Integrating Real Time and Power Management in a Real System

Martin P. Lawitzky[‡§¶] David C. Snowdon[‡§] Stefan M. Petters[‡§]

[‡] NICTA*
Sydney, Australia

[§] University of New South Wales
Sydney, Australia

[¶] TU München
Munich, Germany

firstname.lastname@nicta.com.au

Abstract

Deploying dynamic voltage and frequency scaling (DVFS) techniques in a real-time context has generated some interest in recent years. However, most of this work is based on highly simplifying assumptions regarding the cost and benefit of frequency scaling. We have integrated a measurement-based DVFS technique with an EDF based scheduling framework. This enables the use of the dynamic slack caused by the variability of execution time, to reduce energy consumption and thus extend battery life or reduce thermal load. We have tested the approach using hardware instrumentation on a real system. This paper describes not only the theoretical basis for the work, but also our experiences with DVFS when confronted with physical reality.

1 Introduction

Power management in embedded devices may be motivated by several factors: extended battery lifetime, reduced need for heat sinks and other thermal dissipation devices, a limited power supply (e.g. a solar powered system), or simply improved environmental sustainability.

Various policies for energy savings have been widely deployed in portable devices like laptop computers without real-time requirements. However, due to their interactive nature, heuristics, with ill defined impact on the temporal behaviour of the system are acceptable. This is obviously not the case for real-time systems where temporal behaviour is considered a prime system property similar in importance to the functional behaviour.

In the last ten years a large body of work has been devoted to the integration of power management policies combined with real-time scheduling. However, most approaches assume an inversely proportional relationship between the CPU core frequency and execution time and ignore issues like a substantial frequency switching cost, static power consumption or the effect of a changing memory frequency on performance and power. All of these issues are evident in real hardware platforms.

Within this work we attempt to take our experience with the physical reality of DVFS and develop an integrated real time and power-management scheduling framework. In order to perform DVFS in a real-time environment, tracking of dynamic slack is essential. Dynamic slack is encountered thanks to the difference between the worst case execution time (WCET) and the actual execution time. For most software the actual execution time is subject to substantial variability as the code is subject to different input parameters at run time.

The RBED work by Brandt et al. [1, 2] is an earliest deadline first (EDF) based scheduling framework. Tracking of system slack is an integral part of this scheme. Major advantages of the approach are high utilisation (thanks to EDF), and integrated non-RT and RT scheduling. It does this by temporal isolation and thus ensures a graceful degradation in the event of an overload situation.

We have set out to integrate power management in an RBED-based scheduling framework. To address the simplifications of previous work, a good model for the temporal- and energy-consumption impact at different frequency settings is required. In previous work we have developed such time [3] and energy [4] models. Our proposed approach allows for arbitrary and frequency-dependent cost of a frequency switch in the time and energy domain. Examples for such requirements are XScale-based processors or the Crusoe [5].

Based on these models we have developed a scheme which makes use of the implicit slack tracking of the RBED framework. We have implemented this work on

*NICTA is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs

a XScale-based platform within the OKL4 microkernel. In this initial paper we have kept a number of assumptions made by the original RBED approach about real-time tasks: we assume periodic, non-communicating tasks where deadlines are runnability requirements (i.e. a job must complete before the next job from that task is released) on a single processor. The relaxation of these assumptions is subject to ongoing work within our group.

The next section will cover the most relevant related work before we revisit our previous modeling work and an overview of RBED. Section 5 details our extensions to the RBED model to integrate our power management work. The implementation and our evaluation of it are the subject of Section 6 before summing up with the conclusions.

2 Related work

Strategies for dynamically scaling voltage and frequency (DVFS) to save energy is a wide research area. A number of groups have developed approaches towards both power management in real-time systems and performance prediction under frequency scaling. Within this paper we concentrate on the small representative set of these contributions which we consider most relevant to our work.

