Task-Dependent Processor Shutdown for Hard Real-Time Systems

Henrik Lipskoch[1] and Frank Slomka[2]

[1] C.v.Ossietzky University Oldenburg, Department for Computer Science, Germany
[2] Ulm University, Institute for Embedded Systems, Germany

Abstract. Mobile devices relying on batteries can save energy by using low-power modes of their processors. In a hard real-time environment, one has to prove the real-time feasibility and thereby to guarantee that energy saving methods do not violate real-time constraints. Besides the processor's unavailability during low-power mode, the transition to and from the mode consumes energy and time.
This work introduces a task-dependent policy for mode switching and compares it to procrastination techniques from the literature. The low-power interval is placed between occurrences of one task of the task set such that low-power mode and instances of this task do not overlap.
Optimisation of task to depend on, duration of low-power, and shutdown rate is done with the help of a hard real-time test to provide feasible results. The used test provides analysis for preemptible, deadline scheduled task sets. Tasks are allowed to have periodic, periodic with jitter, sporadic, or other behaviour regarding their occurrence.
Thus, this work extends the applicability of processor shutdown to such environments. And since parameters are determined off-line, apart form a programmable real-time clock, no power consuming extra circuitry is needed. However, the method comes with a slight modification of the task set.

1 Introduction

Portable or hand-held devices rely on batteries, which's capacity determines the operating time of the system. Thus, minimising power consumption will increase the time of operation. Often these devices are subject to hard real-time constraints, e.g. for communication. In order to make them use the energy delivered from the battery more efficient, several energy saving options exist. Additional to dynamic voltage scaling, there exists the possibility to make use of low-power modes, e.g. sleep states or deep-sleep. Here, the devices have parts switched off (eg. no V_{cc} for an unused FFT) or frozen (processor clock stopped) or even the complete system is shut down by stopping the system clock or switching it into standby state.

Low-power modes provide an efficient way to save power during idle periods, even without loosing information stored in processor registers.

Almost every microprocessor can be woken up by an interrupt of its real-time clock (RTC), even out of the deepest mode. Because the timer has to be programmed beforehand, the task system has to be analysed beforehand. The analysis has to determine how

long at least the processor can stay disabled and when it is to be woken up, subject to retaining deadlines, i.e. to calculate only spare time is not enough.

Mobile applications, like mobile data loggers, are likely to have communication for data exchange that must not be interrupted. The point in time when to initiate communication can be chosen manually (sporadic) as well as periodically. In either case the system has to function properly to provide reliable data. Sensors may use interrupt based communication to a central plant, or time slotted data transmission. These communication systems cause jitters.

Thus, we want to minimise power consumption for devices having low-power modes with non-negligible switching overhead and hard real-time tasks running on them, scheduled with earliest deadline first scheduling. Whereby the tasks occurring behaviour is described by an event stream and may include periodic behaviour, jitters, be sporadic, or other, e.g. aroused by drifting clocks. Additional, to avoid extra control circuitry, we want to determine off-line, when and for how long to transit into low-power mode.

The content of this work is organised as follows. First, we give an overview on and discuss works dealing with power saving in hard real-time environments. Second, we briefly introduce the used feasibility test, after which we model and analyse our processor shutdown method. Experiments include comparisons to optimal shutdown, with and without procrastination, and LC-EDF procrastination technique. A concluding section summarises the results.

2 Related Work

We regard works exploring dynamic voltage scaling and power management guaranteeing hard real-time feasibility in worst-case situations, i.e. works providing feasibility for all possible schedules.

2.1 Dynamic Voltage Scaling

Works addressing voltage scaling for hard real-time task sets are [**?**] and [**?**], [**?**], and [**?**], whereof the former two works consider periodic task sets only, and the latter two are able to optimise for other occurrence behaviour as well. Nevertheless, all four works neglect transition costs for mode switching.

