safe but pessimistic assumption that the device will be used during the entire execution time of the corresponding job. Therefore, a device active state is ensured during the entire execution of the job. This category of device scheduling is known as inter-task device scheduling. However, most devices are used for very short intervals of time thus resulting in wasted energy.

In contrast to this, in intra-task device scheduling a device is only turned on when it is requested by the job. To the best of our knowledge, online intra-task device scheduling for hard real-time systems has not been explored in the state-of-the-art techniques. Our algorithm explores online intra-task device scheduling in a hard-real time systems setting, based on a sporadic task model. The objective of this research is to utilise the spare processing resources of the system to reduce the energy consumption of the devices by allowing them to wake-up on demand rather than in a predictive manner.

The contributions of this paper are: 1) the computation of spare capacity in the schedule (slack management algorithms); 2) an intra-task device scheduling algorithm that utilises the collated slack in the schedule and wake-up the devices on demand to enhance the energy performance; 3) an online mechanism to reclaim unused time allocations to devices in the system; 4) a complexity comparison of our algorithm with the state-of-the-art.

The rest of the paper is organised as follows. The next section discusses the related work and is followed by our system model. Our static slack container algorithm (SSC) and corresponding slack management are described in Section IV. Section VI compares the complexity of the state-of-the-art with SSC. Finally, we present the evaluation and conclude with future directions.

II. Related Work

Initially the device power management was extensively studied in a non-real-time setting. These techniques can be divided into three main categories, 1) time-out based, 2) predictive and 3) stochastic. Time-out based algorithms shut-down the devices when they are idle for the specified threshold. The system wakes up the device on the next request of the task. Predictive techniques adopt themselves with the varying workload of the system. Stochastic methods model the requests behaviour with different probabilistic distributions. The device shut-down times are estimated by solving the stochastic models such as Markov chains. For a detailed survey of device power management algorithms in a best effort environment, the reader is directed to the work of Benini et al. [1].

Swaminathan et al. [2] proposed an offline method for dynamic I/O power management with hard real-time constraints. Their low energy device scheduler (LEDES) is based on look-ahead information of the future task-arrival pattern to shut-down the devices. It requires fixed task releases, which limits its applicability in case of a sporadic task model or task-sets with variable execution time. Later on, multi-state constrained low-energy scheduler (MUSCLES), an extension of LEDES for the multiple state devices was proposed by Swaminathan and Chakrabarty [3].

The same authors also developed another energy optimal device scheduler (EDS) [4]. EDS computes a schedule tree for all possible scheduled combination, and prune it based on temporal and energy constraints. Due to high spatial requirement and temporal complexity of EDS, they provide a heuristic which clusters the requests to the same devices to prolong the idle intervals. It is based on the work of Lu et al. [5] that was proposed for best-effort systems. Both heuristic and EDS are based on an inter-task scheduling mechanism, but are computationally expensive and are of limited utility for sporadic task models.

A procrastination based I/O device scheduling algorithm is proposed by Cheng and Goddard [6]. The basic idea is to prolong the device's sleep interval by procrastination of the task execution that requires this device. This method assumes inter-task device scheduling and is computationally expensive. However, it can be applied to a sporadic task model with varying execution times. Later, Devadas and Aydin et al. [7] proposed the device power management algorithm for static priority systems through device forbidden regions. It is based on inter-task device scheduling and enforces idle intervals in the schedule to prolong the device sleep interval. To preserve the schedulability, the bounds on the explicit idle intervals are computed using time bound analysis [8].

Isolation of device power management from CPU power management gives system-wise sub-optimal solutions. Cheng and Goddard [9] integrated device scheduling, and dynamic voltage and frequency scheduling (DVFS). Their approach predicts the device usage times based on future release patterns and accordingly sets timers to initiate the respective device wake-up. DVFS increases the execution time of tasks to reduce dynamic power consumption and consequently, prolongs the active time of the devices as well. The approach aimed to select the processor frequency that minimises the overall energy consumption, however, several strong simplifications of the DVFS model limit the applicability of their work.

The system-level power management algorithm developed by Devadas and Aydin [10] for the frame-based embedded systems similarly addresses the interplay of DVFS and the device power management. Their work finds the optimal frequency set-point for the processor that minimises the energy consumption. While their approach is promising in principle, deficiencies of the power model and the restriction to frame-based tasks would require further work.

The device schedulers proposed for hard real-time systems in the literature assume inter-task device scheduling and are based on unrealistic assumptions such as exact future release information, a simplistic power model as well as consider the device transitions overhead to be very large. Our proposed algorithm has lower complexity and saves more energy while eliminating unrealistic assumptions when compared to the state-of-the-art techniques. It is based on a different paradigm of intra-task device scheduling, which was previously not investigated due to the large overheads of device sleep transitions.

III. System Model

We assume a hard-real time system containing a workload comprised of a set of sporadic tasks each using an individual peripheral device. The task set Γ consists of l independent tasks i.e. $\Gamma = \{\tau_1, \tau_2, \tau_3, \cdots, \tau_l\}$. A task τ_i is specified by a quadruple $\langle C_i, D_i, T_i, \lambda_i \rangle$, where C_i is the worst-case execution time (WCET), D_i is the relative deadline, T_i is the minimum inter-arrival time and λ_i is the device used by the task τ_i.

We used the Rate-Based Earliest Deadline first (RBED) framework [11], which provides temporal isolation via enforced budgets. This temporal isolation allows for mixed criticality workloads consisting of hard RT, soft RT and best-effort type applications. The tasks are allocated a budget of A_i and released as sequence of jobs $j_{i,m}$. Each job $j_{i,m}$ has an absolute deadline $d_{i,m}$, a budget $a_{i,m}$, a release time $r_{i,m}$ and an actual execution time $\hat{c}_{i,m}$.

