Minimizing CPU Energy in Real-Time Systems with Discrete Speed Management

Enrico Bini, **Giorgio Buttazzo**, **Giuseppe Lipari**
Scuola Superiore Sant'Anna, Pisa, Italy
{e.bini,giorgio,lipari}@sssup.it

This paper presents a general framework to analyze and design embedded systems minimizing the energy consumption without violating timing requirements. A set of realistic assumptions is considered in the model in order to apply the results in practical real-time applications. The processor is assumed to have as a set of discrete operating modes, each characterized by speed and power consumption. The energy overhead and the transition delay incurred during mode switches are considered. Task computation times are modeled with a part that scales with the speed and a part having a fixed duration, to take I/O operations into account.

The proposed method allows to compute the optimal sequence of voltage/speed changes that approximates the minimum continuous speed which guarantees the feasibility of a given set of real-time tasks, without violating the deadline constraints. The analysis is performed both under fixed and dynamic priority assignments.

Categories and Subject Descriptors: category1 [**category2**]: category3

General Terms:

Additional Key Words and Phrases: Real-Time Systems, CPU energy reduction, speed modulation

1. INTRODUCTION

The number of embedded systems operated by batteries is constantly increasing in different application domains, from portable devices to mobile communication systems, autonomous robots, and distributed networks of mobile sensors. In these systems, reducing the energy consumption is of primary importance to prolong their lifetime. For this reason, a new generation of processors (such as Intel XScale, Motorola MPC5200, Transmeta Crusoe, Intel Centrino) has been built to provide different operating modes. These processors can dynamically vary the voltage and the operating frequency to balance computational speed versus energy consumption.

When the application has to satisfy real-time requirements, any energy-aware policy acting on the processor speed should also take timing constraints into account, to guarantee the timely execution of those computational activities that have to meet predefined deadlines. 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/Frequency Scaling (DVFS). In these techniques the scheduler, in addition to selecting the executing task, has also to select the operating voltage and frequency.

We distinguish between *static* and *dynamic* DVFS. 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. These methods typically try to foresee the future load requirements and then reduce the speed based on these predicted values [Qadi et al. 2003; Aydin et al. 2004; Scordino and Lipari 2006]. The energy saving achieved by this class of methods is higher than that achievable by static ones. However, since dynamic techniques are typically developed by enhancing static approaches, the interest in static techniques is still high.

Static DVFS 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 [Pillai and Shin 2001] 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 [Saewong and Rajkumar 2003], 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 [Ishihara and Yasuura 1998] modeled processors with a limited number of operating frequencies and proved that the most energy efficient technique to approximate a speed level not provided by the processor is to switch between the two closest ones. However they did not consider speed switching overhead and task preemptions. Irani et al [Irani et al. 2003] did account for the overhead to enter in sleep mode. They showed that a consequence of a non-zero overhead is that it exists a critical speed below which it is not convenient to switch into sleep mode.

In a second class of static DVFS methods, the processor speed is not fixed but **statically** decided before system execution based on the task parameters. In other words, given a set of periodic tasks, the sequence of frequency changes that have to be performed on the processor during execution can be computed off line. Since the task schedule is periodic, the *voltage schedule* obtained by this method is also periodic and can be stored in a table. Some of these methods propose to assign a different speed to each task [Aydin et al. 2004; Saewong and Rajkumar 2003]. 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 [Yao et al. 1995; Liu and Mok 2003]. 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 drawback of this approach derives from the tight relationship established be-

tween the schedule of the tasks 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. Running always at a fixed speed is a more robust design practice, and it is simpler to be implemented.

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 [Lee and Sakurai 2000; Mochocki et al. 2002] such a delay is considered in the processor model, but the methods have been developed only for 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 [Pillai and Shin 2001], so that task set feasibility can be checked using the simple Liu and Layland utilization bound [Liu and Layland 1973], 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.

Finally, it is worth mentioning that recent works have not limited their focus to minimizing the energy consumed by the CPU, but they addressed also the consumption of other devices [Jejurikar and Gupta 2004; Zhuo and Chakrabarti 2005; Aydin et al. 2006].

1.1 Contributions of the paper

In this work, we present a general framework for analyzing and designing embedded systems with energy and timing requirements. This paper extends a previous work [Bini et al. 2005] by the same authors that allows minimizing energy consumption while guaranteeing task deadlines, under discrete voltage/frequency modes. Our method can be classified as a static DFS (Dynamic Frequency Scaling) algorithm, since it is able to compute off line the optimal sequence of voltage/frequency 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 tasks, where deadlines are allowed to be less than or equal to periods (or minimum interarrival times).

—Any possible task activation pattern (periodic, sporadic, jitter, bursty,...) can be taken into account.

—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 and power consumption. The transition delay and energy overhead between any two modes is

taken into account.

—A more accurate task model, introduced by Seth et al. [Seth et al. 2003], 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 extensible 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.

The rest of the paper is organized as follows. Section 2 introduces the system model used throughout the paper. Section 3 presents a model for expressing the computational demand of the application to the processor, considering both EDF and FP scheduling. Section 4 presents the power management policy and describes the method for selecting the operating modes that minimize the energy consumption. In Section 5 and 6 we explain the design methodology also by using two clarifying examples. Finally, Section 7 states our conclusions and future work.

2. SYSTEM MODEL

2.1 Task model

We consider a set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ of n 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), and a relative deadline D_i. The activations of τ_i are modeled by the function $\mathsf{act}_i(t)$, which denotes the maximum number of activations in any interval long t. This task model is borrowed from the event stream task model [Gresser 1993; Richter and Ernst 2002]. We also assume that the deadline D_i is not greater than the minimum separation between two consecutive activations. To analyze the schedulability under EDF, we use $\mathsf{jobs}_i(t)$ to denote the number of τ_i jobs whose arrival time and deadline are in the interval $[0, t]$. If τ_i is a periodic task with period T_i, we have $\mathsf{act}_i(t) = \left\lceil \frac{t}{T_i} \right\rceil$ and $\mathsf{jobs}_i(t) = \max\left\{0, \left\lfloor \frac{t-D_i+T_i}{T_i} \right\rfloor\right\}$.

Tasks are fully preemptive and do not perform blocking operations. Note that intertask communication can still be performed using non-blocking mechanisms, such as Cyclic Asynchronous Buffers [Buttazzo 2004].

As observed by Seth et al. [Seth et al. 2003], 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 [Seth et al. 2003]:

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

where α is the processor speed in cycles per second (cyc/sec).

2.2 Energy model

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 modeled by well known polynomial relationships [Chandrakasan and Brodersen 1995; Hong et al. 1998]. 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) [Rabaey et al. 2002]. Recently, some work addressed the issue of scheduling a real-time application while reducing the leakage power as well [Quan et al. 2004].