Works considering switching overhead are [**?**] and [**?**]. Both works apply to periodic task sets, where the tasks can have dependencies and individual deadlines. The period is identical in either case for the whole task set. Since both works have their task sets scheduled without preemption and calculate a complete schedule, i.e. determining jobs to execute, assigning execution start and end, and processor mode, either proves the exponential complexity of the energy saving problem.

2.2 Processor Shutdown

Dynamic power management is about to explore low-power modes for processors and peripherals at run-time. A lot of research in this area has taken place, for overview and

comparison consider [**?**] or [**?**]. All authors categorise works on dynamic power management into time-out policies, predictive policies, and stochastic policies. The first shuts down the processor after it has been idle for a certain amount of time. The second addresses the wake-up penalty, it tries to predict the next time the processor has to be ready for processing. Stochastic policies use task set information in form of probability distributions on their occurrence, and try to fit time-outs and wake-up predictions for these minimising wake-up penalty and maximising time spent in low-power mode. None of the works mentioned in both surveys addresses hard real-time constraints, though most are aware of a latency caused by back transition from low-power mode to run mode.

The following works are online algorithms, providing deadline retention. Each needs extra control circuitry monitoring incoming tasks and maintaining data for forecasting next probable shutdown durations.

Power management for devices is used in [**?**]. The authors' approach makes use of a resource request list obtained by jobs in the schedulers run-queue. Their algorithm needs the list for a look ahead to ensure deadlines are met; The authors neglect the appliance of their approach to hard real-time systems, because of the latency due shut down devices an incoming job will have.

The authors of [**?**] extend the previously mentioned work by processor voltage scaling to stretch job execution and thus to cluster device usage with the aim of extended idle intervals. If there is no job requesting a device, the device is shut down; the next incoming job requesting this device will be confronted with a delay to switch the device on. The authors solve this drawback by requiring an extra maximal blocking time in their task model, but their power saving and real-time testing algorithm can only deal with periodic tasks.

A static schedule allowing estimation of switching costs is created in [**?**]. The work considers periodic and sporadic tasks. Periodic tasks are scheduled statically. For sporadic tasks, slots are inserted, which may be moved during run-time to further improve the schedule. Task jitters are not taken into account. Reduction of power dissipation is achieved by using dynamic power management and voltage scaling. The latter is preferred, based on an early argument, that neglects leakage power ([**?**]).

Task procrastination may extend idle periods, an algorithm exploiting this was presented in [**?**], known as LC-EDF. Only periodic tasks with deadlines equal to their periods are considered. The procrastination time δ_ω for a task ω is calculated via

$$\sum_{\tau \in \text{Tasks} \setminus \{\omega\}} \frac{c_\tau}{T_\tau} + \frac{c_\omega + \delta_\omega}{T_\omega} = 1, \tag{1}$$

with c,T being worst-case execution time and period. The Equation targets to keep utilisation below 100%. Deadlines shorter than periods may be handled by dividing by deadlines instead of periods, but the result will be very pessimistic and in cases with small deadlines compared to periods, the sum will grow beyond 1 and no procrastination will take place. Further, the authors assume external circuitry, e.g. special purpose device, to shut down and switch on the processor.

The previous work is extended by global processor slowdown in [**?**], with the same drawback regarding short deadlines, and their still requires extra circuitry monitoring job queues.

The work presented in [**?**] provides lower energy consumption, than the previous two. The authors relax the problem of periodic tasks to have their deadlines lower than the periods, but not higher. The approach works on a job set, which is said to contain a number of jobs with their release times, deadlines, and execution times. It considers individual job slow down, and procrastinates idle intervals to merge scattered frames of idle time. For every occurring jitter or other non-periodic behaviour, the job set monitoring device has to perform a re-calculation.

3 Test on Real-Time Feasibility

As motivated in the introductory section, we do not want to limit us to periodic only task systems. Instead, task jitters, sporadic tasks, and other behaviour are included as well. A worst-case execution time is assumed to obtain upper bounds on worst-case demand. We further assume the task set to be scheduled earliest deadline first. We assume a fixed relative deadline for each task, the deadline may be arbitrary.

