Speed Modulation in Energy-Aware Real-Time Systems

Enrico Bini
Scuola Superiore S. Anna, Italy
e.bini@sssup.it

Giorgio Buttazzo
University of Pavia, Italy
buttazzo@unipv.it

Giuseppe Lipari
Scuola Superiore S. Anna, Italy
lipari@sssup.it

Abstract

This paper presents a general framework for analyzing and designing embedded systems with energy and timing requirements. A set of realistic assumptions is considered in the model in order to apply the results in practical real-time applications. For example, the processor is assumed to have as a set of discrete operating modes, each characterized by speed, power consumption. The transition delay between modes is considered. To take I/O operations into account, task computation times are modeled with a part that scales with the speed and a part having a fixed duration. Given a set of real-time tasks, the proposed method allows to compute the optimal sequence of voltage/speed changes that approximates the minimum continuous speed which guarantees the feasibility of the system. The analysis is performed both under fixed and dynamic priority assignments.

1. Introduction

The number of embedded systems operated by batteries is increasing in different application domains. In these systems, reducing the energy consumption is of primary importance to prolong their lifetime. For this reason, a new generation of processors [13, 21, 30, 12] allow the application to dynamically vary the voltage and the operating frequency to balance computational speed versus energy consumption.

At the operating system level, suitable scheduling policies have been proposed in the literature to exploit voltage variable processors. Such policies are referred to as Dynamic Voltage Scheduling (DVS), because the scheduler, in addition to selecting the executing task, has also to select the operating voltage and frequency.

We distinguish between *static* and *dynamic* DVS. Static techniques use off-line parameters, such as periods (or minimum interarrival times) and worst-case execution cycles (WCECs), to select the appropriate voltage/speed operating mode to be used. Dynamic techniques (based on *slack reclamation*) take advantage of early completions of tasks to further reduce the speed and save more energy [3, 27].

Static DVS can be further divided in two classes. In the first class, a single optimal speed is computed off-line and never changed. Pillai and Shin [22] derived the minimal speed that can make a task set schedulable under EDF, and proposed a near-optimal method under RM. Saewong and Rajkumar provided an algorithm to find the optimal speed value for fixed priority assignments [26], assuming that the speed of the processor can be varied continuously in a given range. In practice, however, processors provide a finite number of discrete speeds. If the optimal speed is not available on a processor, it has to be approximated with the closest available discrete level higher than the optimal one. This solution, however, may cause a waste of computational capacity and, consequently, of energy, especially when the number of available speeds is small. For this reason, Ishihara and Yasuura [14] modeled processors with a limited number of operating frequencies. However they did not consider speed switching overhead and task preemptions.

In a second class of static DVS methods, the processor speed is not fixed but **statically** decided before system execution based on the task parameters. Some of these methods propose to assign a different speed to each task [3, 26]. Some others adopt a more general scheme, where the speed switching instants are more freely chosen and, typically, occur at the activation/deadline of some job [31, 19]. The energy saved by these methods is higher because the processor speed can be tightly shaped in order to provide the minimum number of cycles needed in every interval.

A major drawback of this approach, which prevents its use in real-world applications, derives from the tight relationship established between the schedule and the power management scheme. If, for some reason, some task activation is lost or delayed, the entire speed assignment is affected, resulting in a potential domino effect on the other tasks in the system, which could miss their deadlines. In this sense, such a speed assignment scheme is fragile because it is affected by the misbehavior of a task. Running always at a fixed speed is a more robust design practice, because it avoids this potential problem.

Another weakness of many energy-aware algorithms proposed in the literature is due to the set of assumptions, often not realistic, which are made to simplify the solution. Besides considering continuous voltage scaling, most methods neglect the delay due to a voltage transition. In some approaches [15, 20] such a delay is considered in the processor model, but the methods have been developed only for

Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05)
0-7695-2400-1/05 $20.00 **IEEE**

dynamic techniques aimed at reducing the slack time.

Another simplifying hypothesis usually made for reducing the complexity of the schedulability analysis is to consider tasks with relative deadlines equal to periods [22], so that task set feasibility can be checked using the simple Liu and Layland utilization bound [18], both under RM and EDF scheduling. Notice that, under fixed priority scheduling, the use of the utilization bound is even more restrictive, because the Liu and Layland schedulability test is only sufficient, leaving many feasible task sets out of consideration, thus preventing optimal solutions.

1.1. Contributions of the paper

In this paper, we present a general framework for analyzing and designing embedded systems with energy and timing requirements. The proposed approach allows minimizing energy consumption while guaranteeing task deadlines. Our method can be classified as a static DVS algorithm, in that it is able to compute off line the optimal sequence of voltage/speed changes that minimize energy consumption while guaranteeing the absence of deadline misses. In addition, a major contribution of this work is to consider more realistic assumptions in the model, which allow the method to be used in practical applications. In particular, the proposed method presents the following characteristics:

- The algorithm applies to a set of periodic (or sporadic) tasks, where deadlines are allowed to be less than or equal to periods (or minimum interarrival times).
- The algorithm is independent of the task schedule, so it is robust against potential domino effects due to the misbehavior of one or more tasks.
- It does not assume a continuous range of available speeds in the processor, but a set of discrete operating modes, each characterized by speed, power consumption, and transition delay.
- A more accurate task model, introduced by Seth et al. [28], is considered in the analysis to take into account the effects of modern processors with variable speed. According to this model, task computation times consist of a part that scales with the speed and a part having a fixed duration (typically due to the instructions accessing the external bus).
- The analysis is presented both for fixed and dynamic priority systems, and it is easily extendible to any other scheduling policy.
- The minimal energy solution within the proposed scheme is found, since the algorithm is based on exact schedulability analysis.
- The proposed method provides a general framework to describe the schedulability domain, thus enabling the user to select the appropriate design parameters based on a given cost function.

