IEEE Real-Time and Embedded Technology and Applications Symposium

Real-Time Dynamic Power Management through Device Forbidden Regions [*]

Vinay Devadas   Hakan Aydin
Department of Computer Science
George Mason University
Fairfax, VA 22030
{vdevadas, aydin}@cs.gmu.edu

Abstract

Dynamic Power Management (DPM) techniques are crucial in minimizing the overall energy consumption in real-time embedded systems. The timing constraints of real-time applications and non-trivial time/energy transition overheads introduce significant challenges, as the device sleep intervals should be longer than a minimum threshold (called the *break-even time*) to ensure energy-efficiency. In this paper, we present a novel approach to the real-time DPM problem by explicitly enforcing long device sleep intervals for different devices, called *device forbidden regions*. We focus on the application of our technique to task systems with Rate-Monotonic priorities, and develop our algorithm DFR-RMS. Our solution includes a static component where the duration and frequency of forbidden regions are determined through the extended time-demand analysis to preserve the temporal correctness of all the tasks, while enhancing the energy savings. Then, we present a sophisticated on-line component which interacts with existing prediction-based DPM schemes to realize the full potential of device forbidden regions. Further, our scheme can be used with or without Dynamic Voltage Scaling (DVS). Our experimental evaluation hints that significant energy gains can be obtained, when compared to the existing prediction-based techniques. Another contribution of this research effort is to show that the general problem of generating feasible schedules for preemptive periodic real-time tasks where all device sleep intervals are longer than the device break-even times is NP-Hard in the strong sense.

1 Introduction

Due to the ever-increasing presence of computing and communication devices that rely on battery power, energy management remains as one of the most important research avenues. In real-time embedded systems, a widely popular energy management technique is *Dynamic Voltage Scaling (DVS)*. DVS is based on adjusting the CPU voltage and frequency on-the-fly [26]. DVS techniques exploit the convex relationship between the CPU power and processor frequency, and target saving energy by reducing the CPU frequency (speed) and supply voltage. As guaranteeing the timing constraints is of paramount importance in real-time embedded systems, numerous studies were published in the last decade to maximize energy savings by reducing the CPU speed while preserving the *feasibility* under different task/system models [3, 20, 21].

More recently, the research community grew increasingly conscious about the effects of DVS on *system-wide* energy consumption. In fact, CPU is only one of the significant power consumers in a computer system. A number of research groups reported [12, 22] that DVS *can increase* the total energy consumption; mainly because, with increased task execution times, it may force the components other than the CPU (e.g. memory, I/O devices) to remain in active state for longer time intervals. Further, the DVS schemes should recognize that the real-time application workloads have *frequency-independent (off-chip)* components which do *not* scale with the CPU frequency. A number of research studies [2, 13, 28] explored the problem of computing *task-level energy-efficient speeds* below which DVS ceases to be energy-efficient. The work in [2] presents one of the most advanced solutions to this problem, where task-level speed assignments are computed in polynomial-time for periodic real-time applications scheduled by preemptive Earliest-Deadline-First (EDF) policy, while taking into account the task off-chip/on-chip workloads, the CPU and device active powers.

The other major line of research involves *Dynamic Power Management (DPM)*. With DPM, some system devices are put to *low-power/sleep* states when they are not in use, in order to reduce their power consumption [19]. Traditionally, I/O devices (e.g. communication units and disk drives) and memory modules with considerable active power figures have been the primary targets of DPM techniques. The primary challenge in DPM techniques is to decide *when* to switch a device to a low-power state: this is because there is usually a significant energy and time overhead in switching between *active* and *sleep* states. For this reason, there is a device-dependent minimum idle period length that compensates for the overhead involved in the transitions, called the *break-even time*. If the device has to be re-activated before this threshold, then the overall energy consumption will *increase*. Stochastic, predictive and timeout-based techniques were proposed and developed by the research community to address the problem of assessing the right time instant to turn off a component [4].

In real-time systems, predicting the next time instant when a device will be needed is crucial not only because it is the key for effective energy management, but also because inaccurate predictions may hurt the *feasibility* of the system, in view of the non-trivial activation/de-activation delays. Hence, the real-time DPM has a number of unique characteristics. A number

[*]This work was supported by US National Science Foundation CAREER award (CNS-0546244).

of research groups recently tackled this problem. For example, [23, 24] present heuristic-based DPM schemes that can be used with both EDF and RMS scheduling polices. However, the schemes consider only non-preemptive real-time task execution. In [25], the same group of authors present an offline scheme, called *Maximum Device Overlap (MDO)*, for preemptive task scheduling. MDO involves very high time complexity (proportional to the square of the hyperperiod) and it cannot be successfully adapted to dynamic/online settings where job release and execution times can vary considerably.

For such systems, *online* dynamic power management becomes an imperative. The University of Nebraska-Lincoln Real-Time Systems research group exploited various aspects of this problem, mostly within the periodic EDF scheduling framework. In [6, 9], the authors give an online DPM scheme based on next device usage predictions. Further, the algorithm can be used in conjunction with DVS. In [8], the authors present *Energy Efficient Device Scheduling (EEDS)* framework, which is based on exploiting task and device slacks to create long idle intervals, again for preemptive EDF. In [7], EEDS is extended to include non-preemptive shared resources.

In this paper, we present a novel approach to the online real-time DPM problem. Specifically, by observing that creating long device sleep intervals is the key for effective power management, *we explicitly and periodically enforce such intervals for each device at run time*. These intervals, called *device forbidden regions (DFRs)*, enable the system to put these devices to sleep states. The parameters (duration and separation time) of DFRs are determined through static analysis. Further, the forbidden regions are guaranteed to be *longer than the device break-even times*, ensuring energy savings.

While an attractive approach, DFR scheme introduces a number of challenging technical problems. During a forbidden region, none of the tasks using the related device can be dispatched; hence, the duration and period of DFRs must be carefully selected to preserve temporal correctness. The DFR approach is *generic* in the sense that it can be used with any scheduling policy and with/without a specific task speed assignment (i.e. with or without DVS). Moreover, as we show, the activation of a forbidden region can be dynamically postponed under certain conditions, to eliminate unnecessary transition overheads.

In this paper, we illustrate the application of DFRs to fixed-priority real-time systems with Rate-Monotonic scheduling (RMS), and present the details of the resulting algorithm, called DFR-RMS. It is a fact that most of the online real-time DPM schemes to date were developed for EDF [7, 8, 9]. While an evaluation of relative merits of dynamic and fixed priority scheduling in real-time systems is still open to debate, we contend that the well-established design methodologies and large number of real time applications depending on RMS [17] justify such a decision. We illustrate how the well-known Time Demand Analysis (TDA) [1, 15] technique can be extended to assess the feasibility of schedules obtained by DFR-RMS. Further, we present an efficient greedy algorithm to determine the duration and activation periods of individual DFRs to maximize energy savings. Finally, we show how the algorithm can be adapted to DVS settings where dynamic reclaiming [3, 20] helps to save additional energy.