A real-time feasibility test using the hyper-period will have a high complexity as the hyper-period can grow very large, e.g. if some periods contain large prime numbers. We reduce complexity by using a feasibility test which is based on an abstraction.

An upper bound of the execution demand to calculate a lower bound on available idle time is achieved via the demand bound function, introduced in [**?**], together with the event stream model. We use here the same event stream based calculation of the demand bound function and its approximation as given in [**?**] (and refer to that work for a more detailed introduction):

Let Γ be a set of tasks scheduled under earliest deadline first policy. For a task $\tau \in \Gamma$, let c_τ and d_τ be its worst-case execution time and deadline. Let each event stream be denoted by a sequence $\left(a_\tau^{(n)}\right)_{n\in\mathbb{N}}$, it is $a_\tau^{(n)}$ denotes the minimal time to pass for n tasks of type τ to be instantiated. Note, that an event stream is not a schedule, it is an abstraction. Then

$$D(\Delta) := \sum_{\tau\in\Gamma} \max\left\{i \in \mathbb{N}_0 : a_\tau^{(i)} + d_\tau \leq \Delta\right\} \cdot c_\tau \tag{2}$$

is an upper bound on the processing demand to be fulfilled in a time span of length Δ.

Then a system is hard real-time feasible if

$$\text{for all } \Delta \in \mathbb{R}: \quad D(\Delta) \leq \Delta. \tag{3}$$

By approximation (replacing the max-terms in (**??**) by linearisations, see [**?**]) and by testing only those time points at which a deadline is to be met, the complexity of the test is reduced to be polynomial, e.g. if all event streams are approximated after the second element, it is reduced to be twice the number of tasks.

For abbreviation, we define

$$m(\tau, x, \Delta) := \max\{i \in \mathbb{N}_0 : a_\tau^{(i)} + x \leq \Delta\}. \tag{4}$$

The Case of Two Never Overlapping Consecutive Executed Tasks: Suppose there are two tasks ρ, λ in the task set Γ, ρ to trigger λ, the latter to finish before the former has to be finished next, thus never overlapping (remember: earliest deadline first scheduling). Suppose too, that a job of task λ, if running, has always highest priority.

For better understanding the situation, we introduce a third task into the task graph, σ, which's purpose is to represent the trigger of λ, thus σ consumes no execution time, but is assigned a deadline: the latest time to trigger λ. The latter then is assigned a deadline equal to its execution time yielding highest priority. This situation is depicted in Fig. **??**. The latest time to finish an instance of λ must not be later than right before the latest start time of the next instance of ρ, which is equal to $d_\rho - c_\rho + a_\rho^{(2)}$, with $a_\rho^{(2)} \in \left(a_\rho^{(n)}\right)_{n\in\mathbb{N}}$.

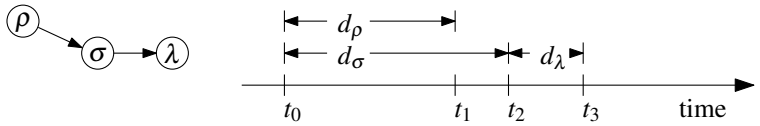

Fig. 1: Task graph of three consecutive tasks, it is $d_\lambda := c_\lambda$, $d_\sigma + d_\lambda \leq d_\rho + a_\rho^{(2)} - c_\rho$, and $d_\rho \leq d_\sigma$