2. System model

We consider a set of n periodic or sporadic tasks that have to be scheduled on a single processor with voltage control capabilities. A task τ_i is a sequence of jobs $\tau_{i,k}$ ($k = 1, 2, \ldots$), each characterized by a number C_i of worst-case execution cycles (WCECs), a minimum interarrival time T_i (often referred to as the task period), and a relative deadline $D_i \leq T_i$. Tasks are fully preemptive and do not perform blocking operations. Note that intertask communication can still be performed using non-blocking mechanisms, as Cyclic Asynchronous Buffers [7].

As observed by Seth et al. [28], not all execution cycles scale with the processor speed, because some operations deal with memory or other I/O devices, whose access time is fixed. The typical example is provided by a memory read: if the data to be read is present in the cache, then the instruction runs at the speed of the processor and so it scales with it. On the other hand, if a cache miss occurs the data is read from the bus. In this case, the duration of the operation is imposed by the bus clock that does not scale with the processor speed.

To take this into account, the number C_i of worst-case execution cycles required by a task is split in two portions: c_i (processor cycles) scales with the clock frequency and m_i (seconds) does not. Thus we have [28]:

$$C_i = c_i + \alpha\, m_i \tag{1}$$

where α is the speed in cycles per second ($\mathsf{cyc}/\mathsf{sec}$).

2.1. Modeling the processor

In CMOS circuits, the power consumption due to dynamic switching dominates the power lost by leakage currents, and the dynamic portion of power consumption is modelled by well known polynomial relationships [8, 11]. However, as the integration technology advances, it is expected that the leakage will significantly affect, if not dominate, the overall energy consumption in integrated circuits (ICs) [10, 25]. Very recently, some work addressed the issue of scheduling a real-time application while reducing the leakage power as well [24]. Also, an important fraction of the consumed energy depends on the memory. It has been shown [23] that at low frequencies the energy consumption is dominated by the memory. At high frequencies the processor core dominates the power consumption.

All these remarks have led us to formulate a model for the processor energy consumption, which generalizes all the former works. Throughout the paper we assume that a power-aware processor is characterized by a set $\mathcal{M} = \{\Lambda_1, \Lambda_2, \ldots, \Lambda_p\}$ of p operating modes, where each mode $\Lambda_k = (\alpha_k, p_k)$ is described by two parameters: α_k is the processor speed in mode k and it is measured as number of cycles per second ($\mathsf{cyc}/\mathsf{sec}$); p_k is the power consumed in mode k, measured in Watts.

In a recent work, AbouGhazaleh et al. [1] proposed a

detailed model for the speed switching overhead, considering *software* causes, due to the new speed computation, and *hardware* causes, due to power management circuits.

Following their scheme, in this paper, the overhead is taken into account through a *matrix of overheads* $\mathcal{O}$, where each element $o_{i,j}$, $i \neq j$, is the time overhead required to switch from the operating mode Λ_i to Λ_j. Moreover, we assume that the power consumption during the transition from Λ_i and Λ_j is p_j. Note that, assuming a more detailed model (such as considering the additional power for the DC-DC regulator) is possible and it does not affect the schedulability analysis, nor the validity of the presented results.

To handle the complexity of our system model we make use of some recent results in the field of hierarchical scheduling [2, 9, 17, 29]. The advantage of using such an approach is that the mechanism managing the processor speed may be seen as a server providing processor cycles to the requesting application. In this way, the speed management can be decoupled from the application events, thus achieving a higher degree of robustness. Using this notion, the problem is split in two separate subproblems: (1) the characterization of the number of cycles demanded by the application and (2) the characterization of the number of cycles provided by the server mechanism.

3. Application demand analysis

In this section we present a model for expressing the computational demand of the application to the processing unit. We consider two major scheduling strategies: Earliest Deadline First (EDF) and Fixed Priority (FP) scheduling.

3.1. EDF analysis

The feasibility of a periodic task set under EDF can be analyzed through the Processor Demand Criterion, proposed by Baruah, Howell and Rosier [4], according to which a set of periodic tasks simultaneously activated at time zero can be feasibly scheduled by EDF if and only if:

$$\forall t \in \mathsf{dSet} \qquad \sum_{i=1}^{n} \mathsf{jobs}_i(t)\, C_i \leq t \tag{2}$$

where $\mathsf{jobs}_i(t)$ is the number of jobs of task τ_i having arrival time and deadline in the interval $[0, t]$ and dSet is the set of all time instants where the test has to be performed.