Using DVFS when considering timing constraints raises the need for precise execution-time prediction. Weissel and Bellosa [6] explored event-counter-based prediction of performance degradation as part of their development of a best-effort DVFS scheme. Monitoring certain system events allows predictions which are within a small margin of error compared with the widespread, but unrealistic, assumption of a linear relationship between core clock frequency and execution time. Refining this approach, Choi at al. [7,8] described the effects of on-chip and off-chip cycles towards performance degradation under frequency scaling. Performance monitoring counters (PMCs) and an on-line regression technique were used to calculate the balance between these two. One disadvantage of this technique was the $100\mu s$ processing time required at each frequency switch. While these papers do explicitly evaluate timing constraints, they do not target real-time systems. Ultimately they had a substantial impact on the development of our previous work [3, 4].

A number of groups [9–12] have integrated DVFS and real-time scheduling approaches. Pillai and Shin [9] simulated their *RT-DVS* algorithm for different task sets and different hardware configurations. They implemented a kernel extension for the Linux scheduler and have shown that deadline-based approaches can perform better than fixed-priority-based scheduling (since they have knowledge of the future workload).

The scheduling of sporadic tasksets was addressed by Qadi et al. [10]. They achieved significant power savings for their particular problem. This work is closely related to [9] and uses off-line techniques to determine a global level of slowdown. Both fall in the category of inter-task DVS, not exploiting the possibilities of DVS inside running jobs.

Dudani et al. [11] used system slack time to allow certain jobs to run at lower CPU core frequencies based on the two common assumptions of many DVFS papers: that execution times scales linearly with the CPU clock frequency and that running jobs slower implies saving energy. Unfortunately, simulating an ideal real-time system without taking comparing with the behaviour of a real system did not expose the issues with their proposed solution.

The real-time DVFS approaches above are based on the assumption that stretching workload maximise utilisation saves energy in general. Depending on the hardware, this assumption may be misleading (as shown by Aydin et al. [12], our previous work [4]).

Exploring the possibilities of DVFS for periodic task sets, Aydin et al. [12] have shown that premature frequency scaling can even lead to *increased* power consumption. Careful consideration of on-chip and off-chip workload has to be made to achieve considerable system wide power savings. However, similarly to the other approaches, Aydin et al. [12] evaluate their approach in idealised simulations rather than in a physical system environment.

The effects of frequency switching overhead which represent significant, un-interruptible sections, were largely ignored in recent work. However, our paper shows how these side effects can be managed in real-world real-time systems.

3 Prediction of Time and Energy

In our previous work [3, 4] we have developed an accurate method to model time and energy consumption under DVFS. This is now deployed in the implementation of the DVFS-RBED policy. Here, we briefly introduce the relevant concepts. Further detail is provided in the original publications [3, 4].

3.1 Time

The execution time for a given piece of software can be described as the sum of the times spent waiting on different functional units of the system. This can be time in the CPU core actually executing instructions, time spent waiting for main memory, time spent waiting for I/O operations to complete and so on.

All of these operations can be considered as scaling inverse proportionally with their respective clock frequencies. This allows the execution time C to be expressed as follows:

$$C = \frac{c_{cpu}}{f_{cpu}} + \frac{c_{bus}}{f_{bus}} + \frac{c_{mem}}{f_{mem}} + \frac{c_{io}}{f_{io}} + \dots \quad (1)$$

The coefficients c_x can be interpreted as being caused by a number of events combined with an number of wait-states associated with each of these events. Ideally there would be a way to observe the events directly. However, many CPU cores provide us with a means to observe some events which are correlated with the events in question. Depending on the architecture chosen, these have different names; e.g. *event ocurrence counter* for the PowerPC or *performance event counter* in AMD chips. In Intel chips these are usually called *performance monitoring counters* (PMCs) and we are going to use this term for the remainder of the paper. The available events which can be observed varies widely between architectures, but usually include beside many others *good* predictors like cache misses, TLB misses, or write backs.