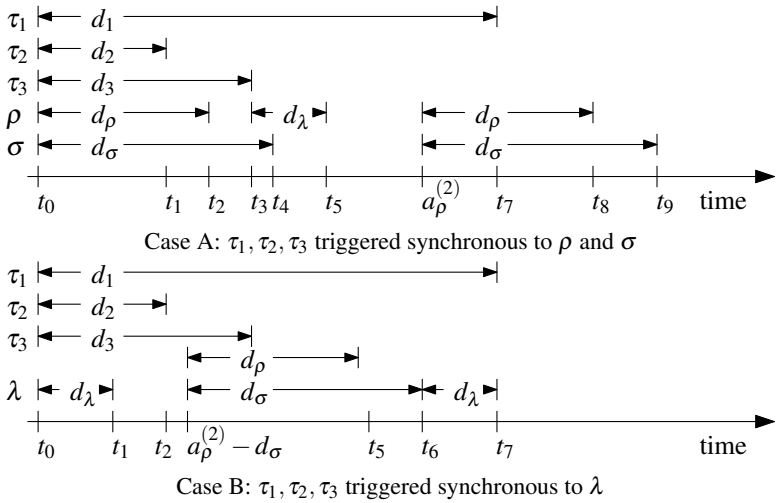

Fig. 2: Two worst-case situations in dealing with a task graph

Now two situations denote a worst-case, each depicted in Fig. **??**. Either task ρ is triggered synchronous with all other tasks, task σ triggers task λ the latest at d_σ (Case A), or d_σ time units later than the last task ρ was triggered, an instance of λ starts

running, blocking the processor for every other task triggered at the same instant (Case B). In Case B the next instance of task ρ is to be finished no later than $d_\rho + a_\rho^{(2)} - d_\sigma$. Then in Case A, the demand is denoted by

$$\begin{aligned} D_A(\Delta) = & \sum_{\tau \in \Gamma \setminus \{\lambda\}} m(\tau, d_\tau, \Delta) c_\tau \\ & + m(\rho, d_\sigma + d_\lambda, \Delta) c_\lambda, \end{aligned} \tag{5}$$

whereas in Case B the demand is denoted by

$$\begin{aligned} D_B(\Delta) = & \sum_{\tau \in \Gamma \setminus \{\rho, \lambda\}} m(\tau, d_\tau, \Delta) c_\tau \\ & + m(\rho, c_\lambda, \Delta) c_\lambda \\ & + m(\rho, d_\rho + a_\rho^{(2)} - d_\sigma, \Delta) c_\rho. \end{aligned} \tag{6}$$

We cannot decide which of the two cases dominates the other:

Lemma 1. *Both D_A and D_B denote a worst-case not included in the other, thus both have to be used for the real-time feasibility test.*

Proof. Let $\Delta = c_\lambda$, then

$$D_A(\Delta) - D_B(\Delta) = -c_\lambda.$$

Let now $\Delta = d_\rho = a_\rho^{(2)}$, then

$$D_A(\Delta) - D_B(\Delta) = c_\rho.$$

The former example may repeat for $\Delta = c_\lambda + a_\rho^{(2)}$, and the latter for $\Delta = d_\rho + n \cdot a_\rho^{(2)} - d_\sigma$. Thus in general it is neither D_A always greater than D_B nor vice versa.

Since both demand bound functions start with all tasks being synchronous, except ρ and λ, there are no other demand bound functions denoting further worst-cases not already included in D_A, D_B.

We conclude for task graphs with precedence constraints, containing tasks having highest priority, two demand bound functions have to be identified, one testing the situation when the task graph is triggered synchronous with all other tasks, and the other testing the situation when each of the high priority tasks is triggered synchronous with the other tasks.

4 Task-Dependent Shutdown Policy

We first provide a means of comparison. Let t_{lp} be the duration of an interval where switching to and from low-power mode is applied. Then, the actual time spent in low-power mode, t_{low}, depends on the time needed for transition, t_{switch}; it is $t_{\text{low}} = t_{\text{lp}} - t_{\text{switch}}$. The duration t_{lp} results in saved energy if and only if the reduced amount of

energy consumed within t_{low} weighs up the energy cost for the transition and is longer than t_{switch}. Therefore, t_{lp} has to exceed the break-even time, t_{BE}, which is

$$t_{\text{BE}} = \max\left\{t_{\text{switch}}, \frac{E_{\text{switch}} + P_{\text{low}} t_{\text{switch}}}{P_{\text{run}}}\right\}. \tag{7}$$

If a processor has multiple low-power modes, one has to determine which mode to use. According to the time available, the solution falls on the mode yielding a maximal difference $t_{\text{lp}} - t_{\text{BE}}$, which denotes the maximal energy saved.