It has been proved that, given a value of α, the set dSet can be effectively computed and it is the set of deadlines within the first busy period [4]. Unfortunately, the length of the busy period depends on α, as well. Hence we assume dSet to be equal to the entire set of all deadlines before the hyperperiod. It is still an open question whether the set of points in dSet can be tightly reduced. However the validity of the presented results is not affected by this improvement.

If the processor runs at a constant fraction α of the nominal speed ($\alpha \leq 1$), only αt cycles are available in $[0, t]$ and, considering the execution model given in equation (1), the schedulability condition becomes:

$$\forall t \in \mathsf{dSet} \qquad \sum_{i=1}^{n} \mathsf{jobs}_i(t)\, (c_i + \alpha\, m_i) \leq \alpha\, t. \tag{3}$$

We can derive the condition that α has to satisfy in order to guarantee the schedulability of the task set:

$$\forall t \in \mathsf{dSet} \qquad \alpha \geq \frac{\sum_{i=1}^{n} \mathsf{jobs}_i(t)\, c_i}{t - \sum_{i=1}^{n} \mathsf{jobs}_i(t)\, m_i}. \tag{4}$$

Then, the minimum speed α_{opt} that ensures feasibility is

$$\alpha_{\text{opt}} = \max_{t \in \mathsf{dSet}} \frac{\sum_{i=1}^{n} \mathsf{jobs}_i(t)\, c_i}{t - \sum_{i=1}^{n} \mathsf{jobs}_i(t)\, m_i}. \tag{5}$$

When relative deadlines are equal to periods, it is known that the maximum occurs when t is equal to the hyperperiod $H = \text{lcm}(T_1\, T_2, \ldots, T_n)$, thus we have that:

$$\alpha_{\text{opt}} = \frac{\sum_{i=1}^{n} c_i/T_i}{1 - \sum_{i=1}^{n} m_i/T_i} \tag{6}$$

which is equivalent to the result provided in [28].

3.2. FP analysis

When using a fixed priority assignment, the necessary and sufficient feasibility condition is:

$$\forall i = 1, \ldots, n \quad \exists t \in \mathsf{tSet}_i \qquad C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t$$

where tSet_i is the set of schedulability points [16, 6] relative to task τ_i, where the test has to be performed.

Considering a processor running at speed α and using the more complete model for the task computation times [28], we have

$$c_i + \alpha\, m_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil (c_j + \alpha\, m_j) \leq \alpha\, t.$$

Hence, the optimal speed α_{opt} is given by:

$$\alpha_{\text{opt}} = \max_{i=1,\ldots,n} \min_{t \in \mathsf{tSet}_i} \frac{c_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil c_j}{t - m_i - \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil m_j}, \tag{7}$$

which provides the minimum speed the processor can run to feasibly schedule the task set with fixed priorities.

4. Power management

Once the application demand has been characterized and the ideal speed α_{opt} computed, different techniques can be adopted to minimize the power consumption. In the unlikely case of availability of an operating mode Λ_k running exactly at the desired speed α_{opt}, we simply select it. Otherwise we have to properly manage the processor operating modes to minimize the energy consumption. To characterize the effects of the management scheme onto the application, we will follow the demand/supply approach [2, 9, 17, 29]. This framework has been successfully

proposed to model a hierarchical scheduler for an application that uses a fraction of the computational resource. The key idea is that the time demanded by the application must never be greater than the time supplied.

Following this approach the number of cycles supplied by the processor is modeled using a function $Z(t)$, defined as *the minimum number of cycles the processor can provide in every interval of length* t. More formally, if $\alpha(t)$ denotes the processor speed at time t, $Z(t)$ can be defined as

$$Z(t) = \min_{t_0} \int_{t_0}^{t_0+t} \alpha(x)\,\mathrm{d}x. \tag{8}$$

We now consider the problem of expressing the proper supply function $Z(t)$ when a specific speed handling policy is adopted for the processor.

If we fix the operating mode Λ_k such that $\alpha_k \geq \alpha_{\text{opt}}$ and p_k is minimum, then the switching overhead is not considered, because the speed is never changed. In this case the supply function $Z(t)$ is simply given by

$$Z(t) = \alpha_k\, t. \tag{9}$$

Note that this is the most used method among the static approaches. However, if the gap between the selected speed α_k and the optimal one α_{opt} is too high and the power consumption is a critical design parameter, it is better to adopt a different approach.

As suggested by Ishihara et al. [14], we propose to switch between two operating modes, Λ_L and Λ_H, such that $\alpha_L < \alpha_{\text{opt}} < \alpha_H$. Such a switching scheme will be referred to as the PWM-mode, for the similarity with the pulse width modulation technique used to drive DC servomotors. When using a PWM-mode, however, the speed switching overhead has to be considered. An example of the speed alternation scheme is illustrated in Figure 1.

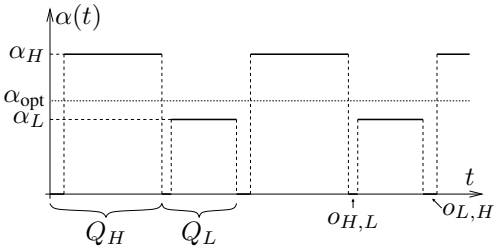

Figure 1. An example of PWM-mode.

