On Energy Reduction for Hard Real-Time Tasks with Stochastic Execution Times

Flavius Gruian
Department of Computer Science, Lund University, Box 118,
S-221 00 Lund, Sweden
<Flavius.Gruian@cs.lth.se>

Abstract — This paper addresses energy reduction in hard real-time systems, containing independent tasks running on dynamic voltage supply processors. Since tasks with probabilistic run times are more realistic, we focus on systems containing such tasks. We show that both off-line and on-line scheduling may contribute in reducing the energy consumption, if the probabilistic behavior is taken into consideration. Stochastic data can be used in deriving schedules both for task sets and individual tasks. We also present a task-level scheduling method which reduces the average energy consumption independent of the other tasks in the system. The experimental results, although in an incipient phase, are promising.

I. Introduction

Low energy consumption is today an important design requirement for digital systems, with impact on operating time, on system cost, and, of no lesser importance, on the environment. Reducing power and energy dissipation in digital systems has long been addressed by several research groups. With the advent of dynamic voltage supply (DVS) processors [6,20], highly flexible systems can be designed, while still taking advantage of supply voltage scaling to reduce the energy consumption. Since the supply voltage has a direct impact on processor speed, classic task scheduling and supply voltage selection have to be addressed together. Scheduling offers thus yet another level of possibilities for achieving energy/power efficient systems, especially when the system architecture is fixed or the system exhibits a very dynamic behavior. For such dynamic systems, various power management techniques exist and are reviewed for example in [4,5]. Yet, these mainly target soft real-time systems, where deadlines can be missed if the Quality of Service is kept. Several scheduling techniques for soft real-time tasks, running on DVS processors have already been described [7,8,9]. Energy reductions can be achieved even in hard real-time systems, where no deadline can be missed, as shown in [10,11,12,13,14]. In this paper, we focus on hard real-time scheduling techniques, where all deadlines have to be met.

Task level voltage scheduling decisions can reduce even further the energy consumption. Some of these intra-task scheduling methods use several re-scheduling points inside a task, and are usually compiler assisted [16,17,18]. Alternatively, fixing the schedule before the task starts executing as in [11,14,19] eliminates the internal scheduling overhead, but with possible affects on energy reduction. Statistics can be used to take full advantage of the dynamic behavior of the system, both at task level [18] and at task-set level [10]. We show the importance of employing stochastic data in deriving efficient voltage schedules at both task and task-set levels.

The rest of the paper is organized as follows. Section II presents the scheduling alternatives taken at task-set level, for three cases, increasing in generality. A task-level scheduling method for low average energy is described in section III. We give examples and some experimental results throughout the paper, in specific sub-sections, hoping to make our point even clearer. Our conclusions and future work make the subject of section IV.

II. Task-Set Level Scheduling Decisions

In this paper we address only independent tasks running on a single dynamic supply voltage processor. Each of the N tasks in the task set is defined by a 4-tuple $\tau_i = (X_i, C_i, T_i, D_i)$ composed of the execution pattern, worst case execution time, period, and deadline for task τ_i. The execution pattern X_i is a random variable describing the number of clock cycles required to execute the task. We will denote by $\overline{X_i}$ the expected value of the random variable X_i. At the fastest processor speed (clock frequency), the maximal value for X_i is C_i.

The focus of this discussion is on the influence of considering tasks with probabilistic execution pattern on the task-set level scheduling strategies. The main goal of all the strategies presented here is reducing the energy consumption. We show that scheduling policies considering only the tasks WCET can be improved when stochastic data is used. We compute therefore the following three measures. The worst case execution speed, s^{WCE}, which does not consider the stochastic behavior of the tasks. The expected ideal speed, s^{ideal}, which assumes that all tasks will always run at their expected time and can take full advantage of this knowledge. And, finally, the average real speed, s, which is a estimate of the realistic case behavior, when the actual execution times are not known a priori. We can use these three measures to compute the energy consumptions for the worst, ideal, and average cases. The energy consumption for a given time interval $[t0, t1]$ is defined as the integral of power over time. Assuming a quadratic dependency between power and speed, as in [10], we obtain the energy as:

$$E = \int_{t0}^{t1} P(s(t))dt \cong \int_{t0}^{t1} (s(t))^2 dt \qquad (1)$$

What remains to be done is to find out the processor speeds in every time moment.