In the original RBED work [11], the allocations of the budget for a soft RT (SRT) or best-effort (BE) task is less than or equal to their WCET ($A_i \leq C_i$). Hard RT (HRT) tasks are allocated a budget equal to their WCET ($A_i = C_i$), to ensure the timely completion of their jobs. However, in this paper, we assume HRT and SRT tasks are allocated a budget of $A_i = C_i$. The scheduler pre-empts every job when it has used up its allocated budget $a_{i,m}$. Thus $j_{i,m}$ exceeding its budget cannot affect the overall system schedulability.

Each device λ_i which is associated with exactly one task (no sharing) is characterised with the following parameters: the active mode power consumption; the sleep state power consumption; the energy consumption during the state transition phase; and the transition delay t_i^{tr} of the state switch. A state transition can be from active to sleep mode or vice versa. For the sake of simplicity, we assume the energy consumption and the transition delay of λ_i is the same for transitioning into or out of a sleep state. A complete transition-phase delay, i.e. from active to sleep and sleep to active mode, of λ_i is denoted as $t_i^{sw} = 2t_i^{tr}$.

The break-even-time t_i^{be} of λ_i is the amount of time a system needs to compensate for the energy lost during the transition phase. The definition and the measurement technique used for t_i^{be} follows from the work of Cheng and Goddard [6]. Each device has only a single sleep state, but this model can be easily adapted for the devices with multiple sleep states. A transition is only initiated in stable state, i.e. active or sleep state. In our algorithm, we assume a device is used once during the job execution, but the exact time instances of the device usage along with its duration within a job's execution is unknown.

IV. Slack Management

In intra-task device scheduling, a device is only woken-up on demand to reduce its active time and consequently the energy consumption. However, the transition time imposes an extra overhead and alters the system schedule, as the task has to wait for a device to become active. Additionally, inserting extra wake-up calls ahead of the device usage into the application code is impractical. A slack management algorithm in the scheduler is needed to collate the idle intervals. Before going into the details of our algorithm, we briefly define the sources of slack in the system.

The unused processing time in a system is called slack. System slack can be categorised as static, execution and sporadic slack. Static slack exists due to spare capacity in the system, which is not loaded with what could be guaranteed by the schedulability test. Execution slack (E^s) comes from the difference of the WCET and the actual execution time, as the RT tasks mostly execute for less than their WCET and subsequently the allocated budget. Finally, sporadic slack is an extra arrival delay beyond the minimum inter-arrival time T_i assumed in the analysis.

A. Device Budget

Definition 1: The device budget D_b of the system is the maximum available spare time in the schedulability test that can be used to compensate for the devices transition delays without causing any application to miss its deadline under worst-case assumptions.

The device budget comes from the static slack of the system. A lower bound on the size of the device budget is determined by considering the temporal correctness of the system. The used RBED framework is based on the Earliest Deadline First algorithm (EDF). The schedulability analysis of the EDF on uniprocessor [12], [13] is presented in the Theorem 1. However, the overall demand bound function for a task-set Γ can be represented as $dbf_\Gamma(L) \stackrel{\text{def}}{=} max_{L_0}\ df(L_0, L_0 + L)$ by following the definition of Rahni et al. [14], where L_0 is a time instant.

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.

Formally, the device budget is defined by exploiting the demand bound function dbf. Considering Theorem 1, a lower bound on the device budget D_b is given in Equation 1, where L is an absolute deadline and L^* is the first idle time in the schedule.

$$\forall L \in L^*, D_b = \min\left(L - dbf(L)\right) \tag{1}$$

A similar definition is used in our previous work [15], however the objective was to use this static limit to find the maximum feasible sleep interval. For the proof, the reader is referred to [15].

B. Execution Slack

We also exploit the execution slack E^s explicitly to use it along with the device budget D_b. Consumption of the sporadic slack is implicit within our algorithm and will be explained in later sections. When execution slack E^s is generated, it is identified with a size E^s_{sz} and a corresponding deadline E^s_{dl}. The algorithm to manage E^s is adapted from [15], [16]. The basic idea is to keep E^s in a central container. If the deadline of the job $j_{i,m}$ is greater than E^s_{dl} (i.e. $E^s_{dl} < d_{i,m}$), then the deadline of E^s is extended to the deadline of $j_{i,m}$, i.e. $E_{dl} = d_{i,m}$. This algorithm is simple in both spatial and temporal terms.

V. Algorithm

A. Shut Down and Wake Up

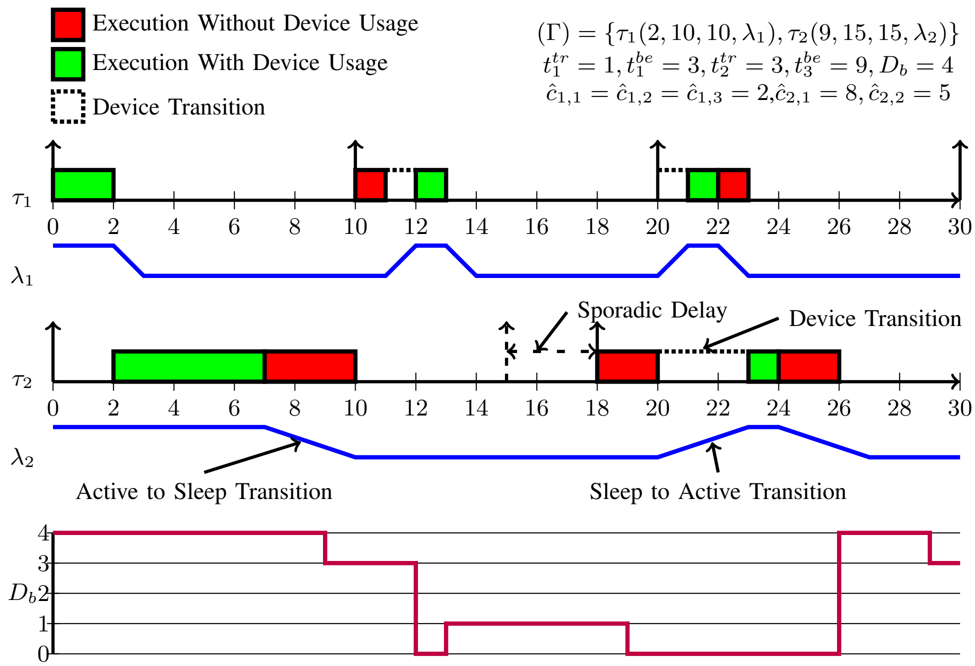
A general sketch of the Static Slack Container algorithm (SSC) that utilises D_b and E^s to save the energy consumption by switching off unused I/O devices is shown in Algorithms 1,2. Some of the notations used throughout this section are given below.