The effective speed α_{eff} achieved by the processor staying for Q_L in mode Λ_L and Q_H in mode Λ_H can be computed as follows:

$$\alpha_{\text{eff}} = \frac{\alpha_H\, Q_H + \alpha_L\, Q_L}{Q_H + Q_L} - \frac{\alpha_H\, o_{L,H} + \alpha_L\, o_{H,L}}{Q_H + Q_L}. \tag{10}$$

Notice that the overheads $o_{H,L}$, $o_{L,H}$ are included within the length of Q_L, Q_H respectively.

In Equation (10) we can clearly notice the ideal term and the loss due to the switching overhead. Notice that, the presence of the overhead does not allow Q_L and Q_H to be arbitrarily small, hence we need to specifically impose α_{eff} to be greater than α_L (it would not make sense to use this scheme if the effective speed is less than α_L). Thus, imposing the condition $\alpha_{\text{eff}} > \alpha_L$ we find that

$$Q_H > \frac{\alpha_H\, o_{L,H} + \alpha_L\, o_{H,L}}{\alpha_H - \alpha_L}.$$

In addition, Q_L and Q_H need to be greater than $o_{H,L}$ and $o_{L,H}$, otherwise no useful processor cycles are available for the task set. The power consumed in the PWM-mode can also be expressed as a function of Q_L and Q_H as[1]:

$$p_{\text{eff}} = \frac{p_H\, Q_H + p_L\, Q_L}{Q_H + Q_L} \tag{11}$$

which is less than p_H, the power that would be consumed if the processor were continuously running in mode Λ_H, since $Q_L > 0$ and $Q_H > 0$. The power saving of the PWM-mode can be explicitly computed as follows:

$$p_{\text{save}} = p_H - p_{\text{eff}} = \frac{1}{1 + Q_H/Q_L}(p_H - p_L). \tag{12}$$

Equation (12) shows that the power saving increases as the ratio Q_H/Q_L decreases.

4.1. Selecting Λ_L and Λ_H

In this section we show how to select the two modes to reduce power consumption. Due to the convexity of the power-speed relationship [25], the speed pair (α_L, α_H) which minimizes the power consumption in the PWM-mode is given by the two speeds closest to α_{opt}. However the power-speed relationship may be different than the ideal polynomial function [25] and there may be modes with the same speed, but different power consumption (due to different voltage). Also the presence of switching overhead alters the ideal speeds because some processor cycles are wasted during the speed transition time.

A convenient way to illustrate how to derive the mode pair (Λ_L, Λ_H) which minimizes the power consumption is to represent the operating modes of the processor in a power/speed graph.

A qualitative example is shown in Figure 2, where the black dots represent the operating modes and the dashed lines contain all the possible pairs $(\alpha_{\text{eff}}, p_{\text{eff}})$ achievable by varying Q_L and Q_H, which can reproduce the speed α_{opt}. For the moment the influence of the overheads is not shown in the figure.

Among the possible pairs, the one which provides the desired speed α_{opt} and, at the same time, minimizes the power consumption can be obtained by the lowest intersection between the dashed lines (representing the mode pairs) and the vertical line at speed α_{opt}. It is worth noticing that, due to the convexity of the power/speed relationship, the optimal pair (Λ_L, Λ_H) is always given by the two points closest to α_{opt} [14]. We now show how to consider the overhead due

[1] Remember we are assuming that the power consumed in mode Λ_k is equal to p_k for the entire duration of Q_k.

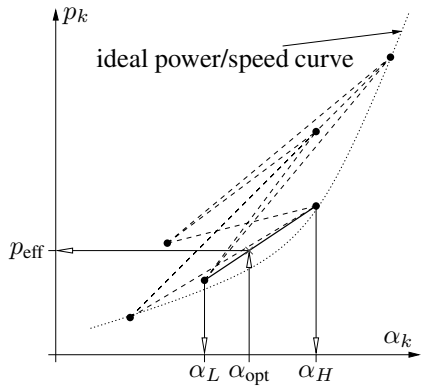

Figure 2. The operating modes in the power/speed space.

to speed switching.

If we set $f = \frac{1}{Q_L+Q_H}$ as the frequency of the PWM-mode scheme, and $\lambda_L = Q_L f$ as the portion of the period $1/f$ running at mode Λ_L, Equation (10) can be rewritten as follows:

$$\begin{aligned}\alpha_{\text{eff}} &= \lambda_L\alpha_L + (1-\lambda_L)\alpha_H - (o_{H,L}\alpha_L + o_{L,H}\alpha_H)f \\ \alpha_{\text{eff}} &= \lambda_L(\alpha_L - \Delta_{L,H}f) + (1-\lambda_L)(\alpha_H - \Delta_{L,H}f)\end{aligned} \tag{13}$$

where $\Delta_{L,H} = o_{H,L}\alpha_L + o_{L,H}\alpha_H$ is the number of processor cycles we lose because of the transition delay.