Because of the dynamic behavior of the system, we need to address not only off-line scheduling methods, but also on-line re-scheduling policies. Depending on the generality of the problem, the complexity of the off-line and on-line policies varies.

We address three cases, with increasing generality. First we consider task sets with tasks having the same period and deadline. Next we analyze tasks having the same period, but different deadlines. Finally, we consider task sets which have both different periods and possibly different deadlines.

A. Unique Period, Unique Deadline

This is the simplest possible case, where the set of tasks has to finish before a certain deadline, in no particular order. Formally we consider $T_i = D_i = T_j = D_j = A, \forall(\tau_i, \tau_j)$. When the actual execution time of each task is known beforehand, the optimal schedule from the energy point of view is given by uniformly stretching the tasks to exactly meet the deadline [14]. The worst case processor speed is then computed as:

$$s^{WCE} = \left(\sum_{1 \le i \le N} C_i\right) / A \qquad (2)$$

while the ideal expected speed:

$$s^{ideal} = \left(\sum_{1 \le i \le N} \overline{X}_i\right) / A . \qquad (3)$$

Using only off-line decisions, one can always guarantee the deadlines only by using the worst case speed. An on-line re-scheduling algorithm may be used every time a task completes its execution. In each of these scheduling points, a new expected optimal processor speed can be computed as:

$$s_j = \left(\sum_{j \le i \le N} C_i\right) / \left(A - \sum_{1 \le k < j} (\overline{X}_k / s_k)\right) \qquad (4)$$

Note that the execution order of the tasks in this case becomes important. Tasks which exhibit a large discrepancy between their worst case execution and their expected (average) execution should be executed first. In this way, the tasks executing later know if they can use up more processor time. To eliminate the biggest uncertainties early, the tasks should therefore be executed in an order according to the following priorities:

$$p_i = 1/(C_i - \overline{X}_i) \qquad (5).$$

Having decided on the schedule, we can now compute the energy consumption for each of the cases presented here. Note that the processor speed is constant on intervals, and for the WCE and ideal case there is a unique speed:

$$E^{WCE} = (s^{WCE})^2 \sum_{1 \le i \le N} (\overline{X}_i / s^{WCE}) = \left(\left(\sum_{1 \le i \le N} C_i\right) \cdot \left(\sum_{1 \le i \le N} \overline{X}_i\right)\right) / A \qquad (6)$$

$$E^{ideal} = A(s^{ideal})^2 = \left(\sum_{1 \le i \le N} \overline{X}_i\right)^2 / A \qquad (7)$$

$$E^{real} = \sum_{1 \le i \le N} \frac{\overline{X}_i}{s_i} \cdot s_i^2 = \sum_{1 \le i \le N} \overline{X}_i \cdot s_i \qquad (8)$$

Note that the last energy value is dependent on the task ordering. This is were one can reduce the energy consumption by using an optimal ordering.

Consider a simple task set, composed of three tasks $\{\tau_1 = (X_1, 20, 100, 100),\ \overline{X_1} = 16;\ \tau_2 = (X_2, 30, 100, 100),\ \overline{X_2} = 20;\ \tau_3 = (X_3, 40, 100, 100),\ \overline{X_3} = 32\}$. The processor speeds for four cases, plus the energy consumptions are gathered in Table 1. The first two columns refer to the worst and the ideal cases. The "worst case" decides an off-line speed such that the tasks will always meet their deadlines, and uses onöy that speed at runtime. The "ideal case" assumes that the exact execution times are known beforehand and decides an optimal processor speed for each period. The real case implies using a runtime re-scheduling policy. The processor speed is recomputed each time a task finishes execution, that is why we give three speed values for the real case. The "best order" in Table 1 is the order given by the priorities computed as in (5). The "reverse best" is when the task assume an inverse order. From the table we can deduce that an on-line strategy reduces further the energy consumption. Moreover, a good ordering also contributes to energy reduction.