- Φ: A set of intra-task device-scheduling compatible devices.
- γ_i: The next utilisation time of any device λ_i. This value is the next expected release time of the job $j_{i,m}$ using device λ_i. It is computed with reference to the previous release information.
- SSSR: The Static Slack Services Register contains references to the inactive devices that acquired their wake-up budget from D_b.
- Ξ_i: The amount of pending workload of high priority than a job $j_{i,m}$ currently residing in the ready queue.
- ESSR: The Execution Slack Serviced Register contains references to the inactive devices prolonging their wake-up time with the execution slack.
- HPW : The workload of higher priority tasks than τ_i executed in the device λ_i transition interval.
- LPW : The workload of lower priority tasks than τ_i executed in the device λ_i transition interval.
- IPW : Intermediate Priority Workload.
- $s_{i,k}$: Start time of t^{tr}_i of λ_i for $j_{i,k}$.

An example in Figure 1 visualises the different concepts of Algorithms 1 and 2. The task-set is composed of two tasks $\Gamma = \{\tau_1(2, 10, 10, \lambda_1), \tau_2(9, 15, 15, \lambda_2)\}$. λ_1 has a transition delay $t^{tr}_1 = 1$ and a break-even-time $t^{be}_1 = 3$. Similarly, λ_2 has $t^{tr}_2 = 3$ and $t^{be}_2 = 9$. Moreover, the given task-set has $D_b = 4$. All the jobs of τ_1 executes for 2 time units. The first job $j_{2,1}$ of τ_2 executes for 8 time units, and second job $j_{2,2}$ executes for 5 time units.

1) Offline Phase: Initially, a system identifies the set of devices compatible with the intra-task device scheduling Φ by using Theorem 2. Non-compatible devices can be shut-down, only in a case, when their corresponding tasks execute for less than their C_i and $D_i - \hat{c}_i \geq t^{sw}_i$. However, a system needs to ensure the wake-up of non-compatible devices before their corresponding task starts its execution. In the given

Fig. 1: Example



example shown in Figure 1 all the devices are intra-task device-scheduling compatible.

Theorem 2: A device λ_i associated to a task τ_i will be compatible with intra-task device scheduling if and only if the overhead of the device t_i^{sw} plus the τ_i's WCET C_i is less than or equal to its relative deadline D_i.

Proof: A proof of this theorem is trivial. Task τ_i using a device λ_i will miss its deadline, if the condition $t_i^{sw} + C_i > D_i$ is true; assuming λ_i started its transition on τ_i release time, woken-up on demand and completes its transition once initiated. ■

2) Scheduling in Static Slack Container: On a system boot up usually all the devices are in active mode. If a running $j_{i,m}$ requests the associate device λ_i and λ_i is in active mode, the job will continue its execution. In our example, Figure 1 $j_{1,1}$ and $j_{2,1}$ finds their corresponding devices on and continue their execution. However, if λ_i is in sleep mode, then $j_{i,m}$ is preempted and inserted into the device waiting queue. Once an interrupt service routine signals that λ_i is ready, $j_{i,m}$ is enqueued again in the ready queue and scheduled according to its priority. For instance, $j_{1,2}, j_{1,3}$ and $j_{2,2}$ in Figure 1 wait for their corresponding devices to transition out of their sleep state and use them in the interval $[12; 13], [21; 22]$ and $[23; 24]$ respectively.

3) Device Shut-Down: Once $j_{i,m}$ has completed its use of λ_i, the scheduler tries to shut it down. If $\lambda_i \in \Phi$, the algorithm takes a conservative approach and performs the shut down when the difference of next utilisation time γ_i of λ_i and current time instant t is greater or equal to its total transition delay t_i^{sw} (i.e. $\gamma_i - t \geq t_i^{sw}$). We considered t_i^{sw} instead of t_i^{be} to exploit the sporadic slack. A timer is set accordingly to wake up the device. In the given example (Figure 1), all the jobs of τ_1 have enough time to shut-down the devices. $j_{2,1}$ completes its device related execution at time instant 7 and has a difference of 8 time units from its next utilisation time γ_2 of 15, which is less than its t_2^{be}. However, λ_2 initiates a sleep transition with an expectation that next job arrival will be delayed due a sporadic slack and the total sleep duration will be more than t_2^{be}. Similarly, $j_{2,2}$ has $\gamma_i - t = 8 > t_i^{sw}$ and it initiates a sleep transition based on the same reasoning given for $j_{2,1}$.

On the other hand, this condition ($\gamma_i - t \geq t_i^{sw}$) may not be applied to $\lambda_i \notin \Phi$, as the system needs to ensure that λ_i should be active before its γ_i. Nevertheless, we can still shut-down these devices with a condition that the jobs related to these devices execute less than their C_i and $\gamma_i - t > t_i^{be}$.