Our experimental evaluation shows that, DFR-RMS can yield significant (up to 27%) gains in device variable active energy, which is typically consumed by short idle intervals during which the device(s) cannot be shutdown by typical DPM algorithms. Further, this translates to non-trivial gains in *overall system energy*. Finally, a by-product of this research effort is to show that the general problem of generating feasible schedules for preemptive periodic real-time tasks where all device sleep intervals are longer than the device break-even times is NP-Hard *in the strong sense*, underlining the inherent difficulty of DPM for real-time applications, in general.

The rest of this paper is organized as follows. In Section 2, we present our notation and system models. Section 3 elaborates on the basic principles of existing online real-time DPM schemes, and illustrates the main rationale of DFR scheme, through a running an example. In Section 4, we explain how the Time Demand Analysis can be used to check the feasibility of the systems with DFR, assuming Rate-Monotonic task priorities. In Section 5, we present a method to determine the duration and minimum separation times of forbidden regions, to maximize energy savings. Section 6 includes a full discussion of the online component of the algorithm. Section 7 presents the experimental evaluation, followed by conclusions in Section 8.

2 System Model and Assumptions

2.1 Task Model

We consider a set of independent periodic real-time tasks $\Psi = \{\tau_1, \dots, \tau_n\}$ that are to be executed on a uniprocessor system. The period of task τ_i is denoted by T_i, and the relative deadline of each task instance (job) is equal to the task period. The j^{th} instance of task τ_i is denoted by $\tau_{i,j}$. Task priorities are inversely proportional to the periods (i.e. we assume rate-monotonic priorities [16]). We assume preemptive scheduling.

The worst-case execution of time of task τ_i is denoted by C_i. The utilization of task τ_i is defined as $U_i = \frac{C_i}{T_i}$. Note that, on DVS-enabled processors where the processor speed (frequency) S can vary between a lower bound S_{min} and an upper bound[1] S_{max}, C_i and U_i will be a function of the CPU speed. Specifically, the worst-case execution time $C_i(S)$ of task τ_i at CPU speed S is given by $C_i(S) = \frac{x_i}{S} + y_i$ where x_i is the task's on-chip workload at S_{max}, and y_i is the task's off-chip workload (that does not scale with the CPU speed) [2]. Similarly, the task's *effective* utilization $U_i(S)$ at CPU speed S can be defined as $\frac{C_i}{S \cdot T_i}$.

[1]For convenience, we normalize the CPU speed with respect to S_{max}; that is, we assume that $S_{max} = 1.0$

Throughout the paper, we will use the notation C_i (U_i) to refer to the worst-case execution time (utilization) under maximum CPU speed. The *base total utilization* U_{tot} of the task set is the aggregate utilization of all the tasks at the maximum CPU speed, that is $U_{tot} = \sum_{i=1}^{n} U_i$. We will assume that the task set is deemed to be feasible when all tasks are executed with S_{max} and by preemptive Rate Monotonic Scheduling policy: any of the several well-known RMS schedulability tests [10, 15, 16] can be used for this purpose.

2.2 Device Model

We assume that the system has a device set $\mathcal{D} = \{D_1, \ldots, D_m\}$ with m off-chip (external) devices. The devices that we consider will have at least an *active* and a *sleep* state. In the *sleep* state, a device typically consumes much less power compared to the *active* state [8].

The devices used by individual tasks are given by *Device Usage Lists (DULs)*[9]. Specifically, DUL_i corresponds to the set of devices needed by τ_i during its execution. Similarly, γ_i denotes the set of tasks that need the device D_i during their execution: formally, $\gamma_i = \{\tau_x | D_i \in DUL_x\}$.

Following [8, 9, 23, 24], we assume *inter-task device scheduling*: all devices in DUL_i need to be in *active* state when τ_i executes. This is a conservative but realistic assumption, considering the non-trivial time and energy overheads involved in performing device state transitions during task execution [9, 23]. We adopt the following notation to show the power/energy characteristics of a given device D_i :

- P_a^i: The power consumption in *active* state
- P_s^i: The power consumption in *sleep* state
- t_{as}^i: The time overhead to perform a transition from *active* to *sleep* state
- t_{sa}^i: The time overhead to perform a transition from *sleep* to *active* state
- E_{as}^i: The energy overhead to perform a transition from *active* to *sleep* state
- E_{sa}^i: The energy overhead to perform a transition from *sleep* to *active* state
- B_i: The *break-even time* of the device

The energy consumption of device D_i can be divided in three components:

$$E_{device}^i = E_{fixed}^i + E_{trans}^i + E_{mod}^i \tag{1}$$

Above, E_{fixed}^i is the energy consumed by the device when it is actively in use by the *running task* τ_x (i.e. when $D_i \in DUL_x$ and τ_x is running). Under the inter-task device scheduling paradigm (see above), for a given workload, the online DPM algorithms do not have direct control on E_{fixed}^i, as the device will be in *active* state whenever a task in γ_i is running.

However, the online DPM algorithms can transition a device to *sleep* state when a task that does not use that device is dispatched (or, when the CPU is idle). If undertaken, such a decision will involve an energy overhead (E_{trans}^i) and a time delay, for transitions in each direction. During intervals where D_i is kept in *active* state, even though $D_i \notin DUL_x$ (where τ_x is the running task), there will be additional energy consumption. This component, denoted by E_{mod}^i, is typically due to an anticipation of a request for the device D_i in the near future. In fact, most of the existing online real-time DPM algorithms try to predict the length L of the idle interval, and initiate the transition to the *sleep* state only when L is larger than a pre-defined threshold value. This threshold value, generally called *the break-even time (B_i)* of device D_i can be computed as [8, 9]:

$$B_i = max\{t_{as}^i + t_{sa}^i, \frac{E_{as}^i + E_{sa}^i - [P_s^i \cdot (t_{as}^i + t_{sa}^i)]}{P_a^i - P_s^i}\} \tag{2}$$

By avoiding switching a device to the *sleep state* when the predicted idle interval is shorter than the break-even time as defined above, the DPM algorithms make sure that device transition is energy-efficient. Further, Equation (2) guarantees that the length of the interval is greater than the time needed to perform the transitions (i.e. $t_{as}^i + t_{sa}^i$).

As a result, the decisions of DPM algorithms affect E_{trans}^i $\underline{\text{and}}$ E_{mod}^i. Defining $E_{var}^i = E_{trans}^i + E_{mod}^i$, we can re-write:

$$E_{device}^i = E_{fixed}^i + E_{var}^i \tag{3}$$

While the system-wide energy consumption remains as a very important metric, observe that the relative merits of an online DPM algorithm can be accurately measured by the savings it yields for $E_{var} = \sum E_{var}^i$.

2.3 System Energy Model

The total energy consumption of an embedded real-time system can be defined by the sum of energy consumption of on-chip (primarily, CPU) and off-chip units (external devices, such as main memory and I/O devices):

$$E_{tot} = E_{CPU} + E_{device} = E_{CPU} + \sum_{i=1}^{m} E_{device}^i \tag{4}$$

On DVS settings, the CPU energy consumption is the sum of the dynamic CPU power consumption (which can be adjusted by modifying the CPU supply voltage and frequency) and static power [2, 14]. If the system does not have the DVS capability, then the system always uses the speed $S_{max} = 1.0$ when executing tasks, and switches to the idle mode when there are no ready tasks. It is assumed that *completely* turning off the CPU or devices at run-time is not feasible, in view of the periodic nature of the real-time applications.