Table 1: An example of case A

$\{\tau_1, \tau_2, \tau_3\}$	WCE	ideal	real expected	
			best order: 2, 3, 1	reverse best: 1, 3, 2
speed, s	0.90	0.68	0.90, 0.77, 0.55	0.90, 0.85, 0.67
normalized energy, E	1.324	1	1.114	1.182

B. Unique Period, Different Deadlines

The problem becomes more complex when the deadlines for the tasks differ: $T_i = T_j = A$, $D_i \ne D_j \le A$, $\forall(\tau_i, \tau_j)$. A deadline monotonic scheduling strategy would, in this case, guarantee feasible schedules up to full processor utilization. Considering the tasks ordered according to their deadlines, the processor speed for each task can be computed as in [10]:

$$s_j^{WCE} = \sum_{j \le i \le N} \left(C_i / \left(D_i - \sum_{1 \le k < j} (C_k / s_k)\right)\right) \qquad (9)$$

and the ideal speed:

$$s_j^{ideal} = \sum_{j \le i \le N} \left(\overline{X}_i / \left(D_i - \sum_{1 \le k < j} (\overline{X}_k / s_k)\right)\right) \qquad (10).$$

As for the on-line scheduling method, the processor speed is recomputed whenever a task finishes execution. The time moment when task j finishes is:

$$t_j = \sum_{1 \le k \le j} (\overline{X}_k / s_k) \qquad (11)$$

The following speed is then re-computed for the next task:

$$s_{j+1} = \sum_{j < i \le N} (C_i / (D_i - t_j)) \qquad (12)$$

As in the previous case, the lowest average energy consumption would be achieved if the tasks with big uncertainties execute first. On the other hand, the deadlines have to be met. With this in mind, we present here an off-line preemptive scheduling strategy which attempts to obtain a low average energy consumption. The algorithm, presented in Fig. 1, uses an initial step in which WCE processor speeds are computed. The actual algorithm assigns time intervals for each task or task part, starting from the lowest priority task. We always schedule

```
begin procedure OfflinePreemptiveSchedule
empty WorkList
  forall deadlines Di, longest..shortest do
    time := Di, empty NextWorkList
    * put all tasks with deadline Di in WorkList
    * order WorkList descending
      using priorities pj := sj/(Cj - Xj)
    forall tasks τk in WorkList do
      if Ck/sk > (time - Di-1) then
        * split τk:
          NextWorkList.add(task with
            C := Ck - (time - Di-1)*sk,
            D := Di-1, X := min(C, Xk))
          Also update Ck := time - Di-1
      end if
      * schedule τk between [time - Ck/sk, time)
      time := time - Ck/sk
    end forall
    * make NextWorkList the new WorkList
  end forall
end procedure
```

Fig. 1. Off-line scheduling algorithm for reduced energy in the case of task sets with unique period and different deadlines

tasks in bursts, between two consecutive deadlines, called scheduling intervals. The tasks having the same deadline are sorted according to priorities first introduced in sub-section A. The tasks exhibiting the smallest discrepancy between their WCET and their expected execution time are scheduled last. Whenever a task does not fit entirely in the current scheduling interval, it is split in two parts such that one of them fits. The other part is treated as a new task, which has to be completed before the current scheduling interval. The on-line re-scheduling algorithm is the same as the one described previously, by equation (12).