4) Device Wake-up: The algorithm's main objective is to extend the sleep interval of the devices already in sleep mode. Whenever, a timer associated to any $\lambda_i \in \Phi$ expires, the system considers spare resources such as device budget D_b or the execution slack E^s to prolong the currently inactive device λ_i. However, this process to prolong the sleep interval of λ_i is not considered for $\lambda_i \notin \Phi$, for which a timer expiration triggers a process to activate the device without any further delay. As mentioned previously in Section IV, D_b is a major source of slack used to extend the sleep interval of the devices. Once a timer associated to any $\lambda_i \in \Phi$ lapse, the system firstly tries to utilise D_b. $D_b \geq t_i^{tr}$ allows to further procrastinate the device activation process, while ensuring the system schedulability. Consequently, t_i^{tr} time is deducted from D_b and λ_i is registered in a special register called static slack serviced register (SSSR). SSSR holds those inactive devices that acquired their wake-up budget from D_b. In our example of Figure 1, timers associated to $j_{1,1}, j_{1,2}, j_{1,3}$ and $j_{2,1}$ expires at time instances $9, 19, 29$ and 12 respectively. For $j_{1,1}, j_{1,2}$ and

Algorithm 1 Static Slack Container Algorithm

Offline Phase
Separate the intra-task scheduling compatible devices from the not compatible one's.
$\Phi = \{\lambda_j : D_j - C_j - t_j^{sw} \geq 0\}$
Calculate the Device Budget D_b for a given task-set Γ
The device λ_i requested by job $j_{i,m}$ in the waiting queue wakes up
Move $j_{i,m}$ from the device waiting queue to the ready queue
if ($j_{i,m}$ on the head of the list) **then**
  Reschedule
end if
Next Utilisation Time(γ):
On release of τ_i: Update τ_i next predicted arrival time in the future release array r^n i.e.
$r_i^n = \gamma_i = r_{i,m} + T_i$.
Device Shut-Down Procedure:
When $j_{i,m}$ has used λ_i, consider the following criteria to shut-down and set the corresponding entry in the sorted list of timer.
if ($\lambda_i \in \Phi$) **then**
  if ($\gamma_i - t \geq t_i^{sw}$) **then**
    Shut-down the device
    Timer = $\gamma_i - t_i^{tr}$
  else
    Keep the device on
  end if
else
  if ($\gamma_i - t > t_i^{be}$) **then**
    Shut-down the Device
    Timer = $\gamma_i - t_i^{tr}$
  else
    Leave the Device On (Otherwise we cannot guarantee the schedulability)
  end if
end if

$j_{1,3}$ we deduct D_b equal to 1 time unit and extend their sleep state unless they are requested again from the subsequent jobs. Similarly, $j_{2,1}$ deducts 3 time units from D_b at time instant 12 (3 time units before its γ_2) and keep the device in sleep mode unless requested again.

In case $D_b < t_i^{tr}$, the system relies on the E^s. λ_i associated to $j_{i,m}$ is eligible for E^s if and only if $d_{i,m}$ of $j_{i,m}$ that will utilise λ_i in the future is greater than or equal to the deadline of the execution slack E_d^s. $d_{i,m}$ of $j_{i,m}$ not released yet can be conservatively predicted by considering its past release information and T_i. The duration of the pending high-priority workload compared to $j_{i,m}$ that currently resides in the ready queue Ξ_i is added to compute the total interval for the device to shut-down. The next wake up time is set to $E_{sz}^s - t_i^{tr} + \sum_{\tau_i \in \Xi_i} C_i$ and the corresponding device is registered in ESSR. A high priority workload from the future can also

Algorithm 2 Static Slack Container Algorithm (Continue)

Device Wake-up Procedure:
When the initial timer to wake-up λ_i expires.
if ($\lambda_i \in \Phi$) **then**
  if ($t_i^{tr} \leq D_b$) **then**
    $D_b = D_b - t_i^{tr}$
    Keep the device off and register its entry in the SSSR
  else if ($E_{sz}^s > t_i^{tr} \&\& E_{dl}^s \leq d_{i,m}$) **then**
    Where $d_{i,m}$ is the deadline of the job $j_{i,m}$ that will require λ_i in future.
    if Ξ_i **then**
      Register the device in ESSR
      Timer = $E_{sz}^s - t_i^{tr} + \sum_{\tau_i \in \Xi_i} C_i$
    else
      Timer= $E_{sz}^s - t_i^w$
    end if
  else
    Wake-up the device
  end if
else if ($\lambda_i \notin \Phi$) **then**
  Wake-up the device
end if
Device Budget D_b Replenishment:
if Ready Queue Empty && Device Waiting Queue Empty **then**
  D_b = Initial Value of $D_b - \sum_{i \in SSSR} t_i^{tr}$
end if

be included but it will increase the online complexity of the algorithm.

In case $\{(D_b < t_i^{tr}) \&\& (E_{dl}^s > d_{i,m} || E_{sz}^s < t_i^{tr})\}$, one can also consider only the high priority workload in the ready queue and from the future. However, the computation of high priority workload increases the overhead and is avoided in our algorithm for the sake of simplicity.

Theorem 3: The schedulability of the system with EDF will be preserved if the replenishment equal to $D_b - \sum_{i \in SSSR} t_i^{tr}$ happens in idle mode when the ready queue along with the device waiting queue is empty.

Proof: This theorem claims if the following two properties are satisfied system schedulability will be preserved. 1) Replenishment should be done in an idle mode when the waiting queue is empty. 2) It should be equal to $D_b - \sum_{i \in SSSR} t_i^{tr}$.

1) The idle mode combined with no job in the waiting queue equates in the worst case to the consideration of the critical instant, resulting in D_b units of time being available at any point in the schedule without violating schedulability of the system.

2) Since tasks in the SSSR have already reserved their share of device budget equal to $\sum_{i \in SSSR} t_i^{tr}$. Would this not be considered, time reserved by tasks in SSSR would be allocated a second time to other tasks, potentially leading to a deadline violation. Hence a replenishment of $D_b - \sum_{i \in SSSR} t_i^{tr}$ maintains a schedulability. ■

The replenishment of D_b in an example Figure 1 is done according to the criterion defined in Theorem 3 at time instances 13 and 26. On the first time instant at 13, λ_2 is still in sleep mode and previously registered its entry in SSSR at time instant 12, therefore, D_b is only replenished with a budget equal to its initial value minus the device transition delay t_2^{tr} of λ_2 (i.e. $4-3=1$). However, at time instant 26 both the devices are in sleep mode but not registered in SSSR, hence, D_b is replenished with a budget equal to its initial value of 4.