Within this paper we focus on events concerning memory accesses, which in this case involve the bus and the memory frequency. In our sample platform the I/O is not subjected to a separate frequency, hence we will concentrate will only use c_{cpu}, c_{bus}, and c_{mem} respectively. Depending on the number of appropriate PMCs available (in this case, 2), the equation can be linearly extended.

$$\begin{aligned} c_{bus} &= \alpha_1 PMC_1 + \alpha_2 PMC_2 + \dots \\ c_{mem} &= \beta_1 PMC_1 + \beta_2 PMC_2 + \dots \end{aligned} \quad (2)$$

The PMC readings are application specific, while the coefficients α_i and β_i are architecture specific. Therefore, a given hardware platform simply needs to be calibrated.

Most architectures provide a cycle counter. This expresses the execution time C of an application in terms of the cycles c_{tot} of the CPU core frequency f_{cpu}. As such the final parameter c_{cpu} can be computed based on a single measurement:

$$c_{cpu} = c_{tot} - \frac{f_{cpu}}{f_{bus}} c_{bus} - \frac{f_{cpu}}{f_{mem}} c_{mem} \quad (3)$$

Now that all parameters of the model are instantiated, we can, from a measured part of the application, reason about the progress the application would make at the fastest frequency setpoint and the time required to execute the remainder of the task in any given frequency. This will be further explained in Section 5.2.

3.2 Energy

The energy model follows a similar logic to the time model: the energy consumed by the system depends on the properties of the workload. It depends largely on the number and type of operations performed, as well as the static power over a given time interval Δt.

$$\begin{aligned} E = {} & V_{cpu}^2(\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem})\Delta t + \\ & V_{cpu}^2(\alpha_0 PMC_0 + \dots + \alpha_m PMC_m) + \\ & \beta_0 PMC_0 + \dots + \beta_m PMC_m + \\ & \gamma_4 f_{mem}\Delta t + P_{static}\Delta t, \end{aligned} \quad (4)$$

Equation 4 consists of five groups of terms.

1. The first term accounts for the constant event rate owing to the CPU, memory and bus clocks within the CPU (in our test system, the memory controller and processor bus are on-die). The number of cycles executed is proportional to the time. These are proportional to the the square of the CPU core voltage.

2. Similarly, the next term associates certain events described by performance monitoring counters PMC_i with energy consumed within the chip (i.e. proportional to V_{cpu}^2). For these, only the event count, rather than the frequency are relevant.

3. The PMCs are also used to describe off chip events, like memory accesses, for which the CPU core voltage scaling has no impact.

4. The memory bus frequency is seen external to the CPU, and is therefore accounted for by a term which is independent from the CPU core voltage.

5. Finally the static power constitutes all of the power not effected by frequency or workload changes (and the energy is therefore proportional to the time spent executing).

The performance monitoring counter values PMC_i have to be a measure of the events during the time interval Δt. In turn the time interval at some target frequency can be expressed using Equation 1. For clarity of the presentation, we have not combined the equations as the result would become unwieldily.

While Equation 4 is presented with a single scalable voltage domain (the CPU core voltage) and a constant voltage domain, it is trivial to extend to multiple scalable voltage domains. Currently, the model does not explicitly take the effects of I/O events into account. The task model assumes that blocking on I/O will invoke the scheduler which initiates a switch to a different task. In this case, I/O is not accounted to the job which actually initiates the I/O operation, but the one executing during the I/O operation. However, as the energy model is determined offline this effect is not taken into account. The effects of I/O on the model are subject of ongoing research in our group. Further details on performance counter selection, etc, can be found in our prior work.