Having decided on the schedule, we can now compute the expected energy consumption for all the cases presented here. As in the previous case, the processor speed is constant on intervals. For brevity we will not expand the expressions: $E^{WCE} = \sum_{1 \le i \le N} \overline{X_i} \cdot s_i^{WCE}$, $E^{ideal} = \sum_{1 \le i \le N} \overline{X_i} \cdot s_i^{ideal}$, $E^{real} = \sum_{1 \le i \le M} \overline{X_i} \cdot s_i$.

Note that in the last case the number of intervals with constant speed can increase to *M* as a result of our off-line preemptive scheduling algorithm.

C. Different Periods

Dealing with tasks which have different periods is an even more complex problem. Formally, now $T_i \neq T_j$, $\forall(\tau_i, \tau_j)$. There are several approaches to scheduling in this situation. One could use task hyperperiods, as in [11], and practically return to the case described in sub-section B. Yet, for certain task sets the complexity of the problem, given by the number of instances that have to be analyzed, becomes huge. In the following we present an alternative approach, developed on the fixed priority (rate or deadline monotonic) scheduling framework. Our method consists as before, from an off-line step and an on-line policy.

The off-line scheduling step. The scheduling condition proposed by Liu and Layland [1] is a sufficient one and covers the worst possible case for the task group characteristics. Yet, an exact analysis as proposed in [2] may reveal possibilities for stretching tasks and still keeping the deadlines. Based on this, [12] describes a method to compute the minimal required frequency (speed) for a task set. In similar way, we go further and compute the minimal achievable processor speed $\{s_i\}_{1 \le i \le n}$ for each task τ_i in the task group $\{\tau_i\}_{1 \le i \le n}$. We consider that the tasks in the group are indexed according to their priority.

We compute the speeds in an iterative manner, from the higher to the lower priority tasks. An index *q* points to the latest task which has been assigned a speed. Initially, $q = 0$. Each of the tasks $\tau_i, q < i \le n$ has to be executed before one of its scheduling points S_i as defined in [2]: $S_i = \{kT_j \mid 1 \le j \le i; 1 \le k \le \lfloor T_i/T_j \rfloor\}$, if $T_i = D_i$. If $T_i \neq D_i$, we only need to change the set of scheduling points according to $S_i' = \{t \mid (t \in S_i) \wedge (t < D_i)\} \cup \{D_i\}$. For each of this scheduling points $S_{ij} \in S_i$, task τ_i exactly meets its deadline if:

$$\sum_{1 \le r \le q} \frac{1}{s_r} C_r \cdot \left\lceil \frac{S_{ij}}{T_r} \right\rceil + \frac{1}{s_{ij}} \cdot \sum_{q < p \le i} C_p \cdot \left\lceil \frac{S_{ij}}{T_p} \right\rceil = S_{ij} \qquad (13)$$

Note that for the tasks which already have assigned a speed we used that one, s_r, while for the rest of the tasks we assumed they will all use the same and yet to be computed speed, s_{ij}, which is dependent on the scheduling point. For the task τ_i the best scheduling choice, from the energy point of view, is the smallest of its s_{ij}. At the same time, from (13), this has to be the equal for all tasks $\tau_i, q < i \le n$. There is a task with index *m* for which its lowest speed is the largest among all other tasks. Note that this in not necessarily the last task, *n*. If $q = 0$, this task sets the minimal clock frequency as computed in [12]. Obtaining the index *m*, all tasks between *q* and *m* will use the same speed as *m*. With this an iteration of the algorithm for finding the task speeds is complete. The next iteration then proceeds for $q = m$. Finally the process ends when *q* reaches *n*, meaning all tasks have been given their own off-line speed.

A numerical example is given in Table 2. Note that tasks 3 and 4 can use a lower speed than 1 and 2, while 5 has the lowest processor speed. As a result of slowing down the tasks, the processor utilization changes from 0.687 to 0.994.