B. Device Budget Reclamation Algorithm

The device budget D_b is a precious resource in our algorithm; therefore, we have also proposed the device budget reclamation algorithm given in Algorithm 3. D_b is only reclaimed from the devices having entry in the SSSR; i.e. devices allocated a device budget equal to their transition delay. All devices discussed onwards in this section assume their entry in SSSR, otherwise a device is not considered for reclamation. D_b by definition is the highest priority budget in the system. When $j_{i,m}$ is allocated a part of D_b to compensate for its device transition, analysis assumes this additional budget will be consumed by $j_{i,m}$ as a part of execution. This assumption is made for a case when there is no other job executing and/or waiting for its device transition during this interval. When there actually is another job executing or waiting for its device, the device budget may be reclaimed depending on the priority of the workload executed in this interval. For instance in Figure 1, within an interval of [20; 21] device transition time of both devices (λ_1 and λ_2) overlaps, similarly, in an interval of [21; 22] execution of $j_{1,3}$ overlaps with the transition of λ_2. These two scenarios are discussed below in details. However, a reclaimed budget is not added back in the given example (Figure 1) for the ease of presentation.

1) Device Overlap: In this scenario multiple jobs are waiting for their device active state. It is evident that a system should only consider a budget consumption of single device in the overlapping period as their wake-up transition happens in parallel. Line 8 in Algorithm 3 reclaims such budget. Suppose $s_{i,k}$ is the transition start time of device λ_i requested by $j_{i,k}$. Assume $j_{i,k}$ is the first job that requested the device at $s_{i,k}$. All jobs excluding $j_{i,k}$ that have their entry in the SSSR and request λ_j after $s_{i,k}$ and before $s_{i,k}+t_i^{tr}$ has overlap with $j_{i,m}$ device transition and are entitled for slack reclamation. The device budget of 1 time unit can be reclaimed at time instant [20; 21] in Figure 1 as λ_1 and λ_2 device transition overlap, and the scheduler should only consider a transition delay of one device.

2) Execution Overlap: Assume a job $j_{i,m}$, currently waiting for the device transition to active state. $j_{i,m}$'s device transition interval is denoted as $[t_1, t_2]$. In this scenario, we explore an overlap of $[t_1, t_2]$ with the execution of other jobs. The execution overlap is divided into two types based on it priority compared to the priority of $j_{i,m}$, i.e. high or low priority workload. The workload having priority equal to $j_{i,m}$ can be considered as a part of a high priority workload. If a

Algorithm 3 Device Budget Reclamation Algorithm

1: $[t_1, t_2]$: Device λ_i transition Interval
2: **if** (HPW $\cap$ $[t_1, t_2]$) **then**
3: $\quad D_b+=t_i^{tr}-$(HPW $\cap$ $[t_1, t_2]$)
4: **end if**
5: **if** (LPW $\cap$ $[t_1, t_2]$) && ! IPW) **then**
6: $\quad D_b+=t_i^{tr}-$(LPW $\cap$ $[t_1, t_2]$)
7: **end if**
8: $D_b+=\omega_1 \cap \omega_2$
Where: $\omega_1=[s_{i,k}, t_i^{tr}+s_{i,k}]$

$$\omega_2 = \bigcup_{\substack{\forall \lambda_j \in SSSR \setminus \lambda_i \\ \forall jobs\ k \\ \forall \ell : s_{i,k} < s_{j,\ell}+t_j^{tr} \wedge s_{i,k}+t_i^{tr} < s_{j,\ell}}} [s_{j,k}, t_j^{tr}+s_{j,k}]$$

high priority workload (HPW) executes during $[t_1, t_2]$, then the size of their overlap can be reclaimed because delayed execution of $j_{i,m}$ due to transition time of its device does not affect the high priority workload. Therefore, a device budget of 1 time unit can be reclaimed for an overlap of λ_2 with an execution of $j_{1,3}$ within an interval of [21; 22].

In the case of low priority workload (LPW) executes in $[t_1, t_2]$ then the system needs to consider the intermediate priority workload (IPW) that may execute during the leftover execution time of $j_{i,m}$. This IPW consists of jobs that will release in future and have deadlines between earliest deadline of the LPW that executed in $[t_1, t_2]$ and the deadline of the $j_{i,m}$. The IPW can be predicted by considering the previous release information.

The delayed execution of $j_{i,m}$ due to the device transition in this scenario can only affect the workload that we define as the IPW. LPW that was executed during $[t_1, t_2]$ will just switch their execution slots with $j_{i,m}$ execution by an amount they have executed in $[t_1, t_2]$. Jobs having priority higher and lower than $j_{i,m}$ will not be affected anyway. The reclaimed budget is added back to D_b.

Theorem 4: If there's a low priority task τ_l executing during a wake-up transition of λ_i and there is no intermediate priority task released prior to the completion of task τ_i (using λ_i) then the wake-up transition time overlapped with τ_l execution can be reclaimed and added back to D_b.

Proof: Since no intermediate priority task is executing, any change in the schedule can only affect the two tasks (τ_l and τ_i) in question. All the tasks having priority higher than τ_i will not be affected because they can preempt as soon as they are released. Similarly, tasks with priority lower than τ_l cannot preempt τ_i or τ_l, hence, will not affect the schedule. If the low priority task τ_l execute during sleep transition, it will swap its execution with the τ_i and either complete its execution earlier or at its normal time. The potential extension of the response time of task τ_i is already considered when obtaining the device budget D_b. As none of the tasks miss their deadlines therefore, theorem holds. ∎