These remarks suggest that the classical polynomial relationship [Rabaey et al. 2002] should be reconsidered. 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 (cyc/sec); p_k is the power consumed in mode k, measured in Watts.

In a recent work, AbouGhazaleh et al. [AbouGhazaleh et al. 2002] took into account in great detail the speed switching overhead, besides a more general policy for placing power management procedure calls in the application code. In their model, the speed switching overhead may be *software*, due to the computation of the new speed, and *hardware*, due to power management circuits.

Following their scheme, in this paper, the overhead is taken into account through a *matrix of time overheads* $\mathcal{O}$, where each element o_{ij}, $i \neq j$, is the time overhead required to switch from the operating mode Λ_i to Λ_j. Similarly we account for energy spent during mode transitions by introducing the *matrix of energy overheads* $\mathcal{E}$, where each element e_{ij}, $i \neq j$ denotes the energy spent when switching from mode Λ_i to mode Λ_j. Both matrices may be non-symmetric.

3. COMPUTING THE OPTIMAL SPEED

In this section we show the method to compute the minimum constant speed which allows all the deadlines of tasks in $\mathcal{T}$ to be met. We consider two major scheduling strategies: Earliest Deadline First (EDF) and Fixed Priority (FP) scheduling. The procedure explained in this section can be viewed as a function findOptSpeed, which takes the task set $\mathcal{T}$ and the scheduling algorithm as inputs and it returns the optimal speed α_{opt} as output.

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 [Baruah et al. 1990], 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 dSet is the set of all time instants where the test has to be performed.

It has been proved that the set dSet is the set of all the deadlines within the first busy period [Baruah et al. 1990; Ripoll et al. 1996]. Unfortunately, the length of the busy period depends on the speed. Hence we assume dSet to be equal to the entire set of all deadlines before the least common multiple of all the periods T_i (called *hyperperiod* in the literature). It is still under investigation 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 frequency α, 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 [Seth et al. 2003].

3.2 FP analysis

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

$$\forall i = 1, \ldots, n \quad \exists t \in \mathsf{tSet}_i \qquad C_i + \sum_{j=1}^{i-1} \mathsf{act}_j\,(t)\; C_j \leq t$$

where tSet_i is the set of scheduling points [Lehoczky et al. 1989; Bini 2004] 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 [Seth et al. 2003], we have

$$\forall i = 1, \ldots, n \quad \exists t \in \mathsf{tSet}_i \qquad c_i + \alpha\, m_i + \sum_{j=1}^{i-1} \mathsf{act}_j\,(t)\,(c_j + \alpha\, m_j) \leq \alpha\, t.$$

Hence, following the same steps which allowed us to derive Eq. (5) from Eq. (2) for an EDF scheduler, the optimal speed α_{opt} for the task set scheduled by FP is given by:

$$\alpha_{\text{opt}} = \max_{i=1,\ldots,n} \min_{t \in \mathsf{tSet}_i} \frac{c_i + \sum_{j=1}^{i-1} \mathsf{act}_i\,(t)\; c_j}{t - m_i - \sum_{j=1}^{i-1} \mathsf{act}_i\,(t)\; m_j}, \tag{7}$$

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

Equation (7) generalizes a previous result [Pillai and Shin 2001]. By setting $\mathsf{tSet}_i = \{D_i\}$ for all $i = 1, \ldots, n$ we can find the same speed computed by Pillai and Shin [Pillai and Shin 2001]. However tSet_i can be strictly larger than $\{D_i\}$, hence their result provides only a suboptimal solution.

Also the algorithm Sys-Clock, proposed by Saewong et al. [Saewong and Rajkumar 2003], produces exactly the same result of Equation (7), but it considers the larger set of schedulability points initially proposed by Lehoczky et al. [Lehoczky et al. 1989].

4. POWER MANAGEMENT

Once the ideal speed α_{opt} is 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 $\alpha_k = \alpha_{\text{opt}}$, we simply select it. Otherwise we have to properly manage the available processor operating modes to approximate the optimal speed.

4.1 The PWM scheme

The simplest solution is to select the least consuming speed higher than the optimal one α_{opt} among the available operating modes. This solution can consume a large amount of energy, although very simple to implement. Saewong et al. [Saewong and Rajkumar 2003] evaluated the cost of "rounding up" the processor speed.

Instead, we propose to switch between two operating modes, Λ_L and Λ_H, such that $\alpha_L < \alpha_{\text{opt}} < \alpha_H$, as suggested by Ishihara et al. [Ishihara and Yasuura 1998]. 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 [Tal and Person 1978]. 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. As it can be noticed in the figure, in the proposed scheme the processor runs for Q_H time units in the Λ_H mode and for Q_L in the Λ_L mode. The overheads o_{HL}, o_{LH} are included within the length of Q_L, Q_H respectively, so that the period of the scheme is $P = Q_L + Q_H$. Sometimes we will also use the frequency of the scheme $f = \frac{1}{P}$. In the figure we also highlight in gray the areas corresponding to the amount of cycles that are wasted due to the time overhead. We denote this amount by Δ_{LH} and it is equal to $\alpha_H o_{LH} + \alpha_L o_{HL}$. Finally $\lambda_L = \frac{Q_L}{P}$ denotes the

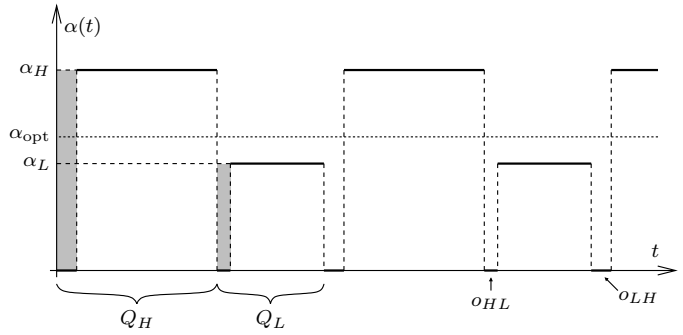

Fig. 1. An example of PWM-mode.

fraction of time during which the processor runs in Λ_L mode, and $\frac{Q_H}{P} = 1 - \lambda_L$ the fraction of time in the mode Λ_H.

4.2 Selecting Λ_L and Λ_H

Given the switching scheme, the first goal is to properly select the two operating modes Λ_L and Λ_H. For convex power-speed relationships, the speed pair (α_L, α_H) that minimizes the power consumption in the PWM-mode is given by the two speeds closest to α_{opt} [Ishihara and Yasuura 1998]. However the power-speed relationship may be different than the ideal polynomial function [Rabaey et al. 2002] and there may be modes with the same speed, but different power consumption (due to different voltage). For example, the Transmeta Crusoe processor has four different idle modes (Auto Halt, Quick Start, Deep Sleep, Extended Deep Sleep) which have decreasing power consumption although increasing overhead.