From Equation (13) the effect of the overhead is highlighted: **the introduction of the overhead $o_{H,L}, o_{L,H}$ in the PWM-mode (Λ_L, Λ_H) is equivalent to left-translating the dashed lines in the power/speed space by an amount equal to $\Delta_{L,H}f$.** Since the optimal choice of the two modes depends on the frequency f, it may happen that a particular pair is only optimal for some range of frequency f.

The optimal pair (Λ_L, Λ_H) for a given frequency f can be found using a simple polynomial algorithm implemented by means of a Matlab code prototype [5]. From now on we assume the two operating modes (Λ_L, Λ_H) are fixed. Note that due to the presence of the overheads, mode Λ_L and Λ_H may not be adjacent.

4.2. Processor supply function

This section describes how to derive the processor supply function $Z(t)$ when the CPU operates according to the PWM-mode illustrated in Figure 1. Since $\alpha(t)$ has periodicity $P = Q_L + Q_H$, we can restrict the study of $Z(t)$ in the interval $[0, P)$, in fact:

$$Z(t) = Z(t - kP) + k\,\alpha_{\text{eff}}P \tag{14}$$

so that the $Z(t)$ only needs to be defined in $[0, P)$.

Due to the speed switching overhead, the longest time where no processor cycles are available is $o_{\max} = \max\{o_{L,H}, o_{H,L}\}$. For this reason $Z(t) = 0$ for $t \in [0, o_{\max})$. Then, for $t \geq o_{\max}$ some cycles are available. In the worst case, the available cycles increase with the speed of α_L. This amount of processor cycles is provided for $Q_L - o_{H,L}$. Then the second (shorter) switching overhead occurs. Finally, in the last part of the period P, the cycles are provided at the maximum speed α_H. The resulting profile of $Z(t)$ in the interval $[0, P)$ is the following:

$$Z(t) = \begin{cases} 0 & t \in [0, o_{\max}) \\ \alpha_L(t - o_{\max}) & t \in [o_{\max}, o_{\max} + Q_L - o_{H,L}) \\ \alpha_L(Q_L - o_{H,L}) & t \in [o_{\max} + Q_L - o_{H,L}, Q_L + o_{L,H}) \\ \alpha_H(t-P) + \alpha_{\text{eff}}P & t \in [Q_L + o_{L,H}, P) \end{cases} \tag{15}$$

where:

$$\begin{aligned} P &= Q_L + Q_H \\ \alpha_{\text{eff}} &= \frac{\alpha_L(Q_L - o_{H,L}) + \alpha_H(Q_H - o_{L,H})}{Q_L + Q_H} \\ o_{\max} &= \max\{o_{H,L}, o_{H,L}\} \end{aligned}$$

The supply function $Z(t)$ for the PWM-mode is also illustrated in Figure 3.

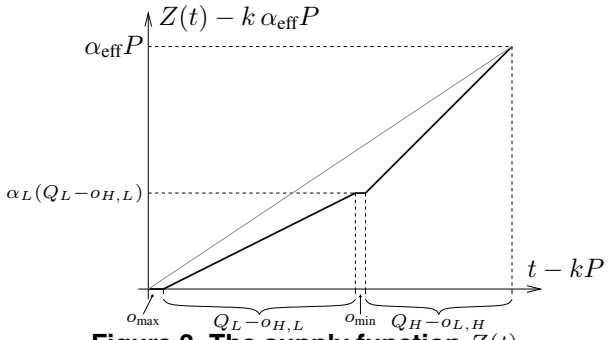

Figure 3. The supply function $Z(t)$.

5. Computing the optimal Q_L and Q_H

In this section the results achieved in Sections 3 and 4 are merged to identify the schedulability region in the (Q_L, Q_H) space, for determining the optimal pair $(Q_L^{\text{opt}}, Q_H^{\text{opt}})$ that guarantees the feasibility of the task set while minimizing energy consumption.

We first notice that a necessary schedulability condition for the task set is:

$$\alpha_{\text{eff}} \geq \alpha_{\text{opt}}$$

meaning that the effective speed cannot be smaller than the optimal speed required by the application, otherwise some deadline will be missed. To derive the necessary and sufficient schedulability region, however, we need to compute the exact supply function $Z(t)$ according to Equation (8).

When adopting the EDF scheduling algorithm, the exact schedulability condition based on the demand bound function becomes:

$$\forall t \in \mathsf{dSet} \qquad Z(t) \geq \sum_{i=1}^{n} \mathsf{jobs}_i(t)\, C_i \tag{16}$$

meaning that the demanded cycles must not exceed the pro-

vided cycles, which are modeled by the function $Z(t)$ of Equation (15).

In a similar fashion, when tasks are scheduled by fixed priorities, the exact schedulability condition is:

$$\forall i = 1, \ldots, n \quad \exists t \in \mathsf{tSet}_i \quad Z(t) \geq C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j.$$

Notice that in both scheduling algorithms the basic condition that needs to be checked can be expressed in the form:

$$Z(t) \geq W \tag{17}$$

where the scheduling algorithm only affects the way in which W is defined and the instants t where the inequality has to be verified to ensure schedulability.