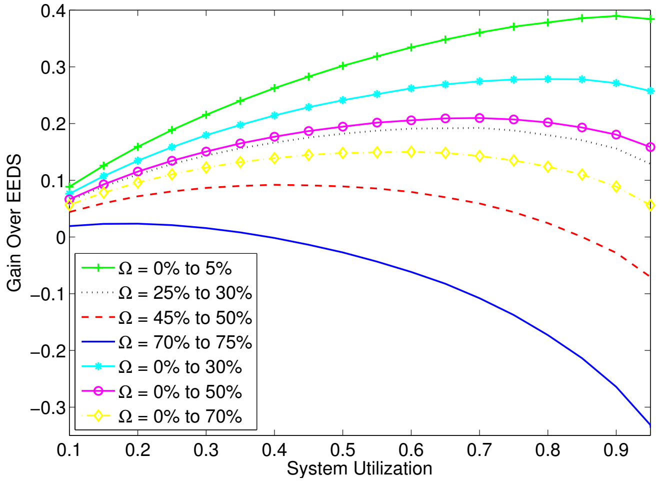


Fig. 2: Variation in Ω

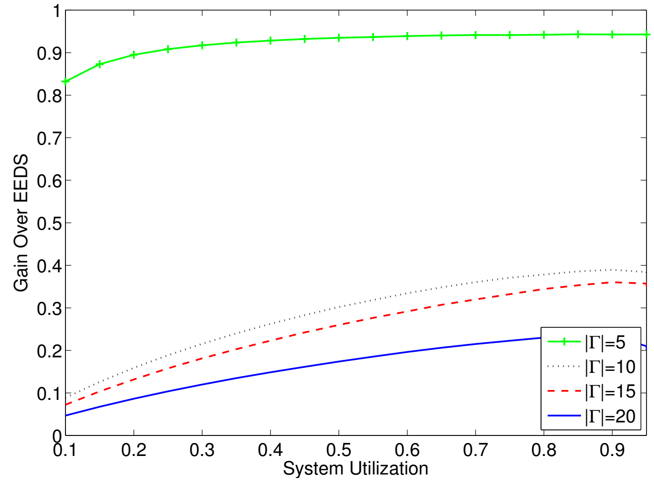


Fig. 3: Variation in Γ against U

TABLE I: Overview of Simulator Parameters

Parameters	Specifications
Task-set sizes $\|T\|$	$\{5, 10, 15, 20\}$
Inter-arrival time T_i for RT tasks	$[30ms, 50ms]$
Inter-arrival time T_i for BE tasks	$[50ms, 1sec]$
Sporadic delay limit $\Upsilon \in$	$\{0, 0.2, 0.4, 0.6, 0.8, 1\}$
Best-Case execution-time limit C^b	$\{0.25, 0.5, 0.75, 1\}$
Share of RT/BE tasks $\xi = \{\xi_1, \xi_2\}$	$\{\langle 40\%, 60\%\rangle, \langle 60\%, 40\%\rangle\}$

TABLE II: Device Power Model Parameters

Device Name	P_a	P_s	P_{tr}	t_i^{tr}
SST Flash $SST39LF020$	125	1	50	1
Simpletech Flash Card	225	20	100	2
Realtech Ethernet Chip	190	85	125	10
Texas Instrument $CC2430$	80.7	.0009	40	.525
MicroSSD(8GB)	412.5	2.31	≈ 0	≈ 0
TJA1043 Transceiver	325	.01	162.5	.05
Mica2Mote	29	.145	72.5	5
Lin Transceiver$NCV7321$	19.2	.12	9.6	.15
IBM MicroDrive	1300	100	500	12

VI. Complexity Comparison

Suppose p is the total number devices in the system. The complexity of the near optimal algorithm MDO [3] is $O(pH^2)$, where H is the hyper-period. SYS-EDF [9] has a complexity of $O(m \times 2^p)$, where m is the number of frequency set-points in the system. EEDS [6] complexity is $O(pl)$. Their algorithm performs the device transition decision on every job release, job completion and when the timer to reactivate the device expires. The state of all the devices is re-evaluated on each of the instants mentioned above. DFR-RMS [10] has the same complexity of EEDS, i.e. $O(pl)$.

SSC proposes the more efficient device energy saving algorithm with low complexity. The overall complexity of our algorithm is $O(l)$. In our algorithm, a device state decision is made when a job requests the device, a job completes its execution or when the timer to activate the device expires. Unlike the state-of-the-art, only a device related to this job will be serviced, the statues of the other devices is not re-evaluated. Only the routine that has to compute the high priority work load has the complexity of $O(l)$. This routine is only used when the timer associated to a device expires and D_b is insufficient. Otherwise, all the other routines have the constant complexity of $O(1)$. The device budget reclamation algorithm has the same complexity of $O(l)$.

VII. Evaluation

We have used a discrete event simulator SPARTS V-2.0 (Simulator for Power Aware and Real-Time System) [17] for the experiments to evaluate the effectiveness of our proposed algorithm. To cover the wide variety of applications we have used task-sets ranging from a larger number of fine grained small tasks (20) to a small number of coarse grained tasks (5). We have extended SPARTS to account for I/O devices, in terms of temporal behaviour and power consumption. Each task is allocated a device and its time of device usage is randomly chosen within its $\hat{c}$ limit. Total usage time of a device within a tasks execution is controlled with two variables that define the lower and upper bounds. These bounds are defined in a percentage of task's WCET.

Generally, we have used the SPARTS simulator with the parameters identified in Table I. This includes as indicated task-set sizes between 5 and 20 tasks. The two different share distributions ξ_1 and ξ_2 divide the task-set size and overall system utilisation between RT and BE tasks. Moreover, the utilisation allocated to each task type is randomly distributed among the tasks of the same class. The minimum inter-arrival times of RT tasks has been set to be in a range of $30ms$ to $50ms$, while those of BE tasks are up to $1sec$ and SPARTS computes the WCET C_i of τ_i to be $U_i * T_i$. In our experiments