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:

$$\alpha_{\text{eff}} = \frac{\alpha_H (Q_H - o_{LH}) + \alpha_L (Q_L - o_{HL})}{Q_H + Q_L} = \lambda_L \alpha_L + (1 - \lambda_L)\alpha_H - \Delta_{LH} f. \quad (8)$$

Equation (8) is quite insightful, since it highlights the active contribution to the effective speed due to the two speeds α_L and α_H and the loss due to the presence of time overhead. As expected, the speed loss due the mode switching grows with the frequency of the scheme f and it becomes negligible for f approaching zero.

Following a similar approach it is also possible to express the average power consumption by the following simple expression:

$$p_{\text{eff}} = \frac{p_H (Q_H - o_{LH}) + p_L (Q_L - o_{HL})}{P} + \frac{e_{LH} + e_{HL}}{P} = \lambda_L p_L + (1 - \lambda_L) p_H + E_{\text{sw}} f \quad (9)$$

where E_{sw} denotes the energy wasted during the two mode switches in one period P:

$$E_{\text{sw}} = e_{LH} - p_H o_{LH} + e_{HL} - p_L o_{HL}. \quad (10)$$

A convenient way to illustrate the method to select 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. Let us assume the operating modes of a sample processor reported in Table I. For simplicity, in this example both time and energy overhead o_{ik} and e_{ik} do not depend on the previous mode Λ_i but only on the entered mode Λ_k.

k	1 (idle)	2	3	4	5	6
α_k [MHz]	0	5	30	40	50	80
p_k [mW]	0	20	50	50	200	500
$o_{ik} = o_{jk}$ [μsec]	1000	10	50	200	40	20
$e_{ik} = e_{jk}$ [μJ]	50	1	5	10	10	60

Table I. An example of processor operating modes.

These modes are represented in Figure 2 by black dots. For all the possible pairs (Λ_L, Λ_H) the dashed lines contain all the values $(\alpha_{\text{eff}}, p_{\text{eff}})$ achievable by varying $\lambda_L \in [0, 1]$.

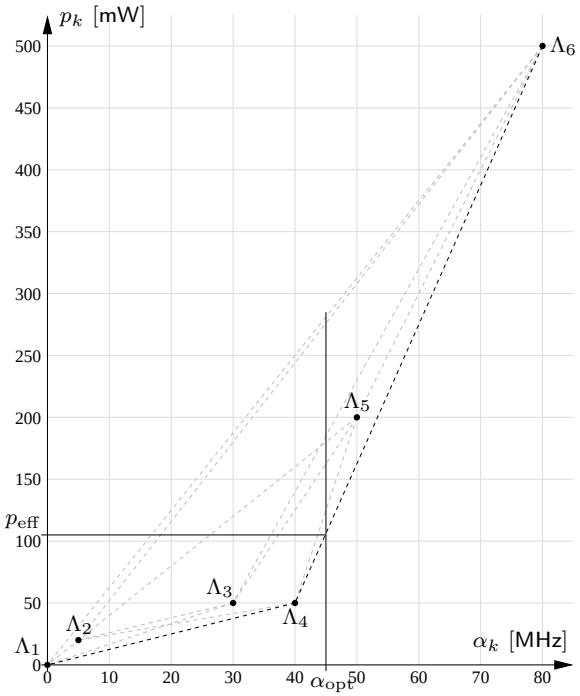

Fig. 2. The operating modes in the power/speed space.

Basically, the PWM scheme introduces additional "virtual" operating modes which are given by the *convex combination* of any two real modes. Clearly there may be more than one mode pair for reproducing a desired speed value α_{opt}. In the example depicted in the figure the speed of $\alpha_{\text{opt}} = 45$MHz can be reproduced in 8 different ways by selecting $\Lambda_L \in \{\Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4\}$ and $\Lambda_H \in \{\Lambda_5, \Lambda_6\}$. Among the possible pairs, the one that consumes least power is (Λ_4, Λ_6).

The selection procedure described until now neglects the delay and the energy overhead incurred at mode switches. As the period P decreases, the impact of the overheads becomes significant. Below some threshold period value we expect that it is more convenient to run continuously in the same mode rather than switching frequently between two modes. Let us evaluate the impact of the overhead in detail.

Suppose we want to emulate a given speed value α_{opt}. Then we can find the fraction of time λ_L to stay in the mode Λ_L by setting $\alpha_{\text{eff}} = \alpha_{\text{opt}}$, as follows

$$\begin{aligned} \alpha_{\text{opt}} &= \alpha_{\text{eff}} \\ \alpha_{\text{opt}} &= \lambda_L \alpha_L + (1 - \lambda_L)\alpha_H - \Delta_{LH} f \\ \lambda_L &= \frac{\alpha_H - \alpha_{\text{opt}} - \Delta_{LH} f}{\alpha_H - \alpha_L}. \end{aligned} \tag{11}$$

Hence the power consumed can be obtained by replacing this value of λ_L, which guarantees a speed equal to α_{opt}, in Eq. (9). We find

$$p_{\text{eff}} = \frac{\alpha_H - \alpha_{\text{opt}}}{\alpha_H - \alpha_L} p_L + \frac{\alpha_{\text{opt}} - \alpha_L}{\alpha_H - \alpha_L} p_H + f \left(\frac{p_H - p_L}{\alpha_H - \alpha_L} \Delta_{LH} + E_{\text{sw}} \right). \tag{12}$$

Equation (12) shows that the power p_{eff} consumed when guaranteeing the speed α_{opt} increases proportionally with the frequency f of the PWM scheme. The proportionality coefficient increases for large overheads. This confirms the intuition that the impact of the overhead is greater for small periods P. Moreover, it follows that the least consuming pair (Λ_L, Λ_H) may vary with the frequency f depending on the energy overheads and the energy spent in the mode switch.

To clarify the effect of the overheads we propose the same example whose data is reported in Table I. In Figure 3 we plot the power consumption p_{eff} with respect to the scheme frequency f. When f approaches to 0 then the least consuming pair is (Λ_4, Λ_6), as we showed in Figure 2 as well. However, the overheads of the modes

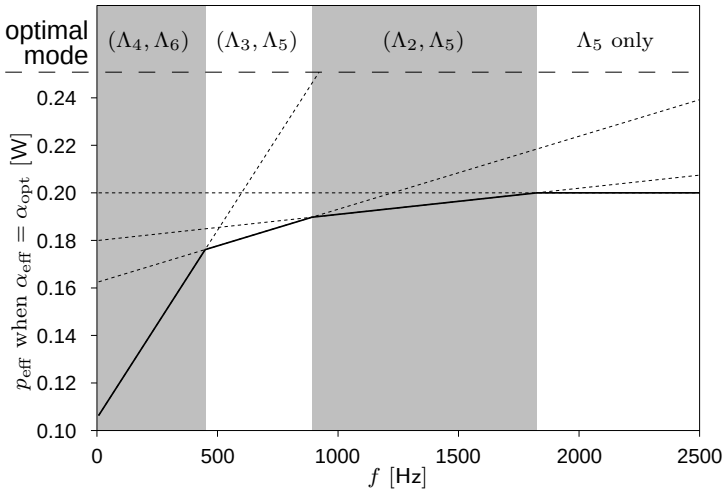

Fig. 3. The least consuming pair as a function of f.