3 Online Dynamic Power Management for Real-Time Systems

One of the major challenges in online real-time dynamic power management is to make sure that the device sleep intervals are longer than the corresponding break-even times, to guarantee the energy efficiency. In theory, the scheduler can attempt to order the execution of tasks in such a way that the device usage and sleep intervals are grouped together to the extent it is possible. Unfortunately, the following result underlines the inherent difficulty of the problem:

Theorem 1 *Given a set of periodic tasks Ψ and a set of devices $\mathcal{D}$, generating a feasible schedule where* each *device sleep interval is greater than the corresponding device break-even time is NP-Hard in the strong sense.*

The full proof of this result, which is omitted due to space constraints, can be found in [11]. In [19], it was already proven that the dynamic power management problem is NP-Complete (in the weak sense) even for tasks without timing constraints. Our result implies that the problem of guaranteeing the feasibility and long sleep intervals at the same time (when one exists) is not likely to admit even a pseudo-polynomial time solution, unless $NP = P$.

3.1 Predicting Device Usage Times

The alternative solution is to commit to a given scheduling policy (such as RMS or EDF) and then perform device transitions judiciously by trying to predict the *next device usage time* of device D_i at time t (denoted by $NDUT_i(t)$), at run-time. The device can be put to *sleep state* with confidence if the difference $NDUT(t) - t > B_i$, as long as $NDUT(t)$ does not overestimate the actual next device usage time.

Unfortunately, the stringent timing constraints of real-time tasks, the variability in actual execution times, and release time jitters make a precise prediction impossible. Further, even if all this information is known to a *clairvoyant* scheduler, performing an exact online response time analysis for various task instances (to figure out when a task in need of D_i will be dispatched next) is not computationally feasible [17].

One conservative but safe solution is to compute $NDUT_i(t)$ by considering the current time, the tasks in ready queue, and the release time of any *uncompleted* job that requires D_i. Hence, if the ready queue contains a task that requires D_i, $NDUT_i(t)$ will be simply t; otherwise it will be the earliest next release time of any instance of a task in γ_i. This is indeed the main principle of the *Conservative Energy-Efficient Device Scheduling (CEEDS)* algorithm, proposed by Cheng and Goddard [6, 9]. When de-activating a device, CEEDS also schedules an *activation time* (or, *UpTime*) by considering typically non-negligible re-activation delays and NDUT. By introducing a running example, we illustrate the main principles of CEEDS.

Consider three real-time tasks τ_1, τ_2 and τ_3 with the following parameters: $C_1 = C_2 = C_3 = 1000, T_1 = 2000, T_2 = 4000$ and $T_3 = 8000$. The devices D_1 and D_2 are used by the tasks τ_1 and τ_2, respectively. For these two devices, $B_1 = 990, t^1_{as} = t^1_{sa} = 495, B_2 = 20$ and $t^2_{as} = t^2_{sa} = 10$. Assume all tasks are released at $t = 0$ and all devices are initially in *active* state.

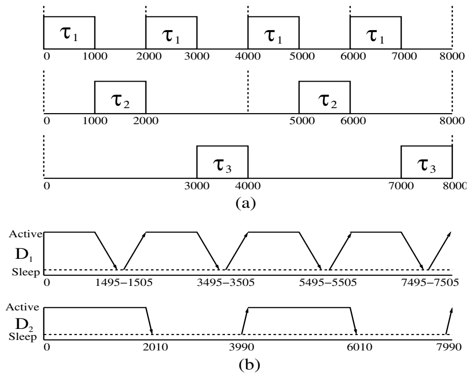


Figure 1: The schedule generated by RMS and CEEDS

Figure 1a shows the task and device schedules generated by RMS and CEEDS. At $t = 1000$, CEEDS switches D_1 to sleep state as the time to predicted next device usage time (NDUT) for D_1 is greater than the device's break-even time. CEEDS also starts switching D_1 to active state at time 1505, to make sure that it is fully *active* by $t = 2000$ (the next release time of τ_1). Similar explanations can be given for other time instants involving D_1 and D_2 activations/deactivations.

Two observations are in order. First, observe that the transition times of D_1 are significant compared to its idle interval length. Thus, the total sleep time of D_1 is very small. Second, even though D_2 is not needed in intervals $[0, 1000]$ and $[4000, 5000]$, it remains active during these intervals. This is because, during these intervals, τ_2 is in the ready queue, D_2 is active and $D_2 \in DUL_2$. Figure 1b shows the device states and transitions over the hyperperiod. From Figure 1b, it can be seen that D_1 is in sleep state for a total of 40 time units while D_2 sleeps for 3960 time units.

3.2 Dynamic Power Management through Device Forbidden Regions

Our approach to real-time DPM problem is based on *the sleep intervals which are explicitly and periodically enforced for each device.* These special intervals are called *device forbidden regions*. A separate forbidden region FR_i is defined for each device D_i. The forbidden region FR_i has a predetermined duration $\Delta_i \geq B_i$; as a result D_i *can be safely put to the sleep state during its forbidden region*.

Further, a *minimum separation time* (or, *period*) Π_i is associated with FR_i, meaning that two consecutive "activations" of a given FR_i should be separated by at least Π_i time units. As it will be seen, under certain circumstances, it may be more beneficial to delay the next activation of a forbidden region. In other words, FR_i's may be activated in "sporadic" manner. The next "earliest release" (activation) time of FR_i during execution is denoted by the variable rf_i.

An extremely important implication of inserting enforced device forbidden regions to real-time schedules is that, *none of the tasks using D_i can be dispatched while FR_i is active*. However, other tasks (i.e. those in $\Psi - \gamma_i$) can still execute. In other words, the tasks in γ_i are effectively "blocked" by FR_i when the latter is active.

Naturally, a task using two different devices D_i and D_j would be blocked *whenever* FR_i or FR_j is active. It goes without saying that determining Δ_i and Π_i values to guarantee the feasibility of the real-time task set, and at the same time, to maximize energy savings is a non-trivial problem. These issues are addressed in Sections 4 and 5.

It is also important to underline that the DFR scheme does not exclude the usage of CEEDS, or for that matter, any other prediction-based online DPM algorithm. In fact, the feasibility analysis that we present (Section 4) treats each forbidden region FR_i as a high priority periodic task delaying the execution of tasks in γ_i. Consequently, when the system is able to shutdown the device D_i using the standard prediction techniques (e.g. through CEEDS), the next activation time rf_i of FR_i can be postponed without affecting feasibility. This, in turn, will help to save the bandwidth of the forbidden region and prolong the sleep interval in the future. Similarly, when a scheduled activation time of FR_i corresponds to a time instant when D_i is actively used by the running task, FR_i should be postponed to a later time, to avoid unnecessary transitions. Clearly, in all these cases, the activation of all subsequent forbidden regions of the same device will be delayed by the same amount.

To illustrate these principles, we re-visit the example given in Section 3.1 and show the schedule generated by DFR-RMS (Figure 2a). In this example, the forbidden region durations and separation times are selected as $\Delta_1 = \Delta_2 = 1000$ and $\Pi_1 = \Pi_2 = 4000$. It is assumed that the first scheduled release time of both forbidden regions is $t = 0$.