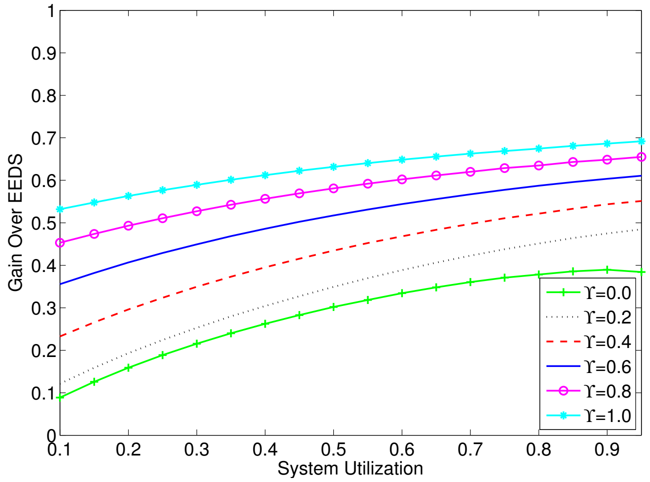


Fig. 4: Variation in Υ

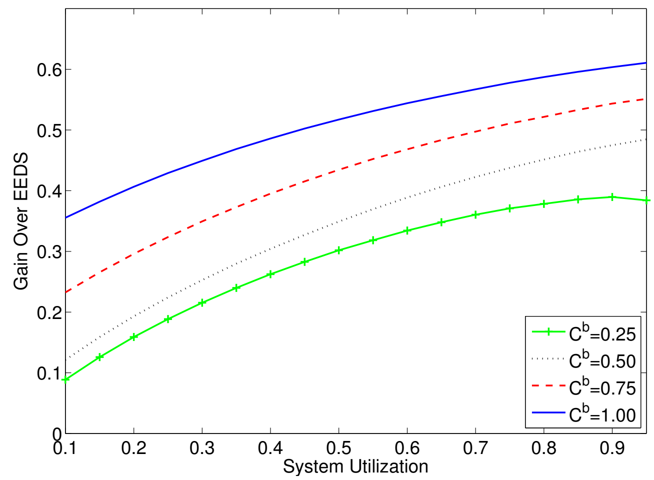


Fig. 5: Variation in C^b

we have found essentially little impact to allow borrowing of best-effort tasks and have hence concentrated on the effect of longer task periods for BE tasks.

The sporadic delay has been set to be in the range of $[0 : 1]$ in steps of 0.2 units of time. We have also experimented with different best-case execution time C_i^b's of a task. SPARTS follows a two level approach, where the actual parameters for an individual job j_i, m are taken from bounds allocated previously to the task τ_i. The interested reader is referred to [17], [18] for more details of SPARTS.

Suppose the percentage share of the device usage time in any job's actual execution time is represented as Ω. Then the device usage time by λ_i in any job $j_{i,m}$ is estimated as $\Omega * c_{i,m}$. The overall system utilisation is varied from 0.1 to 1 with an increment of 0.05. Each task-set is simulated for 100 seconds. For the comparison purposes we have also implemented the EEDS approach [6] in the SPARTS simulator. Furthermore, the power model for different devices used in our algorithm is based on their data sheets values shown in Table II. All the parameters given in Table II are in milliwatts/milliseconds. Furthermore, t_{sw} has been assumed when not given in the data sheet.

We have computed the gain in energy consumption of our algorithm against the EEDS for several scenarios. The effect of variation in Ω on the gain of SSC over EEDS is illustrated in Figure 2. We fixed $|\Gamma| = 10, \xi = 1, C^b = 1, \Upsilon = 0$ for this experiment. If the percentage of the device usage time is within a range of 0% to 70% of $c_{i,m}$, SSC outperforms EEDS. However, if all the jobs use their corresponding devices for a high percentage of their $c_{i,m}$, the performance of SSC declines eventually. For example, consider that the jobs use their corresponding devices for more than 70% of $c_{i,m}$, EEDS performance tends to rise after $U \geq 0.4$. Similarly, if the device usage time is in an interval of 45% to 50% of $c_{i,m}$ then EEDS saves more energy after $U > 0.85$. This occurs because intra-task device scheduling algorithm is designed with a consideration that devices are used for a very short interval, and hence woken up on demand. Systems with very high utilisation of device times are favourable for EEDS. Therefore, for all the following experiments we assume Ω that defines the device usage time for each job is selected randomly between 0% to 5% of the corresponding job's $c_{i,m}$.

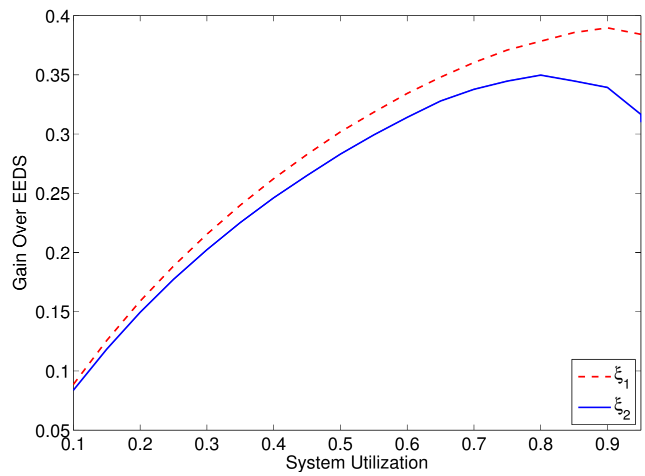


Fig. 6: Variation in ξ

We explored the energy gain of SSC over EEDS for different task-set sizes against the different system utilisations as shown in Figure 3. The parameters fixed for this experiment are $C^b = 1, \xi = 1$ and $\Upsilon = 0$ (i.e. tasks execute for their C_i and are strictly periodic). Figure 3 shows that the gain of our algorithm increases with an increase in the system utilisation. EEDS cannot extend sleep intervals of the devices at higher utilisation. Moreover, with an increase in the task-set size, D_b has to service extra devices and thus the gain decreases. The gain of $|T| = 5$ is very high when compared to other task-set sizes, as we used favourable devices with less overheads to illustrate that the effectiveness of our approach tends to rise with a decrease in device transition overhead.