We now proceed by computing the optimal pair $(Q_L^{\text{opt}}, Q_H^{\text{opt}})$ that minimizes energy consumption. To do that we assume the two speeds α_L and α_H are fixed, as derived by the procedure explained in Section 4.1. We first introduce the notion of **basic Q-domain**:

Definition 1 *The basic Q-domain $\mathbb{Q}(t, W)$ is the set of pairs (Q_L, Q_H) such that*

$$Z(t) \geq W, \tag{18}$$

where $Z(t)$ is the cycle supply function, which depends on (Q_L, Q_H), of the related PWM scheme. Formally:

$$\mathbb{Q}(t, W) = \{(Q_L, Q_H) : Z(t) \geq W\} \tag{19}$$

Using the last definition, it follows from Equation (16) that a task set scheduled by EDF will never miss a deadline in a PWM scheme **iff**:

$$\forall t \in \mathsf{dSet} \qquad (Q_L, Q_H) \in \mathbb{Q}\left(t, \sum_{i=1}^{n} \mathsf{jobs}_i(t)\, C_i\right)$$

$$(Q_L, Q_H) \in \bigcap_{t \in \mathsf{dSet}} \mathbb{Q}\left(t, \sum_{i=1}^{n} \mathsf{jobs}_i(t)\, C_i\right). \tag{20}$$

For the same reasoning, when fixed priorities are used, the set of admissible (Q_L, Q_H) is:

$$(Q_L, Q_H) \in \bigcap_{i=1,\ldots,n} \bigcup_{t \in \mathsf{tSet}_i} \mathbb{Q}\left(t, C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j\right). \tag{21}$$

We first focus our attention on finding the basic Q-domain $\mathbb{Q}(t, W)$ in general, so that Equations (20) and (21) can be computed by combining them.

The analytical expression of the set is found by inverting Equation (17), assuming $Z(t)$ as in Equation (15), thus expressing (Q_L, Q_H) as function of W, t, α_L, $o_{H,L}$, α_H and $o_{L,H}$. First we set $k = \left\lfloor \frac{t}{P} \right\rfloor$. Using the property in Equation (14), the condition of Equation (17) becomes:

$$Z(t - kP) + k\,\alpha_{\text{eff}} P \geq W \tag{22}$$

As we can see from Equation (15), the possible values of $Z(t)$ are four and they need to be considered ad hoc. In the first case, from the expansion of Equation (22), we have

$$k\,\alpha_{\text{eff}}\, P \geq W$$

$$\alpha_L Q_L + \alpha_H Q_H \geq \frac{W}{k} + \Delta_{L,H}. \tag{23}$$

In the second case (i.e. when the slope of $Z(t)$ is α_L):

$$\alpha_L(t - kP - o_{\max}) + k\,\alpha_{\text{eff}} P \geq W$$

$$Q_H \geq \frac{W + k\Delta_{L,H} - \alpha_L(t - o_{\max})}{k(\alpha_H - \alpha_L)}. \tag{24}$$

When $Z(t)$ is constant and equal to $\alpha_L(Q_L - o_{H,L})$:

$$\alpha_L(Q_L - o_{H,L}) + k\,\alpha_{\text{eff}} P \geq W$$

$$(k+1)\alpha_L Q_L + k\alpha_H Q_H \geq W + \alpha_L o_{H,L} + k\Delta_{L,H}. \tag{25}$$

In the fourth and last case we have:

$$\alpha_H(t - (k+1)P) + (k+1)\,\alpha_{\text{eff}} P \geq W$$

$$Q_L \leq \frac{\alpha_H t - (k+1)\Delta_{L,H} - W}{(k+1)(\alpha_H - \alpha_L)}. \tag{26}$$

Thanks to Equations (23), (24), (25) and (26) the region of all the feasible pairs (Q_L, Q_H) is constructed.

5.1. Example of applicability

In order to clarify the design strategy we propose two examples of PWM-mode design.

The case of one task. Let us suppose we have only one task whose data are:

- scalable computation time $c_1 = 240 \cdot 10^3 \mathsf{cyc}$;
- non-scalable computation time $m_1 = 400\,\mu\mathsf{sec}$;
- period and deadline $T_1 = 9.6\,\mathsf{msec}$.

From Equation (6) we find that:

$$\alpha_{\text{opt}} = \frac{c_1/T_1}{1 - m_1/T_1} = 26.087\mathsf{MHz}. \tag{27}$$

We suppose this speed is not available and the two closest available operating modes are $\Lambda_L = (20\mathsf{MHz}, 480\mathsf{mW})$, $\Lambda_H = (40\mathsf{MHz}, 810\mathsf{mW})$ and the overheads $o_{H,L} = 160\mu\mathsf{sec}$, $o_{L,H} = 240\mu\mathsf{sec}$. Since we do not know when the non-scalable computation time occurs during task execution (that is, when running at α_L or at α_H), we must maximize the worst-case execution cycles by setting

$$C_1 = c_1 + m_1\alpha_H = 256 \cdot 10^3 \mathsf{cyc}. \tag{28}$$

However we remind that, as shown in [28], the impact of m_i vs. c_i is minor, meaning that this overestimation is tight.