4 RBED Summary

Since our work is based on the RBED scheduling framework developed by Brandt et al. [1] we will briefly introduce the the background and concept of the RBED approach. Devices with mixed timing requirements represent the majority of today's embedded systems. Many of them allow the installation of arbitrary user software which may have unknown timing behaviour. Demanding precise worst-case execution-times at installation time is not feasible. Therefore, other precautions must be taken to ensure that the behaviour of any given application cannot harm the provision of timing requirements of other programs. The misbehaviour under overload conditions is one of the major shortcomings of classic *earliest deadline first* (EDF) scheduling. Brandt et al. developed a multi-class real-time scheduler for the seamless support of mixed hard, soft and non real-time applications. It ensures a timely separation of tasks by preemptability and a resource allocating governor. Resource allocation takes place at run-time where tasks dynamically request a share of the CPU time. The resource allocator can be implemented as a user space application that performs an on-line schedulability analysis. The preemptive scheduler implements EDF but ensures that only resources granted by the resource allocator are used.

One advantage of this approach is that allocation of resources for soft real-time tasks can be independent of the actual WCET of its jobs. Thus, for a soft real-time task, where an application may miss (a small number of) deadlines, instead of using the worst-case execution time, a smaller timeslice may be allocated. This allows over-allocation of the system while keeping single applications in temporal isolation; i.e. one soft real-time job exceeding its allocated resources has no impact on the schedule of other tasks. This allows seamless integration of hard real-time, soft real-time and best-effort tasks in one system with a unified scheduling policy.

U	utilisation of the entire taskset, $U = \sum_{\forall i} u_i$
u_i	utilisation of a given task τ_i, $u_i = E_i/T_i$ at the top frequency setpoint
$r_{i,n}$	release time of a given job $J_{i,n}$
$d_{i,n}$	absolute deadline of a given job $J_{i,n}$
$x_{i,n}$	current service time $u_i(t-d_{i,n-1})$ of a given job $J_{i,n}$
C_i	WCET of a given task τ_i at the top frequency setpoint
E_i	budget allocated to task τ_i
D_i	relative deadline of task τ_i
T_i	period/minimal inter arrival time of task τ_i
C_i^*	part of current job of task τ_i completed (equivalent at top speed)

Figure 1: Nomenclature Used

For a complete description of the algorithms including the proof of correctness, we refer to Brandt's original work [1]. For the relevant nomenclature see Figure 1.

Given the feasibility of dynamic resource dispatching, the introduction of per-job budgets allows for a trivial measurement of resource usage and on-line reallocation of slack time. The task model is illustrated in Figure 2 and can be described as follows:

A task τ_i consists of multiple subsequent jobs $J_{i,r}$. These jobs cannot overlap ($r_{i,r+1} \geq r_{i,r} + T_i$).

Every job is preemptible at any time and the jobs of a task have a minimum inter-arrival time (IAT) which is called the period in the case of periodic tasks. No more than one job can be released within the period/IAT. Each job has a deadline relative to its release time. For the scope of this paper, the relative deadline equal to the period of the job. ($D_i = T_i$)

Each job has a worst case execution time (WCET) C_i which can be obtained using well-known techniques and is very likely to be larger than the actual execution time $x_{i,r}$. As part of the RBED scheduling, a budget E_i is reserved to each job $J_{i,r}$ representing the timeslice that must be granted to the job by the scheduler. In case

of hard real-time requirements, the budget equals the WCET ($E_i = C_i$) to guarantee timely completion of all jobs. For soft real-time requirements, the assigned budget may be smaller than the WCET $E_i \leq C_i$ to guarantee timely completion of all jobs $\{J_{i,r} | x_{i,r} \leq E_i\}$.

In most cases, the job will finish in less time than reserved ($x_{i,r} \leq E_i$). The remaining budget $E_i - x_i$ is reserved, but not used and is therefore called slack $S_{i,r}$.