Let $t_{\text{lp}}(\Delta)$ denote the least time reserved for low-power mode within an interval of length Δ, and let #transitions(Δ) be the maximal number of transitions into and out of low-power mode, occurring within an interval of length Δ. Then we define effectiveness of a shutdown method as the fraction of time spent in low-power mode:

$$\text{Eff}(\Delta) := \frac{1}{\Delta}\left(t_{\text{lp}}(\Delta) - \#\text{transitions}(\Delta) \cdot t_{\text{BE}}\right). \tag{8}$$

4.1 Shutdown in Between Two Jobs of the Same Task

Our main idea is to exploit the case of *two never overlapping consecutive executed tasks* (see paragraph in Section **??**). Processor shutdown is modelled as a task, and by depending on another task, the rate of shutdown is inherited from the task's instantiation rate.

Method: Append each job of a certain task ρ with an interval of duration c_l at low-power mode. Finish the low-power interval before the next job of ρ has to be begun. Determine a latest start d_l of the low-power interval.

This has the effect that these two participants will neither occur at the same time nor will they intersect each other.

Subject to real-time constraints are the duration of low-power, the deadline for its invocation, and the task to append.

This method has to be implemented in a task itself. As an advantage, if a job at run-time finds it will need less processing time than predicted for the worst-case, it could increase the duration of the low-power task by an amount yet to be determined.

According to (**??**), for a task ρ and duration c_l the efficiency of the method is

$$\text{Eff}(\Delta) = \frac{m(\rho, d_\rho, \Delta) \cdot (c_l - t_{\text{BE}})}{\Delta} \tag{9}$$

or with avg being the average number of jobs per time unit

$$\text{Eff} = \text{avg}(\rho) \cdot (c_l - t_{\text{BE}}). \tag{10}$$

Algorithm **??** determines the parameters ρ, d_l, and c_l. The if-conditions in lines **??** and **??** ensure real-time feasibility of the solution.

The used feasibility tests are those from (**??**), (**??**), in the approximated form.

For each task in the task set, the algorithm explores the design space by interval bisection over possible execution times c of the low-power task (outer while loop), and

tries to determine a feasible deadline by interval bisection over a range of deadlines (inner while loop). If no deadline is found for c, this defines the new maximum $c_{\max}$, and in the opposite case, it defines the new minimum $c_{\min}$.

The decisions in lines **??** and **??** are based on two properties of D_A and D_B. Referring to Fig. **??** and (**??**),(**??**), increasing deadline d_σ leads for D_A to a more relaxed problem, and decreasing d_σ will do the same for D_B. Thus, if a deadline d_σ leads to an infeasible system according to D_A, then increasing the deadline will relax the problem and might yield a feasible system. On the other hand, if d_σ leads to an infeasible system according to D_B, then decreasing it will probably lead to a feasible system.

Algorithm 1: Determination of parameters for task-dependent shutdown

Input: Task set Γ, break-even time t_{BE}
Output: Task ρ, to be followed by time c_l in low-power mode, the latest d_l time units past instantiation of ρ, efficiency $\mathrm{best}_{\mathrm{eff}}$

```
1  best_eff := 0;
2  for τ ∈ Γ do
3      c_max := min{Δ − d(Δ, Γ \ {τ}) : Δ > 0};
4      c_min := t_BE + best_eff/avg(τ);
5      while c_max − c_min < ε_c do
6          c = c_min − (c_max − c_min)/2;
7          d_min := d_τ;
8          d_max := a_τ^(2) + d_τ − c_τ − c;
9          solution_found:=false;
10         if d_min < d_max and feasible_CASE_A (Γ, τ, (c, d_max)) and feasible_CASE_B
           (Γ, τ, (c, d_min)) then
11             while d_max − d_min > ε_d and not solution_found do
12                 d := d_min − (d_max − d_min)/2;
13                 if feasible_CASE_A (Γ, τ, (c, d)) then
14                     if feasible_CASE_B (Γ, τ, (c, d)) then
                           /* CASES A and B successful                */
15                         solution_found:=true;
16                     else
                           /* CASE B failed, increase d               */
17                         d_max := d;
18                 else
                       /* CASE A failed, decrease d                   */
19                     d_min := d;
20         if solution_found then
21             eff_τ := avg(τ) · (c − t_BE);
22             if best_eff < eff_τ then
23                 (ρ, c_l, d_l) := (τ, c, d);
24                 best_eff := eff_τ;
25             c_min := c;
26         else
27             c_max := c;
```