Λ_4 and Λ_6 are relevant. Hence, it follows that the rate of increase of p_{eff} with respect to f is high (see Eq. (12)). At frequency $f = 442\text{Hz}$ it happens that the mode pair (Λ_3, Λ_5), which has smaller overheads, becomes more convenient than (Λ_4, Λ_6). Then at frequency $f = 886\text{Hz}$ the pair (Λ_2, Λ_5) becomes more efficient. Finally, when $f \geq 1818\text{Hz}$ then the PWM scheme is no longer convenient since any mode pair will consume more than simply running constantly in mode Λ_5, which can indeed guarantee the task deadlines since $\alpha_5 = 50 \geq 45 = \alpha_{\text{opt}}$. The presence of a critical frequency f (1818Hz in the example) above which the PWM scheme is no longer convenient is a consequence of the non-zero overhead. This phenomenon also appears when assigning a speed to the tasks [Irani et al. 2003; Aydin et al. 2006].

The procedure described above is implemented by the algorithm findPairs that returns the set of candidate pairs, given the desired speed α_{opt} and the power consumption $p_{\max}$ of the least consuming mode which can feasibly schedule the task set. When the algorithm is invoked by candidatePairs = findPairs$(\alpha_{\text{opt}}, p_{\max})$ it returns a vectors of records candidatePairs$[i]$ whose fields are:

(1) modeL, the mode Λ_L of the pair;

(2) modeH, the mode Λ_H of the pair;

(3) lowFreq, the lower bound of the frequency interval where the pair is convenient;

(4) uppFreq, the upper bound of the frequency interval where the pair is convenient;

(5) minPow the minimum possible power consumption by the pair (i.e. when $f =$ lowFreq).

In Table II we report the result of the invocation candidatePairs$(45 \cdot 10^6, 0.2)$, showed in the previous example (see also Fig. 3). This procedure will be used in

i	modeL	modeH	lowFreq[Hz]	uppFreq[Hz]	minPow[mW]
1	4	6	0	442	106
2	3	5	442	886	176
3	2	5	886	1818	190

Table II. Results returned by findPairs.

the algorithm for the complete design presented in Section 6.

From the discussion developed until now it seems that the solution that saves more energy is $f = 0$, which means to have an *arbitrarily large* period P. However, if we adopt this solution the scheme will also experience arbitrary large intervals where the processor is running continuously in mode Λ_L. During this period we are going to miss some deadlines since $\alpha_L < \alpha_{\text{opt}}$. This problem is caused by a subtle implicit assumption we made: we assumed that the PWM scheme provides a constant speed α_{eff}, whereas the speed α_{eff} is only approximated by the switching between two speeds α_L and α_H.

In the next section we will overcome this assumption by finely modeling the amount of cycles provided by the PWM scheme.

4.3 Processor supply function

To exactly model the cycles provided by the management scheme, we will follow the demand/supply approach [Almeida et al. 2002; Feng and Mok 2002; Lipari and Bini 2003; Shin and Lee 2003], which has been successfully proposed to model a hierarchical scheduler. The key idea is that the time demanded by the application must never exceed the time supplied to it, otherwise some deadlines may be missed.

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 follows:

$$Z(t) = \min_{t_0} \int_{t_0}^{t_0+t} \alpha(x)\,\mathrm{d}x. \tag{13}$$

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

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 the property that:

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

allows the definition of $Z(t)$ for all t.

From Eq. (13), it follows that $Z(t) = 0$ for all t that are smaller than of equal to the longest interval during which no cycle is provided [Almeida et al. 2002; Feng and Mok 2002; Lipari and Bini 2003; Shin and Lee 2003]. Due to the speed switching overhead, the longest time where no processor cycle is available is $o_{\max} = \max\{o_{LH}, o_{HL}\}$. For this reason $Z(t) = 0$ for $t \in [0, o_{\max})$. Then, for $t > o_{\max}$ some cycles are available. In the worst case, the available cycles increase with the rate α_L. This amount of processor cycles is provided for $Q_L - o_{HL}$. Then the second (shorter) switching overhead occurs $o_{\min} = \min\{o_{LH}, o_{HL}\}$. 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:

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

The supply function $Z(t)$ for the PWM-mode is also illustrated in Figure 4. We remark that the supply function depends also on the parameters α_L, Q_L, o_{HL}, α_H, Q_H, and o_{LH}, although not explicitly stated in the notation $Z(t)$ for simplicity.

5. COMPUTING Q_L AND Q_H

Given the pair (Λ_L, Λ_H), in this section we compute the exact lengths Q_L and Q_H, starting from the exact expression of the processor supply function $Z(t)$. Unfortunately, in the previous section we showed that the least consuming mode pair depends on the period of the scheme $P = Q_L + Q_H$. This *circular dependency* between "the selection of the mode pair (Λ_L, Λ_H)" and "the computation of (Q_L, Q_H)" will be solved by a branch-and-bound algorithm later in Section 6.

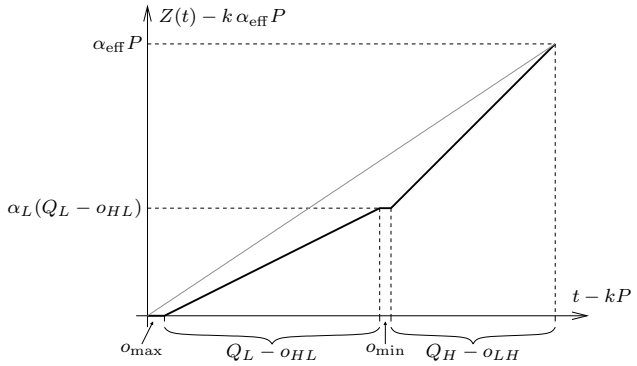

Fig. 4. The supply function $Z(t)$.