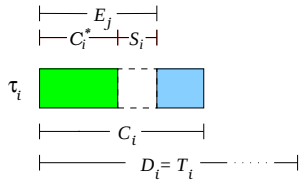

Figure 2: Parameters of Job J_j

If a per-job budget E_i assigned to a task τ_i is smaller than the WCET of a single job $J_{i,j}$ of τ_i, the job's runtime can potentially exceed the budget. Since this is an overload situation and would likely have drastic consequences in classic EDF scheduling, precautions are taken in RBED. The scheduler preempts every job when it has used up its budget E_j and extends the deadline D_j of the job by one period T_j. At the same time, the job's budget E_j is refilled to the assigned value. Thus, the remainder of the execution is postponed until it is scheduled again.

This guarantees a temporal isolation among all tasks in the system as long as $U = \sum_{\forall i} u_i \leq 1$ with $u_i = E_i/T_i$. To ensure this condition is met, the resource allocator (RA) which may be an ordinary user space task must acknowledge all requests for changing job's periods, budgets or deadlines. The idea of dynamic reallocation of processing time is based on five theorems.

Theorem 1 The earliest deadline first (EDF) algorithm will determine a feasible schedule if $U \leq 1$ under the assumption $D_i = T_i$.

Theorem 2 Given a feasible EDF schedule, at any time a task τ_i may increase its utilisation u_i by an amount up to $1 - U$ without causing any task to miss deadlines in the resulting EDF schedule.

Theorem 3 Given a feasible EDF schedule, at any time a task τ_i may increase its period without causing any task to miss deadlines in the resulting EDF schedule.

Theorem 4 Given a feasible EDF schedule, if at time t task τ_i decreases its utilisation to $u_i' = u_i - \Delta$ such that $\Delta \leq x_{i,n}/(t - r_{i,n})$, the freed utilisation Δ is available to other tasks and the schedule remains feasible.

Theorem 5 Given a feasible EDF schedule, if a currently released job $J_{i,n}$ has negative lag at time t (the task is over-allocated), it may shorten its current deadline to at most x_i/u_i and the resulting EDF schedule remains feasible.

The resource allocator algorithm can be described as follows: U_{Kernel} describes the worst case utilisation required by the operating system. $U_{BE,min}$ describes the minimum reserved utilisation reserved for all best-effort tasks.

1. Assign desired utilisation $U_{HRT,i}$ to all hard real-time (HRT) tasks as long as $U_{HRT} \leq 1 - U_{Kernel} - U_{BE,min}$ where $U_{HRT} = \sum_{\forall i_{HRT}} u_{i_{HRT}}$ Reject all other requests for HRT resources.

2. Distribute utilisation not reserved for hard real-time tasks, best-effort tasks or the operating system among the soft real-time tasks according to their requested resources. In case $U_{SRT} = 1 - U_{Kernel} - U_{BE,min} - U_{HRT} < \sum_{\forall i_{SRT}} u_{i,desired}$ each SRT task is assigned $u_i = u_{i,desired} / \sum_{\forall j_{SRT}} u_{j,desired}$

3. The total utilisation reserved for best-effort tasks is the remaining utilisation which can be described as $U_{BE} = 1 - U_{Kernel} - U_{HRT} - U_{SRT} \geq U_{BE,min}$. U_{BE} is equally distributed among all best-effort tasks.

5 Model Extension

In order to integrate our DVFS work with the RBED approach we need to extend the RBED model. Our task model is based on the idea that each job can be slowed down, if this has beneficial effects for the total energy consumed by the system. Jobs are preemptible and after each preemption the frequency is re-evaluated.

One major issue when performing frequency scaling on real-world architectures is the lengthy time sometimes required to switch voltage and frequency. A frequency switch can be modelled as a substantial atomic section. At design time, this must be accounted for appropriately.

The following section describes how this model can be integrated with the RBED scheduling algorithm.