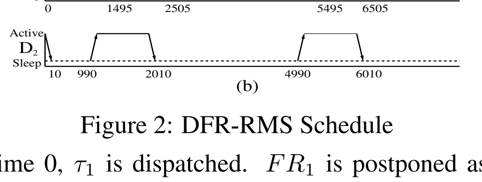


Figure 2: DFR-RMS Schedule

At time 0, τ_1 is dispatched. FR_1 is postponed as D_1 is already active and in use by the current job. FR_2 is enabled: D_2 is put to sleep state since it is currently not in use. We set the next activation time of D_2 to $t + \Delta_2 - t_{sa}^2 = 990$. At time 1000, τ_1 completes and τ_2 is dispatched. τ_2 finds D_2 in active state. Also, FR_2 ends and rf_2 is set to $t + \Pi_2 - \Delta_2 = 4000$. D_1 is put to sleep since $NDUT_1$ is found to be greater than B_1 through CEEDS prediction mechanism. In this case, we can further postpone FR_1 to preserve its "bandwidth" for the future. We set the activation time of D_1 to $NDUT_1 - t_{sa}^1 = 1505$.

At $t = 1505$, the predicted $NDUT_1$ is 2000. DFR-RMS, at this point, realizes that FR_1 is pending and makes a decision to forcefully start FR_1 at $t = 2000$. Observe that this allows D_1 to remain in sleep state without compromising the timing constraints. At $t = 2000$, FR_1 is enabled and the activation time of D_1 is set to 2505. In the mean time, D_2 is shutdown as $NDUT_2 > B_2$. The activation time of D_2 is set to 3990 and T_3 is dispatched. At time 3000, FR_1 is disabled and rf_1 is set to 6000. D_1 completes its transition to the active state and τ_1 preempts τ_3 upon arrival. At time 3990, DFR again postpones activation of D_2 as rf_2 is 4000 which is the same as $NDUT_2$. It is worthwhile to observe how DFR prolongs the idle intervals and prevents D_2 from remaining in active state unnecessarily. The rest of the schedule can be obtained in the same way. Figure 2b shows the device states and transitions for the DFR algorithm. As a result, with DFR-RMS, D_1 remains in sleep state for a total of 2020 units while D_2 sleeps for 5950 units – a significant improvement over the figures obtained by the CEEDS algorithm.

We underline that, even though the on-going discussion assumed the execution at the maximum CPU speed, DFR-RMS scheme can work in conjunction with any DVS-based speed assignment framework, as long as the speed assignments preserve the system feasibility. For instance, the technique in [2] which determines the task speeds by taking into account task on-chip and off-chip workloads, as well as frequency-dependent and -independent power components, can be readily adopted. One crucial difference is that unlike the EDF-based settings of [2] where the effective utilization can be set to 100%, here, total utilization may be limited by the well-known Liu-Layland bound $n(2^{\frac{1}{n}} - 1)$ [16]. In Section 6, we comment also on the potential of using dynamic reclaiming at run-time, in conjunction with DVS, to improve the energy savings.

4 Extending Time Demand Analysis to DFR-RMS

Time Demand Analysis (TDA) technique [15] is a well-established methodology to assess the feasibility of a periodic real-time task set scheduled by the preemptive RMS policy. It provides a sufficient and necessary condition for the schedulability and is generalized to various settings with aperiodic servers, precedence constraints, and task blocking times [1, 17]. TDA relies heavily on the *critical instant* concept, where the response time of a job is maximum when it is released simultaneously with all high-priority tasks [16]. Let $hp(i)$ denote all tasks with priority higher than that of a given periodic task τ_i. The time demand function of τ_i (denoted by $w_i(t)$) is defined as:

$$w_i(t) = C_i + \sum_{\tau_j \in hp(i)} \lceil \frac{t}{T_j} \rceil \cdot C_j$$

Theorem 2 [15, 17] *A set of fixed-priority periodic independent real time tasks with relative deadlines equal to the periods is feasible if and only if* $\forall\, i \;\; \exists\, t \;\; 0 \leq t \leq T_i$, $w_i(t) \leq t$, *under critical instant phasing.*

The exact characterization of the critical instant for DFR-RMS is non-trivial. Multiple forbidden regions that overlap in time will cause a total interference which is less than their total duration, and a given forbidden region FR_j may be dynamically postponed (for example, when D_j is in active use at the release time of FR_j). Further, the tasks in $hp(i)$ that typically delay the execution of τ_i may be themselves delayed by various forbidden regions. Nevertheless, it is still possible to use the traditional TDA technique by conservatively (i.e. by overestimating) the interference of high priority tasks in $hp(i)$ and the forbidden regions, on a given task.

Let $w_i^{FR}(t)$ be the time-demand function of τ_i in the DFR-RMS algorithm, such that the total interference on τ_i from tasks in $hp(i)$ and forbidden regions is maximized.

Proposition 1 *In DFR-RMS, for every task* τ_i *and every t in the range* $[0, T_i]$, *we have:*

$$\begin{aligned} w_i^{FR}(t) &\leq w_i^{max}(t) \\ &= C_i + \sum_{\tau_j \in hp(i)} \lceil \frac{t}{T_j} \rceil \cdot C_j + \sum_{j \in DUL_i} \lceil \frac{t}{\Pi_j} \rceil \cdot \Delta_j \end{aligned}$$

Proof: The right hand side of the expression in Proposition 1 overestimates the *actual* maximum interference on τ_i by assuming that:

- the task τ_i is released at the same time as all tasks in $hp(i)$,
- none of the tasks in $hp(i)$ is delayed by any forbidden regions until τ_i completes, and,
- all forbidden regions FR_j such that $D_j \in DUL_i$ are activated at $t = 0$, and each of these forbidden regions is treated as a high priority preemptive task invoked at the maximum frequency (i.e. every Π_j time units).

Note that the last assumption above effectively ignores the 'overlap effect' of forbidden regions on τ_i: if two forbidden regions FR_j and FR_k that delay τ_i overlap for x time units, then their total interference would be $\Delta_j + \Delta_k - x$, and not $\Delta_k + \Delta_j$ as assumed. However, it is fairly difficult to precisely characterize the aggregate impact of such overlaps – though it is certain that the worst-case interference for τ_i occurs when there are no such overlaps at all. Hence, the proposition holds.

Corollary 1 *A set of periodic tasks can be feasibly scheduled by DFR-RMS if* $\forall\, i \;\; \exists\, t \;\; 0 \leq t \leq T_i$, $w_i^{max}(t) \leq t$.

Note that just like the original TDA algorithm, it is necessary and sufficient to evaluate $w_i^{max}(t)$ at every period boundary in interval [0, T_i]. Hence, the overall complexity of the extended TDA is $O(\frac{max\{P_{max}, \Pi_{max}\}}{P_{min}}(n+m)^2)$.
Remark: If the task set has some shared resources which can lead to additional blocking, then these extra blocking times can be easily incorporated to the framework as illustrated for the traditional TDA in [1, 17].