Since we do not know when the non-scalable computation time will occur during task execution (that is, when running at α_L or at α_H), we must safely assume that all the non-scalable portions of tasks are executed at speed α_H, so that the worst-case execution cycles are maximized. Hence we set

$$\forall i = 1, \ldots, n \qquad c_i \leftarrow c_i + m_i \alpha_H, \quad m_i \leftarrow 0 \tag{16}$$

However we remind that, as shown in [Seth et al. 2003], the impact of m_i vs. c_i is minor, meaning that this overestimation is reasonably tight.

When adopting the EDF scheduling algorithm, the exact schedulability condition of the task set $\mathcal{T}$ on the PWM scheme becomes

$$\forall t \in \mathsf{dSet} \qquad \sum_{i=1}^{n} \mathsf{jobs}_i(t)\, C_i \leq Z(t) \tag{17}$$

meaning that the demanded cycles must not exceed the provided 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:

$$\forall i = 1, \ldots, n \quad \exists t \in \mathsf{tSet}_i \quad C_i + \sum_{j=1}^{i-1} \mathsf{act}_i\,(t)\; C_j \leq Z(t).$$

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

$$W(t) \leq Z(t) \tag{18}$$

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. Table III summarizes the t and W for the two scheduling algorithms.

We now proceed by computing the optimal pair $(Q_L^{\text{opt}}, Q_H^{\text{opt}})$ that minimizes energy consumption. 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*

Alg.	Instants t	Demand W
EDF	$\forall t \in \mathsf{dSet}$	$\sum_{i=1}^{n} \mathsf{jobs}_i(t) C_i$
FP	$\forall i = 1, \ldots, n \quad \exists t \in \mathsf{tSet}_i$	$C_i + \sum_{j=1}^{i-1} \mathsf{act}_j\,(t)\, C_j$

Table III. The demand/supply scheme.

that

$$W(t) \leq Z(t), \tag{19}$$

where $Z(t)$ is the cycle supply function defined in Eq. (15), which depends on (Q_L, Q_H). Formally:

$$\mathbb{Q}(t, W) = \{(Q_L, Q_H) : W \leq Z(t)\}. \tag{20}$$

The advantage of defining $\mathbb{Q}(t, W)$ is that the feasible pairs (Q_L, Q_H) are easily expressed as a combination of basic Q-domains. In fact, from the equivalence

$$W \leq Z(t) \quad \Leftrightarrow \quad (Q_L, Q_H) \in \mathbb{Q}(t, W)$$

Equation (17) allows to prove that a task set scheduled by EDF will never miss a deadline in the PWM scheme **if and only if**:

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

$$(Q_L, Q_H) \in \bigcap_{t \in \mathsf{dSet}} \mathbb{Q}\left(t, \sum_{i=1}^{n} \mathsf{jobs}_i(t)\, C_i\right). \tag{21}$$

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

$$(Q_L, Q_H) \in \bigcap_{i=1,\ldots,n} \bigcup_{t \in \mathsf{tSet}_i} \mathbb{Q}\left(t, C_i + \sum_{j=1}^{i-1} \mathsf{act}_j\,(t)\; C_j\right). \tag{22}$$

Now we focus on finding the basic Q-domain $\mathbb{Q}(t, W)$ for generic t and W.

The analytical expression of $\mathbb{Q}(t, W)$ can be found by inverting Equation (18), assuming $Z(t)$ as in Equation (15), thus expressing (Q_L, Q_H) as a function of W, t, α_L, o_{HL}, α_H and o_{LH}. First we set $k = \left\lfloor \frac{t}{P} \right\rfloor$. Using the property in Equation (14), the condition $W \leq Z(t)$ becomes:

$$Z(t - kP) \; + k\, \alpha_{\text{eff}} P \geq W \tag{23}$$

As we can see from the definition of $Z(t)$ of Equation (15), four cases need to be considered:

(1) when $t - kP \in [0, o_{\max})$:

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

$$\alpha_L Q_L + \alpha_H Q_H \geq \frac{W}{k} + \Delta_{LH}. \tag{24}$$

(2) when $t - kP \in [o_{\max}, o_{\max} + Q_L - o_{HL})$:

$$\begin{aligned} &\alpha_L(t - kP - o_{\max}) + k\,\alpha_{\text{eff}} P \geq W \quad \Leftrightarrow \\ &Q_H \geq \frac{W + k\Delta_{LH} - \alpha_L(t - o_{\max})}{k(\alpha_H - \alpha_L)}. \end{aligned} \tag{25}$$

(3) when $t - kP \in [o_{\max} + Q_L - o_{HL}, Q_L + o_{LH})$:

$$\begin{aligned} &\alpha_L(Q_L - o_{HL}) + k\,\alpha_{\text{eff}} P \geq W \quad \Leftrightarrow \\ &(k+1)\alpha_L Q_L + k\alpha_H Q_H \geq W + \alpha_L o_{HL} + k\Delta_{LH}. \end{aligned} \tag{26}$$

(4) when $t - kP \in [Q_L + o_{LH}, P)$:

$$\begin{aligned} &\alpha_H(t - (k+1)P) + (k+1)\,\alpha_{\text{eff}} P \geq W \quad \Leftrightarrow \\ &Q_L \leq \frac{\alpha_H t - (k+1)\Delta_{LH} - W}{(k+1)(\alpha_H - \alpha_L)}. \end{aligned} \tag{27}$$

Equations (24), (25), (26) and (27) allow to construct the region $\mathbb{Q}(t, W)$ for any value of t and W. Then, by combining these regions as described in Equations (21) and (22) we can finally find the feasible pairs (Q_L, Q_H), which allows to construct a PWM-scheme that guarantees all the deadlines.

The construction of the region of the feasible time quanta (Q_L, Q_H) is implemented by the procedure buildQDomain that takes as inputs the modes Λ_L and Λ_H, the task set $\mathcal{T}$, and the scheduling algorithm, and it returns the region of the feasible (Q_L, Q_H). This procedure is used in Section 6 to implement the complete algorithm for the selection of the modes Λ_L, Λ_H, and the Q_L, Q_H which minimize the energy consumption.

5.1 Selecting the least consuming (Q_L, Q_H)

Once the region of the feasible values of (Q_L, Q_H) is described, we select the pair (Q_L, Q_H) which minimizes the power consumption expressed in Eq. (9). It is quite interesting to study how the power consumption p_{eff} varies as a function of (Q_L, Q_H). From Eq. (9) we find

$$\begin{aligned} p_{\text{eff}} &= \frac{Q_L}{Q_L + Q_H} p_L + \frac{Q_H}{Q_L + Q_H} p_H + \frac{1}{Q_L + Q_H} E_{\text{sw}} \\ (Q_L + Q_H) p_{\text{eff}} &= Q_L p_L + Q_H p_H + E_{\text{sw}} \\ (p_{\text{eff}} - p_L) Q_L &- (p_H - p_{\text{eff}}) Q_H = E_{\text{sw}} \end{aligned} \tag{28}$$

meaning that the level curves of a constant p_{eff} are lines in the plane (Q_L, Q_H) with a unique point of intersection at $\left(\frac{E_{\text{sw}}}{p_H - p_L}, -\frac{E_{\text{sw}}}{p_H - p_L}\right)$. The algorithm for finding the least consuming pair (Q_L, Q_H) is implemented by the procedure minNrgQ, which will be used also in Section 6.