5.1 Budget for Switching

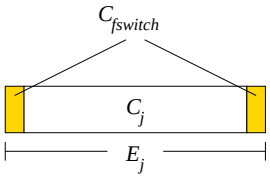

Figure 3: Budget Components

One of the fundamental advantages of RBED scheduling is the absence of a complex schedulability test which requires a priori knowledge of all parameters of the taskset. Allocating time budgets isolates tasks from one another. To guarantee this isolation, it is necessary to ensure that no job can use more resources than allocated *a priori*.

Figure 3 shows, how a time budget must be allocated in order to guarantee isolation within our DVFS framework. The first frequency switch that needs to be considered is the initial one, which may be required if the previous job was scaled to a frequency lower than the minimum frequency necessary to guarantee timeliness.

Each job $J_{j,r}$ can potentially preempt another job $J_{i,s}$ when it becomes un-blocked due to an interrupt if $D_j < D_i$. At preemption time, the scheduler determines the energy optimal frequency setting for job $J_{j,r}$ which may differ from the optimal set point for job $J_{i,s}$. In the case where a frequency switch is performed, sufficient time must be allowed such that the preempted job $J_{i,s}$ is able to restore its frequency to guarantee timely completion. Our approach is the automatic donation the time required for one frequency switch to the preempted job. A second frequency switch must be accounted for in each task's budget. This policy is illustrated in Figure 4 a) and b). These depict the point in time of the preemption and the subsequent donation of the switching cost for the second frequency switch of job $J_{j,r}$ to job $J_{i,s}$. The dashed boxes indicate as-yet unused time budget.

When a job becomes ready and the system is in its idle state, no automatic donation need to be performed since the idle task does not need to restore its frequency set point. In this case, the job can use up the additional budget for further slowdown if beneficial for the system's total energy use.

The key point of this section is this: within a job, an arbitrary number of frequency changes can be performed as long as the budget accounts for two frequency changes.

Technically, it would be sufficient to perform the automatic donation only if a frequency switch actually takes place. However, this requires an unreasonable amount of house keeping without tangible benefit.

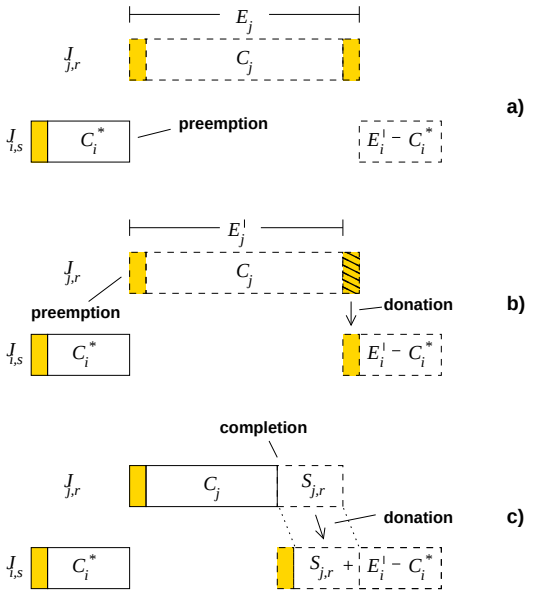

Figure 4: DVFS-RBED Preemption Behaviour

5.2 Dynamic Slack Management

On top of the RBED scheduling algorithm, Lin and Brandt [2] have developed a set of slack reclaiming mechanisms which enhance the temporal behaviour of soft real-time applications in RBED: SRAND and SLAD are consistent with our approach as they allocate slack to the highest priority task as soon as possible. SLASH which attempts to borrow slack from future jobs of the same task and BACKSLASH, which allocates slack to past jobs which have not completed within their budget, could be integrated with our work without conflict.

The task model of RBED allows implicit knowledge of slack time in the schedule. Slack $S_{i,r}$ generated by job $J_{i,r}$ represents the amount of scheduled but unused processing time. This slack time can be pushed forward to the next runnable task's budget in the schedule without harming the timely behaviour of any task in the task set. Such a donation of slack is depicted in Figure 4 c). A formal proof can be found in [2]. It should be noted, however, that this implies that slack is only donated to jobs with a later deadline. When extending the work to communicating and blocking tasks, this condition needs

to be revisited and ascertained.