Table 2: Numerical Example for Off-line Scheduling

Task τ			Off-line Speed s	
No.	WCET (C)	Period (T)	value	iterations needed
1	1	5	0.70	1
2	5	11	0.70	1
3	1	45	0.56	2
4	1	130	0.56	2
5	1	370	0.42	3

The on-line strategy. During runtime we exploit the stochastic nature of the system. It is important to use the variations in execution length of the various task instances to be able to execute at lower processor speed and, thus, consume less energy. In [12] the only situation when a task executes slower is when

it is the only one running and has enough time until the next task instance arrives. In all other situations tasks are executed at the speed dictated by the off-line analysis. In [16] tasks are slowed down to cover their WCET at runtime, independent of other tasks, using several checking/re-scheduling points during a task instance. Our method is perhaps most resemblant to the optimal scheduling method OPASTS presented in [11]. Yet, OPASTS performs analysis over task hyperperiods, which may lead to working on a huge number of task instances for certain task sets. Our method, briefly described next, keeps a low and the same computational complexity, regardless of the task set.

An early finishing task may pass on its unused processor time for any of the tasks executing next. But this time slack can not be used by any task at any time since deadlines have to be met. We solve this by considering several levels of slacks, with different priorities, as in the slack stealing algorithm [3]. The slack in each level is a cumulative value, the sum of the unused processor times remaining from the tasks with higher priority. Whenever a task finishes its execution early, it contributes to the corresponding slack level with the amount of unused time. This is then distributed to the lower priority tasks present in the system, according to their expected execution time. The slack distribution is performed such that the highest processor speeds are reduced, lowering the energy consumption. We present a more detailed description of the on-line strategy in [15]. We also give a proof that our scheduling method using slack levels meets all the deadlines.

III. Task Level Scheduling Decisions

Task-level voltage scheduling has captured the attention of the research community rather recently [19]. Fine grain scheduling, where several re-scheduling points are used inside a task were presented in [16,18]. In [18] statistical data is used to improve the task level schedule, by slowing down different regions of a task according to their average execution time. Our approach produces voltage schedules only when a task starts executing, while using stochastic data more aggressively.

In our model a task τ_i can be executed in phases, at different available voltages, depending on its allowed execution time A_i. The ideal case states that the most energy is saved when the processor uses the voltage for which the task exactly covers its allowed execution time. This corresponds to an ideal voltage which may not overlap with the available voltages. A close to optimal solution is to execute the task in two phases at two of the available voltages. These two voltages are the ones bounding the ideal voltage [14,19].

An important observation is that tasks may finish, and in many cases do finish, before their worst case execution time (WCET). Therefore it makes sense to execute first at a low voltage and accelerate the execution, instead of executing at high voltage first and decelerate. In this manner, if a task instance is not the worst case, one skips executing high voltage regions.

In the following we will distinguish between three modes of execution for a task, as depicted in Fig. 2. The ideal case (mode 1) is when the actual execution pattern (the number of clock cycles) becomes known when the task arrives. We can stretch then the actual execution time of the task to exactly fill the allowed time. This mode requires rather accurate execution pattern estimates, depending on the input data, and therefore is rarely achievable in practice. The second mode (mode 2) is the WCE stretching - the voltage schedule for the task is determined as if the task will exhibit its worst case behavior. These first two modes use at most two voltage regions, and therefore at most one DC-DC switch. The third mode (mode 3), described in more detail next, uses stochastic data to build a multiple voltage schedule. The purpose for using stochastic data is to minimize the average case energy consumption. Note that the voltage schedules in all these three modes are decided at a task instance arrival. Unlike in [16,17] no rescheduling is done while the task is executing. The only overhead during task execution is the one given by the changes in the supply voltage.

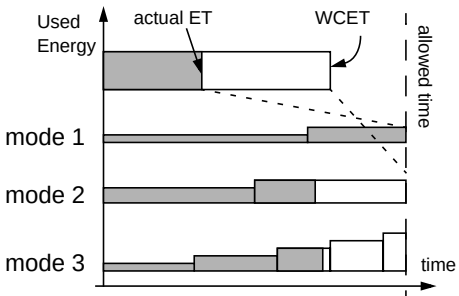

Fig. 2. Voltage scheduling modes for tasks: 1) ideal schedule, 2) WCET oriented schedule, 3) stochastic schedule.