5.2 Example of applicability

To clarify the method adopted for building the space of the feasible (Q_L, Q_H) and selecting the pairs which consumes the minimum amount of energy, we propose a simple example with only one task. Later in Section 6.1 we show an example

where the selection of the modes and the time quanta is performed in an integrated framework.

Let us suppose that we have only one task whose data are:

—scalable execution cycles $c_1 = 240 \cdot 10^3 \mathsf{cyc}$;
—non-scalable computation time $m_1 = 400\,\mu\mathsf{sec}$;
—period and deadline $T_1 = D_1 = 9.6\,\mathsf{msec}$.

From Equation (6) we find that:

$$\alpha_{\text{opt}} = \frac{c_1/T_1}{1 - m_1/T_1} = 26.087\mathsf{MHz}. \tag{29}$$

We suppose this speed is not available and the two closest available operating modes are $\Lambda_L = (20\mathsf{MHz}, 200\mathsf{mW})$, $\Lambda_H = (40\mathsf{MHz}, 800\mathsf{mW})$ and the overheads $o_{HL} = 160\mu\mathsf{sec}$, $o_{LH} = 240\mu\mathsf{sec}$, $e_{LH} = e_{HL} = 220\mu\mathsf{J}$. 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 safely assume that the non-scalable portion of task executes at speed α_H, so that the worst-case execution cycles are maximized. Hence the required cycles are

$$C_1 = c_1 + m_1\alpha_H = 256 \cdot 10^3 \mathsf{cyc}. \tag{30}$$

as indicated in Eq. (16).

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:

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

Notice that this condition ensures the task schedulability both under FP and EDF, because the two algorithms coincide when only one task is in the system. The resulting set $\mathbb{Q}(T_1, C_1)$ is shown in Figure 5 in white.

For each value of k (remember that $k = \left\lfloor \frac{t}{P} \right\rfloor = \left\lfloor \frac{T_1}{Q_L + Q_H} \right\rfloor$) the domain boundary is composed by the four segments of Equations (24)–(27), since $Z(t)$ is defined on four different intervals.

In Figure 5 we also represent by dotted lines the level curves of a constant power consumption p_{eff}, defined by Equation (28). For each line the power saving of p_{eff} with respect to p_H is reported. These lines all intersect at the point $(0.36, -0.36)$ since $\frac{E_{\text{sw}}}{p_H - p_L} = \frac{216 \cdot 10^{-6}}{600 \cdot 10^{-3}} = 0.36\mathsf{msec}$. The admissible pair that achieves the greatest power saving is at the vertex with $Q_L = 5.76\,\mathsf{msec}$ and $Q_H = 3.84\,\mathsf{msec}$, where the power consumed is $p_{\text{eff}} = 462.5\mathsf{mW}$, which allows to save the 42.2% of energy with respect to running continuously in mode Λ_H.

6. THE COMPLETE DESIGN ALGORITHM

In Section 4.2 we showed the method to select the least consuming pair (Λ_L, Λ_H) for a given period of the scheme $P = Q_L + Q_H$. Given the mode pair (Λ_L, Λ_H), in Section 5 we described the region of all the feasible values of (Q_L, Q_H) which guarantee the deadline constraints of the tasks in $\mathcal{T}$. We also described (in Section 5.1) how to select the least consuming time quanta in the feasible region. Unfortunately

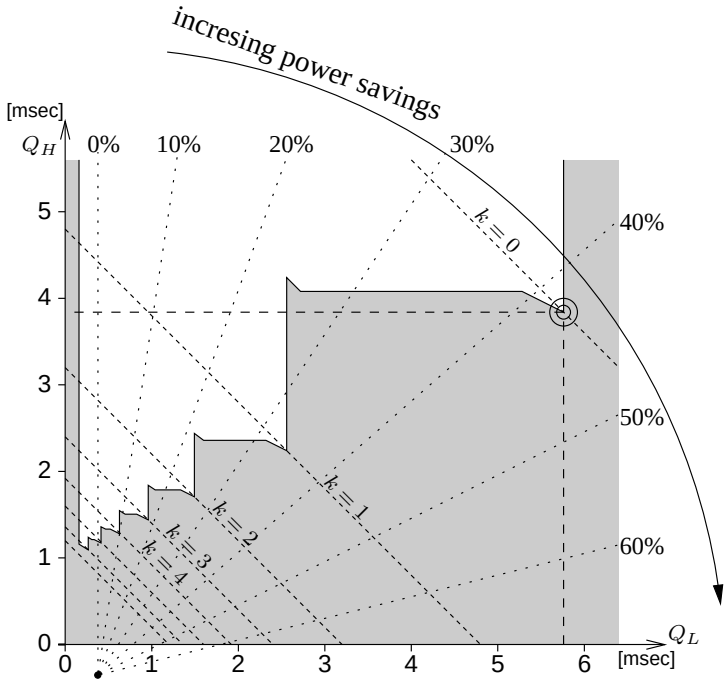

Fig. 5. Schedulability region (white area) in the Q-Space.

the mode selection requires the knowledge of the scheme period $Q_L + Q_H$, whereas the time quanta selection requires to know the modes Λ_L and Λ_H in advance.

Such a *circular dependency* between "the selection of the mode pair (Λ_L, Λ_H)" and "the computation of (Q_L, Q_H)" is solved by the branch-and-bound algorithm reported in Figure 6.

The algorithm takes as inputs the task set $\mathcal{T}$ and the scheduling algorithm alg (currently we developed the solution only for EDF and FP as scheduling algorithms). First we compute the optimal constant speed α_{opt}, which can guarantee the deadlines, and minimum power consumption of the available modes which can run not slower than α_{opt} (lines 2–3). The solution is initialized with the zero solution that is running constantly in the same mode. At line 6 we store in modeP all the possible mode pairs that can reproduce the speed α_{opt} without exceeding a power consumption $p_{\max}$. Then we loop (line 7) on all the candidate pairs that can possibly provide a better solution than the current (line 8).