In order to schedule the task, the PWM-mode must supply at least C_1 cycles in every interval T_1. So it must be:

$$Z(T_1) \geq C_1.$$

where the overhead is already included in $Z(t)$. Notice that this condition ensures the task schedulability in both the scheduling algorithms (FP and EDF), because the two al-

gorithms coincide when only one task is in the system. The resulting set $\mathbb{Q}(T_1, C_1)$ is shown in Figure 4.

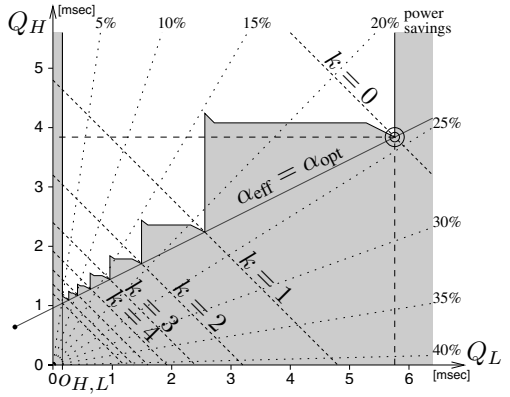

Figure 4. Schedulability region in the Q-Space.

For each value of k (remember that $k = \lfloor t/P \rfloor$) the domain boundary is composed by four segments, since $Z(t)$ is defined on four different intervals.

As expected, for small values of (Q_L, Q_H), the region $\mathbb{Q}(T_1, C_1)$ approximates the ideal fluid allocation. However, a greater amount of power is saved for big values of (Q_L, Q_H), as shown by the power savings level curves. For this reason, the pair within the admissible region that achieves the greatest power saving is at the vertex with $Q_L = 5.76$ msec and $Q_H = 3.84$ msec.

Three tasks scheduled by FP. In the second more realistic example (also available in [5]), we assume to have a processor whose operating modes are listed in Table 1. Tasks are scheduled by FP and their parameters $(c_i[\text{Kcycles}], m_i[\mu\text{sec}], T_i[\text{msec}])$ are: $\tau_1 = (10, 0, 2.2)$, $\tau_2 = (20, 100, 10)$ and $\tau_3 = (20, 20, 35)$.

k	1	2	3	4	5	6	7	8	9
a_k	0	0	2	5	10	20	40	50	80
p_k	0	1	10	20	50	50	50	200	500

Table 1. An example of processor operating modes.

Our goal is to find the pair of operating modes which guarantees feasibility and minimizes the energy consumption. By means of Equation (7) we compute the optimal speed, which results to be $\alpha_{\text{opt}} = 74.124$ MHz. Applying the rules explained in Section 4.1, we find that the two operating modes to be alternated are Λ_7 and Λ_9. Note that due to the parameters of the operating modes, Λ_8 is not the best choice for Λ_L. For the selected pair the speed switching overhead are $o_{7,9} = 20\,\mu$sec and $o_{9,7} = 0.2$ msec.

Then we maximize the number of cycles required by the non-scalable computation time m_i assuming it runs at the higher speed α_H:

$$C_i = c_i + m_i\,\alpha_H \tag{29}$$

and we obtain $C_1 = 100\,000$ cycles, $C_2 = 208\,000$ cycles and $C_3 = 201\,600$ cycles.

For task τ_1, the set of schedulability points tSet_1 only contains the deadline $D_1 = 2.2$ msec. Hence the pairs (Q_L, Q_H) that can feasibly schedule task τ_1 are within $\mathbb{Q}(D_1, C_1) = \mathbb{Q}(2.2 \cdot 10^{-3}, 10^5)$, which is above the line labeled by "τ_1", in Figure 5.

Task τ_2 has two schedulability points at $D_2 = 10$ msec and $\left\lfloor \frac{D_2}{T_1} \right\rfloor T_1 = 8.8$ msec. The region of the admissible pairs (Q_L, Q_H), also plotted in Figure 5, is then given by the union of the two basic Q-domains resulting from each schedulability point. Similarly, the set of schedulability points for τ_3, in msec, is $\text{tSet}_3 = \{35, 33, 30, 28.6\}$ and the region of admissible pairs is above the boundary labeled by "τ_3".

In Figure 5, we plot the Q-domain of the admissible pairs (Q_L, Q_H), as found from Equation (21). This final region is the intersection of the three regions previously found, because all the tasks must be schedulable. The thick black dashed line is the final domain containing the admissible pairs. As depicted in the plot, the pair that minimizes the energy corresponds to the point $Q_L = 1.2$ msec and $Q_H = 8.8$ msec. The power consumed at this working point is 446 mW, as given by Equation (11), which is 10.8% less than always running in the mode $\Lambda_H = \Lambda_9$.