The stochastic voltage schedule (mode 3 in Fig. 2) for a task is obtained using the probability distribution of the execution pattern for a task (the number of clock cycles used). This probability distribution can be obtained off-line, via simulation, or built and improved at runtime. The actual voltage schedule is obtained by minimizing the expected value of the energy consumption. We present adetailed description of computing such a schedule in [15].

Two examples of stochastic voltage schedules are given in Fig. 3. We assumed a normal probability distribution with the mean of 70 cycles, and standard deviation of 10. WCE is 100. Assuming we only have four available clock frequencies f, f/2, f/3, and f/4, we give two voltage schedules obtained for two different values of the allowed execution time. The schedules are given in number of clock cycles executed at each available frequency. The allowed execution time is reported in percentage of the time needed for executing the worst case behavior at the highest clock frequency (f).

Next we present an experiment that examines the energy gains of using a stochastic voltage schedule at task level. For this we considered a single task with execution time varying between a best case (BCE) and a worst case (WCE) according to a normal distribution. All distributions have the mean (BCE+WCE)/2 and standard deviation (WCE-BCE)/6. For a several cases ranging from highly flexible execution time (BCE/WCE is 0.1) to almost fixed (BCE/WCE is 0.9) we built stochastic schedules for a range of allowed execution times (from WCE to 3x WCE). We assumed that our processor has 9 different voltage levels, equally distributed between f and f/3.

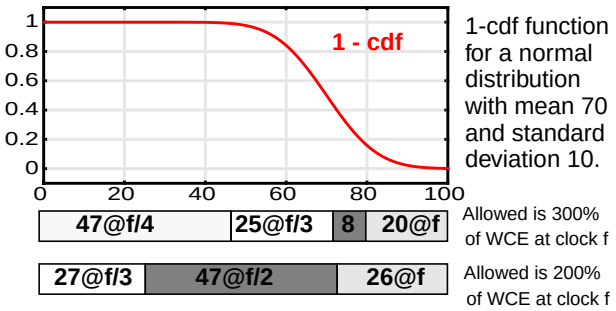

Fig. 3. Examples of stochastic voltage schedules for a task with normal distribution execution time and worst case behavior of 100 cycles

For a large number of task instances generated according to the given distribution we computed both the energy of the stochastic schedule (mode 3 in Fig. 2) and the WCE-stretch schedule (mode 2 in Fig. 2). We depict in Fig. 4 the average energy consumption of the stochastic schedule as a part of the WCE-stretch schedule. Note that when the allowed time approaches either WCE or 3-times WCE, the energy consumptions become equal. The lowest possible clock frequency is f/3 which anyway means 3-times WCE, so there is no better schedule for these cases. On the other hand when the allowed time closes WCE, there is no other way but to use the fastest clock. Somewhere between the slowest and the fastest frequencies (Allowed/WCE = 2) is the largest energy gain since the stochastic schedule can use the whole spectrum of available frequencies. Note that the energy gains become more important when the task execution time varies much (BCE/WCE closes 0.1). Notice also that WCE-stretch already gains very much energy compared to the non-scaling case. For example when the allowed time is twice the WCE, the WCE-stretch energy is around 25% of the no-scaling energy. A stochastic approach contributes even more to these gains, as the figure shows.

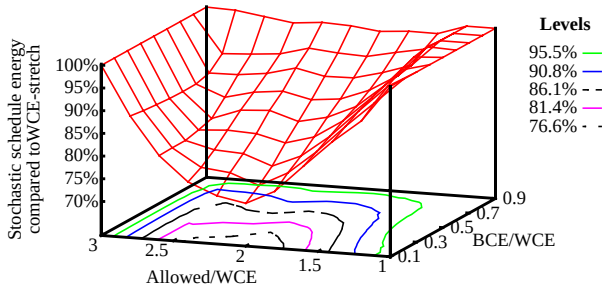

Fig. 4. The average energy consumption of a stochastic voltage schedule compared to the energy consumption of a WCE- stretch schedule.