5 Determining Forbidden Region Parameters

Before DFR scheme can be used with a given task set, the duration (Δ_i) and period (Π_i) of each forbidden region FR_i needs to be determined. While a fundamental requirement is to ensure the feasibility of the real-time task set, another major objective is to optimize energy savings with the selected Δ_i and Π_i values.

Intuitively, the longer Δ_i (beyond B_i), the higher energy savings for device D_i. Similarly, as Π_i decreases, the number of forbidden region instances of FR_i gets higher, increasing the overall sleep time. Also, considering that the energy savings of D_i during a sleep interval is proportional to the difference ($P_a^i - P_s^i$), we can define the *Expected Energy Savings (EES)* of D_i as:

$$EES_i = \frac{\Delta_i - B_i}{\Pi_i}(P_a^i - P_s^i) \quad (5)$$

In order to perform an efficient search in the possible $\{\Delta_i, \Pi_i\}$ spaces, we need to first establish some lower and upper bounds on these quantities.

Bounding Δ_i: Observe that having a forbidden region duration $\Delta_i \leq B_i$ is not helpful: this stems from the very definition of break-even time. Similarly, it is easy to see that Δ_i cannot exceed $T_j - C_j$ (the maximum allowable laxity for τ_j) for any task τ_j that uses D_i. Doing otherwise may result in a deadline miss for τ_j, in case that an instance of τ_j is released immediately after the activation of FR_i. Hence, we have:

$$B_i < \Delta_i \leq \min_{\tau_j \in \gamma_i}(T_j - C_j)$$

Bounding Π_i: Consider the quantity $z_i = \max_{j \in \gamma_i}\{T_j\}$. When evaluating the feasibility of any task in γ_i through the TDA in Section 4, the quantity $\lceil \frac{t}{\Pi_i} \rceil = 1$ for any $\Pi_i \geq z_i$. Hence, if a given task τ_x is infeasible with a certain $\Pi_i = z_i$ value ($D_i \in DUL_x$), then it is guaranteed to remain infeasible for any $\Pi_i > z_i$. Recalling that increasing Π_i does not help to improve the energy savings either, we can obtain $\Pi_i \leq \max_{j \in \gamma_i}\{T_j\}$.

Define the utilization U_{D_i} of a given device D_i as the total utilization of tasks in γ_i. Observe that the ratio $\frac{\Delta_i}{\Pi_i}$ for given forbidden region FR_i cannot exceed $1 - U_{D_i}$, since this is the maximum amount of time during which D_i can be in sleep state. Combining all this, we get:

$$\frac{\Delta_i}{1 - U_{D_i}} \leq \Pi_i \leq \max_{\tau_j \in \gamma_i}\{T_j\}$$

Given the expected energy savings formula (Equation (5)), a reasonable approach is to treat the devices with large EES_i values with high priority in the search process, to increase the potential energy savings. Moreover, for a given D_i, the Δ and Π ranges can be scanned at equi-distant points to assess the feasibility and EES_i with the given values. Our experience shows that evaluating EES_i for 10-15 equi-distant candidate Δ_i and Π_i values gives significant energy savings while keeping the running time at acceptable levels. The resulting *Greedy FR Assignment* algorithm is given below.

Complexity: At each iteration, the complexity of finding the best (Δ_i, Π_i) pairs is still $O(\frac{max\{P_{max},\Pi_{max}\}}{P_{min}}(n + m)^2)$, given the constant number of candidate points evaluated in the range. Hence, the overall complexity of this static algorithm is $O(\frac{max\{P_{max},\Pi_{max}\}}{P_{min}}m(n + m)^2)$.

Greedy FR Assignment Algorithm

```
• $Z = \{D_1, \ldots, D_m\}$
• $W = \emptyset$
• Repeat
    – for each device $D_i$ in $Z$, compute the ($\Delta_i$, $\Pi_i$) pair that gives the best $EES_i$ while committing to $\Delta$, $\Pi$ values already in $W$ and maintaining the feasibility.
    – Commit to ($\Delta_j$, $\Pi_j$) pair for $D_j$ in $Z$ that gives the maximum EES.
    – Set $W = W \cup D_j$
    – Set $Z = Z - D_j$
• Until ($\nexists i | D_i \in Z \wedge \{\Delta_i, \Pi_i\}$ is feasible)
```

6 DFR Algorithm: Online Routines

In this section, we present full details of the online component of the algorithm. For every forbidden region FR, Π and Δ values are computed using the technique given in the preceding section. Initial release times rf_i of all forbidden regions are set to 0 and all devices are assumed to be initially in *active* state. The algorithm performs device and forbidden region related actions at dynamically scheduled *device management points (DMPs).*

There are four DMP types: ENABLE, DISABLE, ACTIVATE and DEACTIVATE, corresponding to starting an FR, ending an FR, activating a device and deactivating a device, respectively. DMPs can be implemented using an internal table and a timer. The internal table can be used to record points at which DFR actions are scheduled. We assume the existence of a routine *ScheduleDMP()* for this purpose. *ScheduleDMP()* takes as input the DMP type and the action's scheduled timing information. Further, the system is assumed to have a programmable timer which issues interrupts at requested times, corresponding to DMPs. Upon the receipt of such an interrupt, the algorithm will be invoked by the operating system and will scan its internal table, performing all actions that were scheduled for this DMP. We now explain how Device Management Routines in Figure 3 can be used to perform DFR actions listed in Figure 4.

The system's ready queue is assumed to be divided into two components: R_a, which contains tasks currently eligible for execution, and R_b, which contains ready tasks currently "blocked" by active forbidden regions. At every job arrival point, the new job of task τ_i, is inserted to either R_a or R_b, depending on current active forbidden regions.

Note that each FR can be in three possible states. If an FR has started, it is in ENABLED state and upon termination the FR is switched to DISABLED state. FR is in PENDING state if it has been postponed. Starting an FR can be performed in two modes. In FORCE mode, FR is activated without any secondary check. In DEFAULT mode, some look-ahead is used to see if FR can be postponed to save additional energy, as explained below.

The *scheduler* function, shown in Figure 5, tries to put all active devices not in use by the current job to sleep, by invoking *TryShutDown()* function. There are two alternative ways of shutting down a device: either by predicting its next usage time and comparing it against the break-even time, or through forbidden regions. When predicting the next device usage time, our scheme takes also into account the blocking times of other tasks by other forbidden regions, in addition to standard CEEDS prediction mechanism. The second option involving FRs occurs when the FR associated with the device is in PENDING state. This occurs due to dynamic postponements of FR next release times.