Since the amount of budget allocated for each job should be sufficient for the job to complete within the given budget of the job, additional budget allows longer processing than required.

Reducing the clock frequencies (f_{cpu}, f_{mem}, f_{bus}) can reduce the total energy consumption for the job's execution. To determine feasible frequency setpoints, the time model in Section 3.1 is used. The set of feasible setpoints $\{\sigma = \{f_{cpu}, f_{mem}, f_{bus}\} | C^{\sigma}_{i,r} \leq E_{i,r}\}$ for job J_i, r is then investigated for their potential effects on the job's energy consumption using the model described in Section 3.2. If a slower execution saves energy, a frequency switch is performed.

The algorithm illustrated in Figure 5 shows the actions to be taken as part of each scheduler invocation. First, all frequency set points are tested for feasibility. Therefore, the time for potential frequency switches is taken into account. Note, the time needed for a frequency switch is not necessarily constant. It is zero in case the frequency is unchanged and may be substantial for other transitions. We assume, changing from set point σ_A to σ_B is a constant, but the transition from σ_A to σ_C may take a different amount of time. A real world example for this behaviour is the Crusoe processor [5]. Xscale processors like the PXA270 or PXA555 have certain frequency combinations which where the transition is almost instantaneous (turbo mode changes) and others which require a substantial amount of time.

Second, the energy consumption for the job at all feasible set points is investigated. The set point which leads to the lowest energy consumption is then chosen.

After returning from a preemption, the job may have received a donation of slack time if the preempting job has not used up its entire reserved budget. Therefore, the energy optimal frequency is recalculated on each scheduler invocation.

One related issue is the absence of a precise measure for the progress of a job. Thus, the progress is estimated using the same time model (Section 3.1). Knowing the frequency set point and the number of events which have occurred during the job's execution so far, the time model can be used to determine the remaining processing time. Figure 6 illustrates this idea. The events depicted represent PMC events. The number of these events is in reality large, but has been limited for illustrative purposes.

A job of task τ_i at maximum frequency σ_{max} is depicted in Figure 6 a) the height of the boxes indicates the power consumption and thus relates to the frequency setpoint. In the example, five events $\epsilon_{1..5}$ happen during execution of this job. Figure 6 b) shows the execution