In Figure 4, we have varied Υ with $|\Gamma| = 10$ and $\xi = 1$. To only demonstrate the effect of variation in the sporadic slack, C^b is set to 1. An increase in Υ injects more sporadic slack in the system, and hence, extra sporadic slack allows for larger gains in energy consumption. SSC makes an efficient use of the sporadic slack because device is only woken up on demand and kept in sleep mode if the task arrives later than its T_i. However, EEDS has the requirement to keep the device on during C_i; therefore, devices are woken up assuming a worst-case scenario of task arrival after every T_i. Additionally, it is hard to predict the sporadic slack in the system, thus no such mechanism can be integrated into EEDS to make use of the sporadic slack.

The third scenario shows the effect of variation in C^b (variation in execution slack) on the energy gain of SSC over EEDS in Figure 5. We fixed $\Upsilon = 0$ (no sporadic slack), $\xi = 1$ and $|\Gamma| = 10$. SSC performs well with an increase in system utilisation. However, the gain decreases with an increase in execution slack for an obvious reason that if tasks finish their execution earlier than C_i, EEDS has a chance to turn their corresponding devices off immediately afterwards.

Figure 6 shows that the gain in energy consumption of ξ_1 is higher than ξ_2. In ξ_1, the percentage of BE tasks in task-set size is greater than ξ_2. BE tasks usually run for long intervals. Therefore EEDS in ξ_2 keep the devices on for longer duration for more BE tasks when compared to ξ_1 and consequently consumes slightly more energy. In SSC at $U = 1$, device budget $D_b = 0$ and hence, it has to just rely on the next device usage time information of the device. However, the aggressive nature of EEDS algorithm to re-evaluate each device's status on every job release, job completion and time-out, pays off and saves more energy when compared to SSC.

VIII. Conclusions and Future Work

This paper presents the intra-task device scheduling algorithm SSC, which requests the device on demand rather than keeping it unnecessary active throughout the execution of its corresponding job. SSC makes explicit use of static and dynamic slack. Our extensive evaluation demonstrates its efficiency. Furthermore, sporadic slack is implicitly used in our algorithm. It has low complexity when compared to the state-of-the-art and reduces the assumptions that restrict the practical implication of these approaches. In the future, we intended to further relax the assumptions made in this research effort that will enhance the applicability of this algorithm to more versatile systems. Our goal is to allow device sharing among jobs and add flexibility to use multiple devices in a single job.

Acknoledgements

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within REPOMUC project, ref. FCOMP–01-0124-FEDER-015050, and by FCT and the EU ARTEMIS JU funding, within RECOMP project, ref. ARTEMIS/0202/2009, JU grant nr. 100202.

References

[1] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," IEEE Transactions on Very Large Scale Integration Systems, vol. 8, no. 3, pp. 299 –316, june 2000. 1

[2] V. Swaminathan, K. Chakrabarty, and S. Iyengar, "Dynamic i/o power management for hard real-time systems," in Proceedings of the 9th International Symposium on Hardware/Software Codesign, 2001, pp. 237 –242. 1

[3] V. Swaminathan and K. Chakrabarty, "Energy-conscious, deterministic i/o device scheduling in hard real-time systems," IEEE Transactions on CAD ICAS, vol. 22, no. 7, pp. 847 – 858, july 2003. 2, 7

[4] ——, "Pruning-based, energy-optimal, deterministic i/o device scheduling for hard real-time systems," ACM Transactions on Embedded Computing Systems, vol. 4, pp. 141–167, February 2005. 2

[5] Y.-H. Lu, L. Benini, and G. De Micheli, "Low-power task scheduling for multiple devices," in Proceedings of the 8th International Workshop on Hardware/Software Codesign, ser. CODES '00. New York, NY, USA: ACM, 2000, pp. 39–43. 2

[6] H. Cheng and S. Goddard, "Online energy-aware i/o device scheduling for hard real-time systems," in Proceedings of the 43rd ACM/IEEE Conference on Design Automation Conference. Leuven, Belgium: European Design and Automation Association, 2006, pp. 1055–1060. 2, 7, 8

[7] V. Devadas and H. Aydin, "Real-time dynamic power management through device forbidden regions," in Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium. Washington, DC, USA: IEEE Computer Society, 2008, pp. 34–44. 2

[8] J. P. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in Proceedings of the 10th IEEE Real-Time Systems Symposium, 1989, pp. 166–171. 2

[9] H. Cheng and S. Goddard, "Integrated device scheduling and processor voltage scaling for system-wide energy conservation," in Proceedings of the 2005 Workshop on Power Aware Real-time Computing, Sep. 2005. 2, 7

[10] V. Devadas and H. Aydin, "On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications," in Proceedings of the 8th International Conference on Embedded Software. Atlanta, GA, USA: ACM, 2008, pp. 99–108. 2, 7

[11] 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 Proceedings of the 24th IEEE Real-Time Systems Symposium, Cancun, Mexico, Dec. 2003. 2

[12] 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," Journal of Real–Time Systems, vol. 2, pp. 301–324, 1990. 3

[13] R. Pellizzoni and G. Lipari, "Feasibility analysis of real-time periodic tasks with offsets," Journal of Real–Time Systems, vol. 30, pp. 105–128, 2005. 3

[14] A. Rahni, E. Grolleau, and M. Richard, "Feasibility analysis of non-concrete real-time transactions with edf assignment priority," in Proceedings of the 16th Conference Real-Time and Networked Systems, Oct. 2008. 3

[15] M. A. Awan and S. M. Petters, "Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems," in Proceedings of the 23rd Euromicro Conference on Real-Time Systems, 2011, pp. 92–101. 3

[16] C. Lin and S. A. Brandt, "Improving soft real-time performance through better slack management," in Proceedings of the 26th IEEE Real-Time Systems Symposium, Miami, FL, USA, Dec. 2005. 3

[17] B. Nikolic, M. A. Awan, and S. M. Petters, "SPARTS: Simulator for power aware and real-time systems," in Proceedings of the 8th IEEE International Conference on Embedded Software and Systems. Changsha, China: IEEE, Nov. 2011. 7, 8

[18] ——, "SPARTS: Simulator for power aware and real-time systems," 2011, http://www.cister.isep.ipp.pt/projects/sparts/. 8