For each candidate pair (Λ_L, Λ_H), first we modify the worst-case execution cycles assuming that the non-scalable part occur when running at speed α_H (lines 13–14). Then we build the region of the feasible time quanta (line 15) and we intersect it with the constraints deriving from the selection of current mode pair (Λ_L, Λ_H). In fact, at this stage we must remember that the current mode pair is the least

```
1: procedure optimPWMscheme(T, alg)                          ▷ task set, scheduling alg.
2:     α_opt ← findOptSpeed(T, alg)                          ▷ computation of α_opt in Sec. 3
3:     p_max ← min{p_k : α_k ≥ α_opt}                        ▷ min power of a feasible speed
4:     curPair ← 0                          ▷ the "pair" 0 means that a constant mode is used
5:     curPow ← p_max
6:     modeP ← findPairs(α_opt, p_max)                       ▷ see Sec. 4.2
7:     for j = 1, ..., length(modeP) do                      ▷ loop on the pairs
8:         if modeP[j].minPow ≥ curPow then
9:             do nothing                  ▷ no better solution is possible. Skip the pair
10:        else
11:            L ← modeP[j].modeL                            ▷ current low mode
12:            H ← modeP[j].modeH                            ▷ current high mode
13:            ∀i   T'.c_i ← T.c_i + T.m_i α_H               ▷ see Eq. (16)
14:            ∀i   T'.m_i ← 0
15:            domQ ← buildQDomain(Λ_L, Λ_H, T', alg)        ▷ see Eq. (21), (22)
16:                                          ▷ intersect with the period constraint
17:            domQ ← domQ ∩ {modeP[j].lowFreq ≤ 1/P ≤ modeP[j].uppFreq}
18:            (Q_L, Q_H) ← minNrgQ(domQ)                    ▷ see Sec. 5.1
19:            p_eff = (Q_L p_L + Q_H p_H + E_sw) / (Q_L + Q_H)   ▷ from Eq. (9)
20:            if p_eff < curPow then                        ▷ the current pair is better
21:                curPair ← j
22:                curPow = p_eff
23:                store the values (Q_L, Q_H)
24:            end if
25:        end if
26:    end for
27: end procedure
```

Fig. 6. Algorithm for computing the least consuming (Q_L, Q_H).

We must say that the algorithm described in Figure 6 is indeed very complex. The most complex activity is the invocation of buildQDomain that requires to compose as many basic Q-domains as the number of points in dSet, if EDF is used, or as many as in tSet, if FP is used. Unfortunately the cardinality of these set is exponential in the number of tasks. This makes the routine impractical for on-line usage. However the main purpose of the procedure is to design off-line a PWM scheme that is capable to minimize the energy.

In the next section we propose a simple example to better show the applicability of the algorithm.

6.1 Example of applicability

In this example we consider a set of three periodic tasks with periods $T_1 = 3$msec, $T_2 = 8$msec, and $T_3 = 20$msec. The worst-case execution cycles are $c_1 = c_2 = 10^5$ and $c_3 = 2 \cdot 10^5$, where the non-scalable parts m_i are assumed equal to zero for simplicity. Deadlines are set equal to periods. The available operating modes are those reported in Table I.

We start assuming an EDF scheduler. Since deadlines are equal to periods then the optimal speed α_{opt} is equal to $\sum_i \frac{c_i}{T_i} = 55.83$MHz. The only mode that can provide this speed is Λ_6, whose power consumption is $p_{\max} = p_6 = 500$mW.

The mode pairs that can reproduce the speed α_{opt} are reported in Table IV. For convenience, the table reports the maximum and minimum periods $P_{\max}$, $P_{\min}$

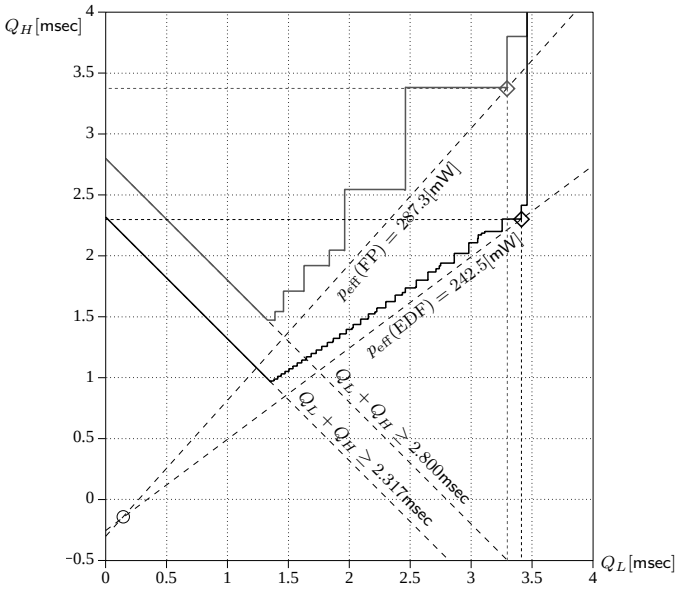

Fig. 7. The feasible (Q_L, Q_H), when $\Lambda_L = \Lambda_4$ and $\Lambda_H = \Lambda_6$.

instead of their respective inverse lowFreq and uppFreq.

modeL	modeH	$P_{\max}$[msec]	$P_{\min}$[msec]	minPow[mW]
4	6	$+\infty$	2.317	228
5	6	2.317	0.364	296

Table IV. Results of findPairs, when $\alpha_{\text{opt}} = 55.83$MHz.

Now we start considering the first of the two possible solutions. The first candidate mode pair (Λ_4, Λ_6) is the least consuming only when $Q_L + Q_H \geq 2.317$msec. In Figure 7 we draw in black the boundary of the feasible (Q_L, Q_H) values, intersected with the lower bound on the period $P = Q_L + Q_H$.

As explained in Section 5.1, the optimum occurs at the intersection between the line with the minimum speed and the feasible values (Q_L, Q_H). In the EDF case, we find $Q_L = 3.412$msec and $Q_H = 2.302$msec, achieving a power consumption $p_{\text{eff}} = 242.5$mW.

If we consider the second candidate pair (Λ_5, Λ_6) (see Table IV) we realize that the current solution allows consuming less than any possible solution running according to the second pair. Hence the solution running for 3.412msec in mode Λ_4 and for 2.302msec in Λ_6 is the best solution for EDF.

It is quite insightful to find the optimal solution also when a FP scheduler is adopted. From Eq. (7) it follows that the optimal speed is $\alpha_{\text{opt}} = 60$MHz. Then the resulting candidate mode pairs are reported in Table V.

In Figure 7 we overlap in gray the feasible time quanta (Q_L, Q_H) when the FP scheduler is adopted. The optimal solution found is $Q_L = 3.294$msec and $Q_H = 3.374$msec, which consumes $p_{\text{eff}} = 287.3$mW. It is again not necessary to

modeL	modeH	$P_{\max}$[msec]	$P_{\min}$[msec]	minPow[mW]
4	6	$+\infty$	2.800	275
5	6	2.800	0.440	331

Table V. Results of findPairs, when $\alpha_{\text{opt}} = 60$MHz.

evaluate any solution with the other mode (Λ_5, Λ_6), because the power consumed by this solution is less than the minimum power consumption achievable in this second pair.