Given these two options, the algorithm tries to shutdown the device using CEEDS so as to postpone FR and preserve its bandwidth for future, in the first place. However, if CEEDS is unable to shutdown the device and FR associated with the device is in PENDING state (has been postponed), we shutdown the device using FR scheme. Also notice here that if the predicted next device usage time $NDUT_i(t)$ of D_i is such that $t + B_i \geq NDUT_i(t) \geq rf_i$, then D_i can be safely and efficiently put to sleep. Given this, FR_i will be postponed at rf_i. At device activation point we can force this postponed FR_i to start at predicted next device usage time and hence postpone device activation. This helps in increasing device idle interval

```
1 GetNextDeviceUsage($D_i$,$t$):
   if $\tau_j$ in ready queue
    z = max($FR_h$ endtime|$FR_h$ blocks $\tau_j$)
    $NDUT(D_i,\tau_j)$ = max($t$,z)
   else
    $NDUT(D_i,\tau_j)$ = release time of $\tau_j \geq t$
   end if
   $NDUT_i(t)$ = min($NDUT(D_i,\tau_j)$) $\tau_j \in \gamma_i$
2 WakeUp($D_i$):
   if($NDUT_i$(t)$-t > B_i$)
    ScheduleDMP(ACTIVATE,$D_i$,$NDUT_i(t) - t^i_{sa}$)
   elseif($FR_i$ is PENDING)
    ScheduleDMP(ENABLE,FORCE,$FR_i$,$NDUT_i$(t))
   elseif($NDUT_i(t) \geq rf_i$)
    ScheduleDMP(ENABLE,FORCE,$FR_i$,$NDUT_i$(t))
   else
    Activate $D_i$
3 ShutDown($D_i$,TurnOnTime):
   TurnOnTime = max($t$+1,TurnOnTime)
   Deactivate $D_i$
   ScheduleDMP(ACTIVATE,$D_i$,TurnOnTime)
4 TryShutDown($D_i$):
   if($NDUT_i(t) - t > B_i$ OR $NDUT_i(t) \geq rf_i$)
     ShutDown($D_i$,$NDUT_i(t) - t^i_{sa}$)
   elseif ($FR_i$ is PENDING)
     StartFR($FR_i$)
5 StartFR($FR_i$):
   ShutDown($D_i$,$t + \Delta_i - t^i_{sa}$)
   ScheduleDMP(DISABLE,$FR_i$,$t + \Delta_i$)
   Move $\tau_j$ in ($\gamma_i \cap R_a$) from $R_a$ to $R_b$
   Set $FR_i$ to ENABLED
   Scheduler()
6 TerminateFR($FR_i$):
   Set $rf_i$ to $t + \Pi_i - \Delta_i$
   ScheduleDMP(ENABLE,DEFAULT,$FR_i$,$t$+$\Pi_i$−$\Delta_i$)
   Move $\tau_j$ in ($\gamma_i \cap R_b$) from $R_b$ to $R_a$
   Set $FR_i$ to DISABLED
   Scheduler()
7 DeviceCntrlRoutine($FR_i$, mode):
   if(mode equals FORCE)
    StartFR($FR_i$)
   else /* DEFAULT mode*/
    switch($D_i$ state)
    Case1:$D_i$ SLEEP OR TRANSITIONING
          Set $FR_i$ PENDING
    Case2:$D_i$ ACTIVE
          if($D_i \in DUL_h$)
           Set $FR_i$ PENDING
          else
           if($NDUT_i(t) - t > B_i$)
            ShutDown($D_i$,$NDUT_i(t) - t^i_{sa}$)
            Set $FR_i$ PENDING
           else
            StartFR($FR_i$)
           end if
          end if
   end if
```

Figure 3: Device Management Routines

```
1 At task arrival time $t$:
   if $\exists FR_i$:$FR_i$ ENABLED AND $D_i \in DUL_i$
    insert $\tau_i$ to $R_b$
   else
    insert $\tau_i$ to $R_a$
2 At task completion time $t$:
   if $R_a$ not empty
    Scheduler()
   else
    for every $D_i$ ACTIVE
     TryShutDown($D_i$)
   end if
3 At scheduled device management points:
   switch(DMPType)
   case ENABLE:DeviceCntrlRoutine($FR_j$,mode)
   case DISABLE:TerminateFR($D_j$)
   case ACTIVATE:WakeUp($D_j$)
   case DEACTIVATE:ShutDown($D_j$)
```

Figure 4: The DFR Actions

```
Scheduler():
 $\tau_h$ = highest priority task in $R_a$
 for every $D_i$ ACTIVE and not used by $\tau_h$
  TryShutDown($D_i$)
 Dispatch $\tau_h$
```

Figure 5: The scheduler function

lengths. The operations described above are also performed on all active devices whenever a task completes and the ready queue is empty.

At device management points the system takes required actions. As said, there can be a match involving four DMP types. First, if this is a scheduled FR start time then *DeviceCntrl-Routine()* is called. If this routine is called in FORCE mode then FR is immediately activated. The FORCE mode operation helps make sure the devices continue in sleep state by postponing device activations. In DEFAULT mode, a decision to start or postpone FR is made. If the device is already in sleep state or is active, and being used by the current job there is no motive to start a forbidden region; hence, it is postponed. FR is also postponed when corresponding devices are transitioning from active to sleep or sleep to active. When the device is active and not in use we again perform a check to see if the device can be shutdown using CEEDS. If this succeeds, we postpone FR else start FR and shutdown the device.

Second, if this is a scheduled FR end time we schedule the next FR. Whenever an FR starts or terminates, it is essential to invoke the scheduler, as jobs may migrate between R_a and R_b.

Third, if this corresponds to a device activation time we check to see if this activation can be postponed. The rationale behind doing this is that between the time a device activation was scheduled and is performed, a forbidden region may have been postponed or started, both of which may enable the device to continue in sleep state. Thus, the activation of the device may be postponed reflecting the new state of the system. Also in this function, we take actions to deal with forceful FR start which was explained earlier. If device activations cannot be postponed, then we start switching the device to *active* state immediately. This is accomplished through the *WakeUp()* function.

Lastly, in *ShutDown()* function, we start switching the device to sleep state and schedule an activation time for the device in future. This activation time is computed based on either the next device usage time or forbidden region end point.

Run-time Complexity: At every DMP, the system may have to perform at most two actions for *every* device: ENABLE/DISABLE FR and/or ACTIVATE/DEACTIVATE device. The next device usage time information can be computed in $O(n)$ time for each device. Therefore the algorithm runs in $O(mn)$ time at each invocation point. Since the number of devices is typically small compared to the number of tasks in the system, the DFR's run-time complexity can be considered as linear in the number of tasks in the system.

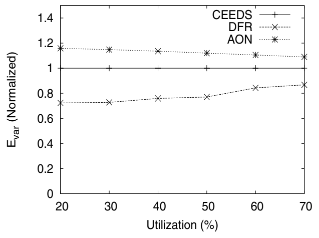
Extensions to DVS Settings with Dynamic Slack Reclaiming: The DFR algorithm can be used in DVS settings with any feasible task speed assignment. Further, we use an extension of SDRA algorithm [2] to fixed-priority settings for reclaiming. In SDRA, at dispatch time a job computes its 'earliness' by considering the slack of all higher priority jobs. Earliness represents the total slack available in the system from higher priority tasks that the current job can reclaim. Earliness is modeled through maintaining the α-queue, which represents the canonical schedule in which all jobs take their worst-case execution time. All jobs, upon arrival, insert their worst case execution times (under the *nominal* speed) to the α-queue. Jobs in the α-queue are ordered based on task priorities. A consequence of DFR algorithm is that, at dispatch time, care must be taken *not* to consider blindly all higher priority tasks in the α-queue: Some of these higher priority tasks may be currently blocked by forbidden regions. Reclaiming from such a task might indeed lead to potential deadline misses. Therefore, in DFR settings, the earliness is computed by considering only the higher priority jobs that have completed their execution.