IV. Conclusions and Future Work

In this paper we addressed scheduling methods targeting energy reduction in hard real-time systems containing dynamic supply voltage processors. Both task-set and individual task level scheduling decisions were described. For task sets we started from the simple case of tasks having the same unique deadline and period. We then generalized the problem by considering different deadlines, and then different periods. In all cases we showed that more efficient schedules, from the energy point of view, can be derived when stochastic data is taken into consideration. We also presented a task level stochastic voltage schedule which is able to reduce the average energy consumption of a task even more, compared to classic approaches.

As future work, we plan to do extensive experiments using all the scheduling methods presented here, focusing more on the task-set level policies. We also want to perform a thorough theoretical examination of the most general case and eventually come up with realistic lower bounds for the energy consumption of systems containing stochastic task sets.

References

[1] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprograming in a hard real time environment," *JACM* 20 (1), 1973, pp. 46-61.

[2] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," *Proc. of the Real-Time Systems Symposium* 1989, pp. 166-171.

[3] J. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," *Proc. of the Real-Time Systems Symposium* 1992, pp. 110-123.

[4] L. Benini and G. DeMicheli, "System-level power optimization: techniques and tools," *ACM Trans. on Design Automation of Electronic Systems*, No. 2, Vol. 5, April 2000, pp. 115-192.

[5] Pedram M., "Power optimization and management in embedded systems," *Proc. of ASP-DAC 2001*, pp. 239-244.

[6] K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, and T. Kuroda, "A 300MIPS/W RISC core processor with variable supply-voltage scheme in variable threshold-voltage CMOS," *Proc. of the IEEE Custom Integrated Circuits Conference 1997*, pp. 587-590.

[7] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms,"*Proc. of the '98 ISLPED*, pp 76-81.

[8] A. Chnadrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: an approach for energy efficient computing," *Proc. of the '96 ISLPED*, pp. 347-352.

[9] M. Weiser, B. Welch, A Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Proc. of the First Symposium on Operating Systems Design and Implementation*, November 1994.

[10] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *Proc. of the 36th Symposium on Foundations of Computer Science*, pp. 374-382, 1995.

[11] I. Hong, M. Potkonjak, and M.B. Srivastava, "On-line scheduling of hard real-time tasks on variable voltage processor," *Digest of Technical Papers of the 1998 ICCAD*, pp. 653-656.

[12] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," *Proc. of the 36th DAC, 1999*, pp. 134-139.

[13] Y.-H. Lee and C.M. Krishna, "Voltage-clock scaling for low energy consumption in real-time embedded systems," *Proc. of the 6th International Conference on Real-Time Computing Systems and Applications*, 1999, pp. 272-279.

[14] F. Gruian and K. Kuchcinski, "LEneS: task scheduling for low-energy systems using variable voltage processors," *Proc. of ASP-DAC2001*, pp. 449-455.

[15] F. Gruian, "Hard real-time scheduling using stochastic data and DVS Processors," submitted for review.

[16] S. Lee and T. Sakurai, "Run-time voltage hopping for low-power real-time systems," *Proc. of the 37th DAC*, 2000, pp. 806-809.

[17] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *Special Issue of IEEE Design and Test of Computers*, October 2000.

[18] D. Mossé, H. Aydin, B. Childers, and R. Melhem, "Compiler-assisted dynamic power-aware scheduling for real-time applications," *Worksop on Compilers and Operating Systems for Low-Power*, October 2000.

[19] T. Ishihara,H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proc. of the '98 ISLPED*, pp 197-202.

[20] http://www.transmeta.com