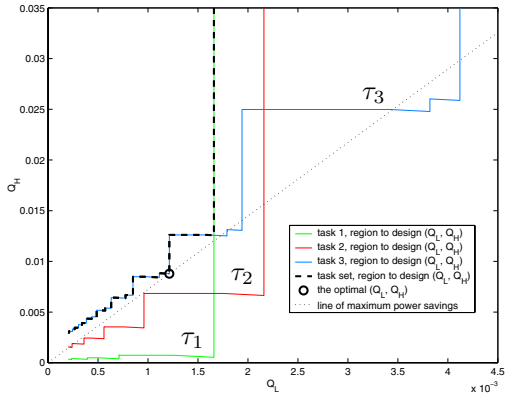

Figure 5. The Q-domain for Example 2.

6. Conclusions and future work

In this paper we presented a method for minimizing the energy consumption in periodic/sporadic task systems executing in processors with a discrete number of operating modes, each characterized by speed, power consumption, and transition delay. The proposed approach allows the user to compute the optimal sequence of voltage/speed changes that minimize energy consumption while guaranteeing the

feasibility of the schedule.

The analysis has been carried out under a set of realistic assumptions and the increased complexity has been handled through a hierarchical scheduling approach [2, 9, 17, 29], which considers the processor speed manager as a server providing processor cycles to the requesting application. By means of this separation of concerns, the problem has been divided into the analysis of the number of cycles demanded by the application and the analysis of the number of cycles provided by the processor.

This approach has the benefit of proposing a general framework to describe the schedulability domain, applicable under fixed as well as dynamic priority assignments, thus enabling the user to select the appropriate design parameters based on a given cost function.

In the future we plan to combine our static analysis to dynamic algorithms, in order to combine the advantages of our PWM-mode management with the greater amount of power savings due to reclamation of unused processor cycles.

Acknowledgements. The authors would like to thank Prof. Daniel Mossé for his valuable comments on a preliminary draft of this paper.

References

[1] N. AbouGhazaleh, D. Mossé, B. Childers, and R. Melhem. *Toward the placement of Power Management Points in Real-Time Applications*, chapter 1. Compilers and Operating Systems for Low Power. Kluwer Academic Publishers, 2002.

[2] L. Almeida, P. Pedreiras, and J. A. G. Fonseca. The FTT-CAN protocol: Why and how. *IEEE Transaction on Industrial Electronics*, 49(6):1189–1201, Dec. 2002.

[3] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(5):584–600, May 2004.

[4] S. K. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[5] E. Bini. Matlab code prototype. `http://feanor.sssup.it/~bini/pubblications/2005BinButLip.html`, Mar. 2005.

[6] E. Bini and G. C. Buttazzo. Schedulability analysis of periodic fi xed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, Nov. 2004.

[7] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Verlag, second edition, 2004. ISBN: 0-387-23137-4.

[8] A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995. ISBN: 0-7923-9576-X.

[9] X. Feng and A. K. Mok. A model of hierarchical real-time virtual resources. In *Proc. of the* 23^{rd} *IEEE Real-Time Systems Symposium*, 2002.

[10] I. T. R. for Semiconductors. International sematech. `http://public.itrs.net/`.

[11] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proc. of the RTSS*, 1998.

[12] Intel, `http://developer.intel.com/design/mobile/centrinoplatformoverview.htm`. *Intel Centrino Mobile Technology*.

[13] Intel, `http://developer.intel.com/design/intelxscale/`. *Intel XScale Technology*.

[14] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. *International Symposium on Low Power Electronics and Design*, 1998.

[15] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proc. of the* 37^{th} *DAC*, 2000.

[16] J. P. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proc. of the RTSS*, 1989.

[17] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proc. of the* 15^{th} *Euromicro Conference on Real-Time Systems*, 2003.

[18] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, Jan. 1973.

[19] Y. Liu and A. K. Mok. An integrated approach for applying dynamic voltage scaling to hard real-time systems. In *Proc. off the* 9^{th} *IEEE RTAS*, 2003.

[20] B. Mochocki, X. S. Hu, and G. Quan. A realistic variable voltage scheduling model for real-time applications. In *Proc. of the ICCAD*, 2002.

[21] Motorola, `http://e-www.motorola.com/webapp/sps/library/prod_lib.jsp`. *MPC5200: 32 bit Embedded Processor*.

[22] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. 18^{th} *ACM Symposium on Operating System Principles*, 2001.

[23] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proc. of the ACM International Conference on Mobile Computing and Networking*, 2001.

[24] G. Quan, L. Niu, X. S. Hu, and B. Mochocki. Fixed priority based real-time scheduling for reducing energy on variable voltage processors. In *Proc. of the* 25^{th} *IEEE RTSS*, 2004.

[25] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, second edition, 2002.

[26] S. Saewong and R. Rajkumar. Pratical voltage-scaling for fi xed-priority RT-systems. *Proc. of the RTAS*, 2003.

[27] C. Scordino and G. Lipari. Using resource reservation techniques for power-aware scheduling. In *Proc. of the* 4^{th} *ACM International Conference on Embedded Software*, 2004.

[28] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. FAST: Frequency-aware static timing analysis. In *Proc. of the* 24^{th} *RTSS*, 2003.

[29] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. of the* 24^{th} *RTSS*, 2003.

[30] Transmeta, `http://www.transmeta.com/crusoe/`. *The Crusoe Processor*.

[31] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. of the* 36^{th} *Annual Symposium on Foundations of Computer Science*, 1995.