7 Experimental Results

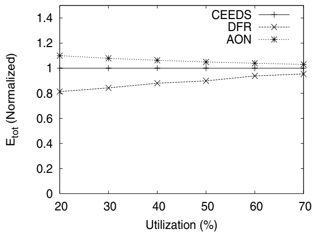
In this section, we show the performance evaluation of DFR scheme through simulations. We implemented a discrete event simulator to evaluate our scheme. We randomly generated 1000 synthetic task sets, each containing 20 periodic tasks. The periods of the tasks were randomly chosen to be in the interval [25ms, 1300ms] which corresponds to the period range observed in example real-time applications [18]. We recorded the energy consumption of the task set during the hyperperiod (LCM). Each task is assumed to use 0-2 devices, determined randomly from the device list given in [8], as done in the same paper. We adopt the device specifications provided in [8]. The simulator is modeled after the Intel Xscale processor, with specifications in [27]. On DVS settings, if the speed that we target does not correspond to the existing frequency levels, we use the next (higher) speed level.

We compare the performance of three schemes:

- **Always On (AON)**, where the devices remain in active state throughout the simulation (no dynamic power management).
- **CEEDS** [8], adapted to RMS settings. This scheme performs device shutdown and activation based solely on next device usage time predictions.
- **DFR-RMS**, in which devices are put to prolonged sleep intervals at run time through device forbidden regions. The minimum separation time (period) and duration of forbidden regions are computed through the algorithm given in Section 5.



(a) Energy Consumption (E_{var})

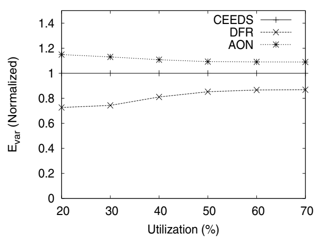


(b) Energy Consumption (E_{tot})

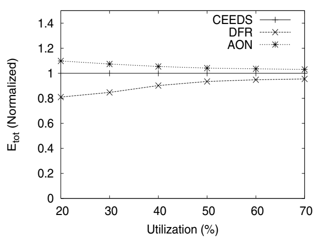
Figure 6: **Energy Consumption (No DVS)**

When evaluating the energy consumption, we present the total device variable active energy (E_{var}) and the total system energy (E_{tot}), separately. As discussed in Section 2.2, the DPM algorithms do not have a direct impact on E_{fixed}, but only on E_{var}, which can be reduced to the extent the algorithm is successful in increasing the length of sleep intervals and reducing unnecessary transitions. At the second level, we also include E_{tot} values, showing the impact of DPM schemes on overall system energy.

First, we consider settings where the actual execution time of each job is equal to its worst-case execution time. We compare the three schemes as a function of utilization. The task utilizations are scaled from 0.2 to 0.7. Figure 6 shows the relative performance of the DPM schemes, on settings where all jobs run at the maximum CPU speed (no DVS). All results are normalized with respect to the energy consumption of CEEDS. At lower utilization values, DFR-RMS effectively postpones device activations using forbidden regions. This helps in achieving prolonged sleep periods and clustered active periods where the device is in continuous use. As a result, DFR-RMS reduces E_{var} significantly: we obtain E_{var} gains in the range of 14% to 27% when compared to CEEDS. DFR-RMS scheme successfully reduces E_{tot} by 5% to 19%,

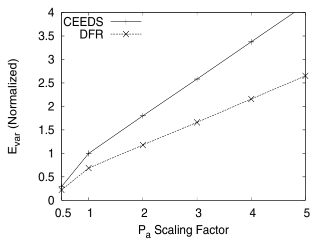


(a) Energy Consumption (E_{var})

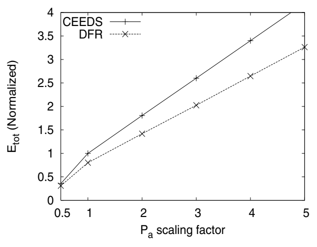


(b) Energy Consumption (E_{tot})

Figure 7: **Energy Consumption (with DVS)**



(a) Energy Consumption (E_{var})



(b) Energy Consumption (E_{tot})

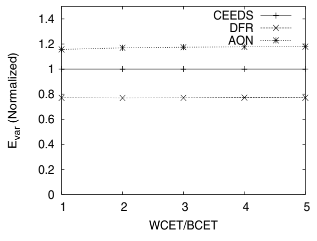
Figure 8: **Energy Consumption with P_a scaling**

as well. At high utilization values, the device idle intervals are strongly constrained by device usage patterns, periods of tasks and break-even times of devices. These factors reduce the gains obtained by DFR-RMS scheme at high utilizations. Notice that even at 70% utilization we are able to schedule some forbidden regions and obtain gains. This is due to the fact that schedulability of the task set is checked through time demand analysis given in Section 4 and forbidden regions are successfully able to exploit the CPU idle times.

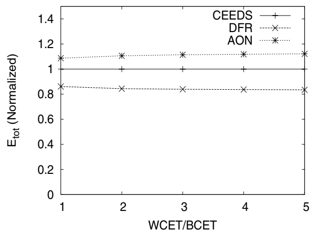
We repeated the above experiments for DVS settings (Figure 7). An energy-efficient speed threshold for each task is computed through the technique given in [2]. Then, task level speed assignments are computed by the algorithm in [2] with feasibility bound set to the Liu-Layland bound for 20 tasks (approximately 70 %). The results are similar showing DFR-RMS's gains on DVS settings as well. Even though the *relative* performance of DFR-RMS is the same in both settings, we found that the absolute E_{tot} values for DPM schemes with DVS are lower than those without DVS, showing the positive impact of DVS.

In the next experiment set, we show the impact of varying device characteristics on DPM schemes in general. In this setup, we compare CEEDS and DFR-RMS. DVS is enabled and nominal speed values are assigned as above. The system utilization is fixed at 40%. For a given task set and a device usage pattern, we first record E_{var} and E_{tot} values for P_a values given in [8]. Then, for the same task set and device usage pattern, we repeat the experiment by scaling up/down P_a for all devices and re-computing break-even times. Figure 8 shows the results normalized with respect to energy consumption of the task set with unscaled P_a values, i.e. when the scaling factor is one. Note that, CEEDS forces a device to be in active state at the predicted next device usage time, however the device may not be used at this point. With DFR-RMS scheme, DFR postponements and forceful DFR starts at next device usage times help prolong to extend sleep intervals. This makes DFR scheme more tolerant to P_a scaling. Thus, the gains of DFR scheme increase significantly as we scale up P_a, indicating that DFR would be even more applicable in systems with significant power consumer devices.

Finally, we consider the effect of reclaiming on DFR scheme. Again, the utilization is set to 40%. The actual workload is varied randomly between best case execution time (BCET) and worst case execution time (WCET) of the task. We vary the ratio of $\frac{WCET}{BCET}$ from 1 to 5. The SDRA algorithm [2] is adopted for reclaiming. As mentioned in Section 6, a consequence of DFR algorithm is that it is possible to reclaim slack only from higher priority tasks that have completed execution in the α-queue. This somewhat limits the gains for DFR. In fact, the *relative* performances of CEEDS and DFR appear to be mostly uniform throughout the spectrum.