4.2 Shutdown in Between n Jobs of the Same Task

Following the same argumentation as in the subsection before, it can also be thought of to wait for a number of instances, say n, then to apply shutdown, and then again wait for the same number of instances before the next shutdown. This can lead to longer low-power durations. And since the number of shutdowns depends on the task's instantiation rate, this modification may reduce the number of switches and may increase the efficiency.

The critical situations are the same as for the method in the previous subsection, because each shutdown transition is again placed between two consecutive task instances.

The efficiency formula in line **??** is replaced by "$\text{eff}_\tau := \frac{avg(\tau)}{n} \cdot (c - t_{\text{BE}})$".

With this modification, the Algorithm **??** gets an extra loop for each task, where it is to cycle through possible factorisations, $n = 1, \ldots,$ of the task period, stopping cycling, when no further improvement is likely, that is $c_{\min} > c_{\max}$ (Note, in line **??**, "avg(τ)" is replaced by "$\frac{\text{avg}(\tau)}{n}$", thus becomes less by the number of task instances to wait for, this becomes eventually the source of overlapping bounds before entering the loop in line **??**).

We refer to this modification as "Alg. **??** + mod." in the following section.

5 Experiments

We have investigated the effectiveness of each method by applying them on two different task sets, of which one is a periodic task set, and the other contains jitters and deadlines not equal to periods. For each task set we computed the effectiveness of either method along a range of break-even times and compared them to the best possible power saving, which we computed for a range of sample schedules. We applied best power saving with ("Opt. w/ p.") and without procrastination ("Opt. w/o p.").

Best possible power saving is hereby understood as optimisation by inserting shutdown intervals where the optimisation makes use of a beforehand computed occurrence schedule, i.e. knows for certain all occurrences, and thus can globally optimise over the complete schedule.

As our optimisations yield static results designed for the worst-case of processor demand, they will not vary for individual schedules. On the other hand, variations will occur for each of the online-methods. Therefore, we created random schedules for each of our experiments and calculated the mean, a 5% minimal bound, and a 95% maximal bound, thus both together depict a 90% confidence interval around the mean value, and to illustrate the variations, our figures depict efficiency as bands.

We did not explicitly calculate values for the online method of [**?**], but since it performs better than the LC-EDF method, we believe their results to be between LC-EDF and the optimal with procrastination.

Figures **??** and **??** show the cost dependent effective energy savings obtained by each method for our examples. On the y-axis it is shown the achieved portion of time spent in low-power mode, the rest of the time the system spends processing tasks or transiting from and into low-power modes. The x-axis gives the minimal duration for one mode transition, the break-even time.

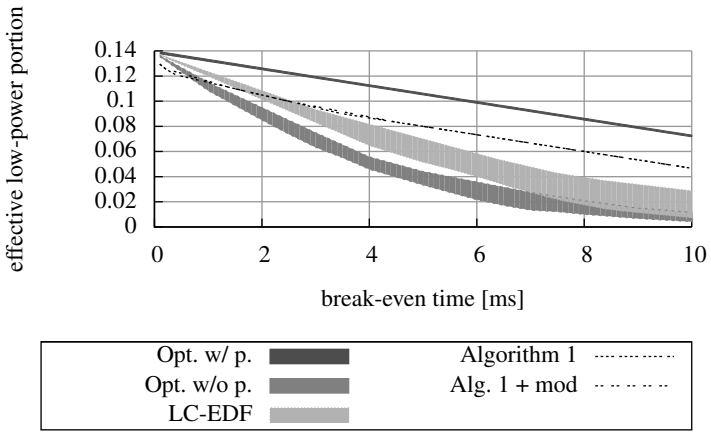

Fig. 3: Experiment Palm-pilot Original

The first experiment is on a task set given in [**?**]. It consists of seven periodic tasks, periods ranging from 20 to 150 ms, having their deadline equal to their periods. The tasks are to be run on a Palm-Pilot. The processor utilisation is 86.17% leaving 13.83% spare time for power saving.