7. 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, transition delay, and energy overhead. The proposed approach allows the user to compute the optimal sequence of voltage/speed changes that minimizes the 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 [Almeida et al. 2002; Feng and Mok 2002; Lipari and Bini 2003; Shin and Lee 2003], 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.

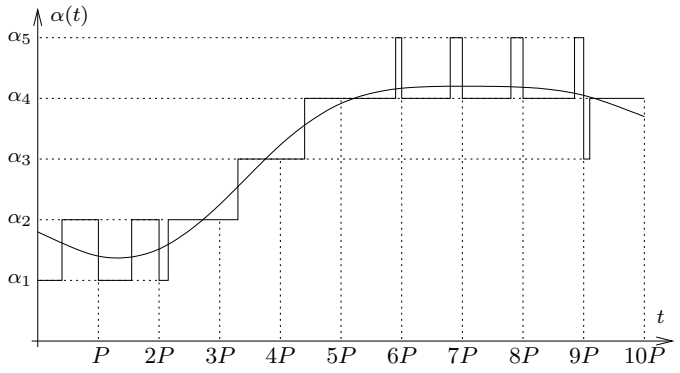

Fig. 8. Applying the PWM-mode to a dynamic scheme.

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. As depicted in Figure 8, the idea is to modulate the time-variant speed $\alpha(t)$ by switching periodically between the available operating modes.

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

REFERENCES

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

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

Aydin, H., Devadas, V., and Zhu, D. 2006. System-level energy management for periodic real-time tasks. In *Proceedings of the* 27^{th} *IEEE International Real-Time Systems Symposium*. Rio de Janeiro, Brazil, 313–322.

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

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

Bini, E. 2004. The design domain of real-time system. Ph.D. thesis, Scuola Superiore Sant'Anna, Pisa, Italy. available at `http://retis.sssup.it/~bini/thesis/`.

Bini, E., Buttazzo, G., and Lipari, G. 2005. Speed modulation in energy-aware real-time systems. In *Proceedings of the* 17^{th} *Euromicro Conference on Real-Time Systems*. Palma de Mallorca, Spain, 3–10.

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

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

Feng, X. and Mok, A. K. 2002. A model of hierarchical real-time virtual resources. In *Proceedings of the* 23^{rd} *IEEE Real-Time Systems Symposium*. Austin, TX, U.S.A., 26–35.

Gresser, K. 1993. An event model for deadline verification of hard real-time systems. In *Proceedings of the* 5^{th} *Euromicro Workshop on Real-Time Systems*. Oulu, Finland, 118–123.

Hong, I., Qu, G., Potkonjak, M., and Srivastava, M. B. 1998. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proceedings of the* 19^{th} *IEEE Real-Time Systems Symposium*. Madrid, Spain, 178–187.

Irani, S., Shukla, S., and Gupta, R. 2003. Algorithms for power savings. In *Proceedings of the* 14^{th} *annual ACM-SIAM symposium on Discrete algorithms*. Baltimore, MD, 37–46.

Ishihara, T. and Yasuura, H. 1998. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*. Monterey, CA U.S.A., 197–202.

Jejurikar, R. and Gupta, R. 2004. Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *Proceedings of the International Symposium on Low Power Electronics and Design*. Newport Beach, CA, USA, 78–81.

Lee, S. and Sakurai, T. 2000. Run-time voltage hopping for low-power real-time systems. In *Proceedings of the* 37^{th} *Design Automation Conference*. Los Angeles, CA U.S.A., 806–809.

Lehoczky, J. P., Sha, L., and Ding, Y. 1989. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the* 10^{th} *IEEE Real-Time Systems Symposium*. Santa Monica (CA), U.S.A., 166–171.

Lipari, G. and Bini, E. 2003. Resource partitioning among real-time applications. In *Proceedings of the* 15^{th} *Euromicro Conference on Real-Time Systems*. Porto, Portugal, 151–158.

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

Liu, Y. and Mok, A. K. 2003. An integrated approach for applying dynamic voltage scaling to hard real-time systems. In *Proceedings of the* 9^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium*. Washington, DC U.S.A., 116–123.

Mochocki, B., Hu, X. S., and Quan, G. 2002. A realistic variable voltage scheduling model for real-time applications. In *Proceedings of the International Conference on Computer Aided Design*. San José, CA U.S.A., 726–731.

Pillai, P. and Shin, K. G. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the* 18^{th} *ACM Symposium on Operating System Principles*. Banff, Canada, 89–102.

Qadi, A., Goddard, S., and Farritor, S. 2003. A dynamic voltage scaling algorithm for sporadic tasks. In *Proceedings of the* 24^{th} *Real-Time Systems Symposium*. Cancun, Mexico, 52–62.

Quan, G., Niu, L., Hu, X. S., and Mochocki, B. 2004. Fixed priority based real-time scheduling for reducing energy on variable voltage processors. In *Proceedings of the* 25^{th} *IEEE Real-Time Systems Symposium*. Lisbon, Portugal, 309–318.

Rabaey, J. M., Chandrakasan, A., and Nikolic, B. 2002. *Digital Integrated Circuits*, second ed. Prentice Hall. ISBN: 0-13-090996-3.

Richter, K. and Ernst, R. 2002. Event model interfaces for heterogeneous system analysis. In *Design, Automation and Test in Europe (DATE)*. Paris, France, 506–513.

Ripoll, I., Crespo, A., and Mok, A. K. 1996. Improvement in feasibility testing for real-time tasks. *Real-Time Systems 11,* 1, 19–39.

Saewong, S. and Rajkumar, R. 2003. Pratical voltage-scaling for fixed-priority RT-systems. In *Proceedings of the* 9^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium*. Washington (DC), U.S.A., 106–114.

Scordino, C. and Lipari, G. 2006. A resource reservation algorithm for power-aware scheduling of periodic and aperiodic real-time tasks. *IEEE Transactions on Computers 55,* 12 (Dec.), 1509–1522.

Seth, K., Anantaraman, A., Mueller, F., and Rotenberg, E. 2003. FAST: Frequency-aware static timing analysis. In *Proceedings of the* 24^{th} *IEEE Real-Time Systems Symposium*. Cancun, Mexico, 40–51.

Shin, I. and Lee, I. 2003. Periodic resource model for compositional real-time guarantees. In *Proceedings of the* 24^{th} *Real-Time Systems Symposium*. Cancun, Mexico, 2–13.

Tal, J. and Person, E. K. 1978. Pulsewidth modulated amplifier for dc servo system. *DC motor and control systems*.

Yao, F.,