(a) Energy Consumption (E_{var})



(b) Energy Consumption (E_{tot})

Figure 9: **Energy Consumption with reclaiming enabled**

8 Conclusions

In this paper, we addressed the problem of efficient device power management for hard real time systems. The problem of generating feasible schedules such that every device idle interval is longer than the corresponding device break-even times turns out to be NP-Hard in the *strong sense*. Then, we proposed a novel and efficient online DPM algorithm, built on the idea of *Device Forbidden Regions (DFR)*. Our scheme explicitly inserts idle intervals to the schedule at run-time to help transition devices to sleep state. We extended the well-known Time-Demand Analysis technique to determine the feasibility in the presence of forbidden regions and developed an algorithm to assign the forbidden region parameters to improve the energy savings. Through experimental evaluation, we showed how our algorithm can manage to significantly reduce the unnecessarily consumed device active power, by imposing sleep intervals that are guaranteed to exceed the break-even times. The gains are also reflected in total system energy figures.

The DFR algorithm can be used in conjunction with or without DVS: our results show that it is equally effective in both cases. Further, the well-known dynamic reclaiming algorithms can be used in conjunction with DVS and DFR.

References

[1] N. Audsley, A. Burns, K. Tindell, M. Richardson and A. Weilings. Applying new scheduling theory to static priority preemptive scheduling, In *Software Engineering Journal,* Vol. 8, No. 5, pp 284-295, 1993.

[2] H. Aydin, V. Devadas, D. Zhu. System-level Energy Management for Periodic Real-Time Tasks. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'06)*, 2006.

[3] H. Aydin, R. Melhem, D. Mossé and P. Mejia-Alvarez. Power-aware Scheduling for Periodic Real-time Tasks. *IEEE Transactions on Computers*, Vol. 53, No. 10, pp. 584-600, 2004.

[4] L. Benini, A. Bogliolo, and G. D. Micheli . A Survey of Design Techniques for System-Level Dynamic Power Management. In *Proc. of the IEEE Transactions on VLSI Systems,* Vol. 8, No. 3, pp 299-316, 2000.

[5] E. Bini, G.C. Buttazzo and G. Lipari. Speed Modulation in Energy-Aware Real-Time Systems. in *Proc. of the IEEE Euromicro Conference on Real-Time Systems (ECRTS'05)*, 2005.

[6] H. Cheng and S. Goddard. Integrated Device Scheduling and Processor Voltage Scaling for System-wide Energy Conservation. In *Proc. of the Int'l Workshop on Power-Aware Real-Time Computing (PARC'05)*, 2005.

[7] H. Cheng and S. Goddard. EEDS-NR: An online Energy-Efficient I/O Device Scheduling Algorithm for Hard Real Time Systems with Non-Preemptive resources.In *Proc. of the IEEE Euromicro Conference on Real Time Systems (ECRTS'06)*, 2006.

[8] H. Cheng and S. Goddard. Online Energy-Aware I/O Device Scheduling for Hard Real Time Systems. In *Proc. of Design Automation and Test in Europe (DATE'06)*, 2006.

[9] H. Cheng and S. Goddard. SYS-EDF: A System-wide Energy-efficient Scheduling Algorithm for Hard Real Time Systems. In *the International Journal of Embedded Systems on Low Power Real-Time Embedded Computing,* Vol. 4, No. 4, pp.45-56, 2006.

[10] Enrico Bini, Giorgio Buttazzo and Giuseppe Buttazzo. A Hyperbolic Bound for the Rate Monotonic Algorithm. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS'01)*, 2001.

[11] V. Devadas and H. Aydin. Real-Time Dynamic Power Management through Device Forbidden Regions. *Technical Report.* Computer Science Department, George Mason University, October 2007. Available online at http://www.cs.gmu.edu/˜aydin/tr-2007-23.pdf

[12] X. Fan, C. Ellis, and A. Lebeck. The Synergy Between Power-aware Memory systems and Processor Voltage. In *Workshop on Power-Aware Computing Systems (PACS'03)*, 2003.

[13] R. Jejurikar and R. Gupta. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design (ISLPED'04)*, 2004.

[14] R. Jejurikar, C. Pereira, and R. Gupta. Leakage Aware Dynamic Voltage Scaling for Real-time Embedded Systems. In *Proceedings of the Annual Conference on Design Automation (DAC'04),* 2004.

[15] J. Lehoczky, L. Sha and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behaviour. In *Proc. of Real-Time Systems Symposium (RTSS'89)*, 1989.

[16] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. In *Journal of the Association for Computing Machinery,* Vol. 20, No. 1, pp 46-61, 1973.

[17] J. Liu. *Real Time Systems* Prentice Hall, NJ, 2000.

[18] D.C. Locke, D. Vogel, T. Mesler. Building a predictable avionics platform in Ada: a case study. In *Proc. of the Real-Time Systems Symposium (RTSS'91)*, 1991.

[19] Y. Lu, L. Benini, and G. D. Micheli. Power Aware Operating systems for interactive systems. In *Proc. of the IEEE Transactions on VLSI Systems,* Vol. 10, No. 2, pp 119-134, 2002.

[20] P. Pillai and K. G. Shin. Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems. In *Proceedings of the ACM Symposium on Operating Systems Principles*, 2001.

[21] S. Saewong and R. Rajkumar. Practical voltage-scaling for fixed-priority RT-systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03),* 2003.

[22] D. Snowdon, S. Ruocco and G. Heiser. Power Management and Dynamic Voltage Scaling: Myths and Facts. In *Proc. of the Int'l Workshop on Power-Aware Real-Time Computing (PARC'05)*, 2005.

[23] V. Swaminathan, K. Chakrabarty and S. S. Iyengar. Dynamic I/O Power Management for Hard Real Time Systems.In *Proc. of the International Conference on Hardware-Software Codesign and System Synthesis (CODES'01)*, 2001.

[24] V. Swaminathan and K. Chakrabarty. Energy Conscious Deterministic I/O device Scheduling in Hard Real Time Systems.In *Proc. of the International Conference in Computer Aided Design (ICCAD'03)*, 2003.

[25] V. Swaminathan and K. Chakrabarty. Pruning-Based, Energy-Optimal Deterministic I/O Scheduling for Hard Real-Time Systems.In *ACM Transactions on Embedded Computing Systems,* Vol. 4, No. 1, pp 141-167, 2005.

[26] M. Weiser, B. Welch, A. Demers and S. Shenker. Scheduling for Reduced CPU energy. In *USENIX Symposium on Operating Systems Design and Implementation*, 1994.

[27] R. Xu, C. Xi, R. Melhem and D. Mosse. Pratical PACE for Embedded Systems. In *Proceedings of the ACM International Conference on Embedded Software (EMSOFT'04)*, 2004.

[28] J. Zhuo and C. Chakrabarti. System-level energy-efficient dynamic task scheduling. In *Proceedings of the Conference on Design Automation (DAC'05)*, 2005.