Figure **??** shows the results. The figure also depicts the range of obtained efficiency values along our sample schedules for the LC-EDF method, since it is applicable here ("LC-EDF"). It consists of a band, depicting the range of the efficiencies of 90% obtained for the sample schedules, 5% lie above, 5% lie below the band. We observe the results of our optimisations overlapping, our modification seems not to have an effect. For a break-even time of 1 ms our methods intersect the ideal optimisation without procrastination, and for a break-even time of 2 ms, our methods intersect the LC-EDF, and supersedes it for a break-even time of 4 ms.

The next task set is taken from [**?**]. It represents software for an aircraft controller. It consists of 17 tasks, with 9 having jitters. Periods are in the range of 1 s to 800μs. Processor utilisation is 65.2%, leaving a slack of 34.8% available for power saving. Experimental results are depicted in Fig. **??**.

We observe, that the policy to append shutdown after each job of a task does not perform well. The reason is it cannot trade off rate against duration of shutdown, which the modification is able to do. The latter is close to the optimal shutdown without procrastination, although our method lacks the advantage of online knowledge.

Procrastination estimation via (**??**) yields a delay of -0.1816 s per second, disabling LC-EDF, because this is an example with some tasks having deadlines shorter than their periods.

Additional, this example contains tasks with deadlines higher than their periods. Therefore, the work of [**?**] is not applicable here as well.

That is, our method is the only applicable in this case.

Note that for the feasibility test, we have limited the number of test points to be 10 for every task in both examples, which made the computations very fast (in both cases

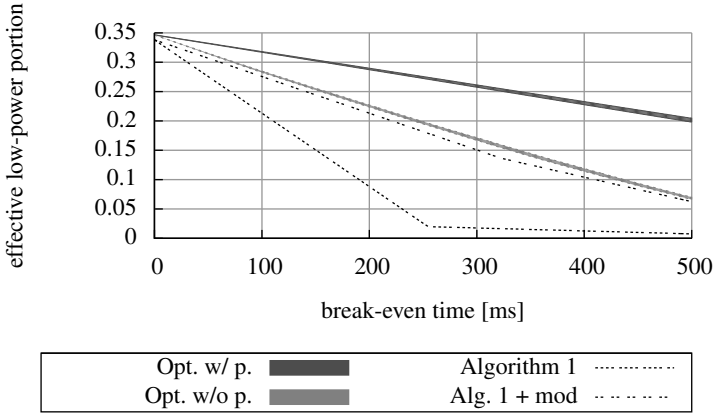

Fig. 4: Experiment Aircraft Control

results were obtained in the order of seconds on a common computer with a 2 GHz CPU).

6 Conclusion

In this work, we have presented our method of task-dependent processor shutdown, where the transition into low-power mode inherits its activation rate from a certain task in a given task set. We have compared the method with optimal shutdown and with the LC-EDF method. Our examples show that our method performs better than LC-EDF for higher break-even times.

We used a fast real-time feasibility test allowing us to model periodic, jittering, or sporadic tasks. This enables guaranteed real-time feasible power optimisations with low complexity, and extends the applicability of processor shutdown to these kinds of occurring behaviour. Further, we can allow task deadlines to be greater than task periods.

Our optimisations are done off-line, yield predictable shutdown intervals, and thus be of use by an embedded software programmer in programming real-time clocks beforehand to use also deep sleep processor states.

The off-line prediction can also be understood as guarantee on maximal energy consumption a task set may yield, helping the designer to take right decisions if there is more than one implementation of the embedded software available.

Our solution needs a slight modification of task implementation, but does not require extra circuitry.