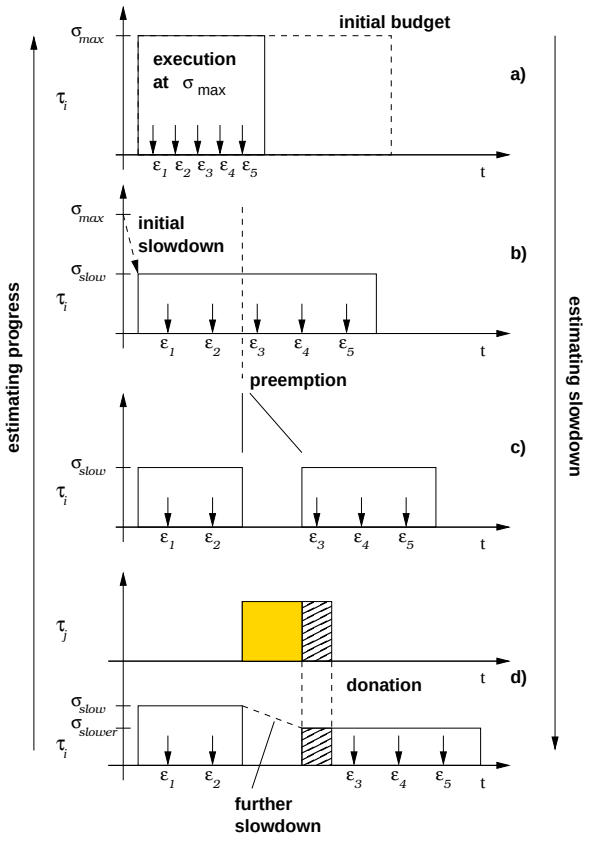

Figure 6: DVFS-RBED Progress estimation and dynamic slowdown

of the same job at a lower frequency set point σ_x. The number of events remains five. The preemption of the job is depicted in Figure 6 c). This preemption changes neither the total execution time of the job, nor the number of events during execution. Figure 6 d) shows the actual execution of task τ_j preempting task τ_i. While the number of events remains five a further slowdown is possible due to a donation of slack from task τ_j.

To perform a further slowdown after a preemption and donation, the remaining runtime of the job $J_{i,r}$ must be determined. A measure for progress is needed. Since there is no absolute measure for an application's progress the scheduler could determine, the estimation technique in 3.1 is performed in the reverse order. The job's execution time so far and number of events can be mapped to an equivalent progress at σ_{max}. We call this the absolute progress $C^{*}_{i,r}(t)$ of job $J_{i,r}$ at time t.

The remaining execution time at any frequency can now be determined in the reverse order, using the difference between progress and WCET ($C_i - C^{*}_{i,r}(t)$) and the time model 3.1.

```
newEnergy = energyAtCurrentFrequency
newFrequency = currentFrequency
for frequency in frequencySetPoints
    if executionTimeAtSwitchedFrequency + switchingCost.Time < remainingBudget
            && switchingCost.Energy + energyAtSwitchedFrequency < newEnergy
        newEnergy = switchingCost.Energy + energyAtSwitchedFrequency;
        newFrequency = frequency
if newFrequency != currentFrequency
    switchFrequency(newFrequency)
```

Figure 5: Algorithm

In cases for which the performance degradation can not be estimated safely, the algorithm may use a look-up table to determine the WCET C_i of hard real-time tasks. In such a scenario progress estimation can not be performed safely either, thus a further slowdown is not possible for hard real-time jobs.

Since our mechanism is based on the fact that slack time is released at the end of a job, slack time can only be re-allocated if a job finishes execution. In the case of sporadic tasks, no jobs may be executed for a long period of time. This slack time is currently not taken into account in our algorithm and would be assigned to best-effort tasks (if available) or the idle task.

5.3 Static Slack Management

To gain maximum possible energy savings, static slack time must be distributed entirely. Static slack time describes the amount of time in the schedule which is not allocated. In other words, it describes the sum of all idle times in the schedule.

Static slack which is not allocated to any task cannot be reclaimed or used for power management purposes using our proposed algorithm. We found two different options, to ensure full utilisation of the system. The reader may recall $u_i = E_i/T_i$ where E_i does not necessarily describe the true execution time but the reserved time slice.

One solution is a distribution of static slack among all existing tasks in the system which is later forwarded as dynamic slack since jobs will likely complete earlier than their budget expires. Dynamic slack is then used for optimal frequency scaling. The advantage of this approach is that all tasks benefit from the slack evenly likely leading to a decreased energy consumption.

The other solution is the introduction of a *ghost* task. This task's sole purpose is freeing up its own budget. This approach has the benefit of keeping track of static slack explicitly. This enables easier exploitation of this slack in a dynamic situation where tasks are added at runtime.

6 Implementation Issues and Lessons Learned

6.1 Experimental platform

We implemented the proposed algorithm on an off-the-shelf *Gumstix Connex* platform which runs the L4 microkernel *OKL4 v1.5.2* which was the latest release at the time of implementation and the *Iguana* operating system [13].

In particular, we wrote three software modules implementing the proposed algorithms. First is the preemptive EDF scheduler which replaces the fixed priority scheduling algorithm of the L4 microkernel. Priority was replaced in the task control block by deadlines, budget and period. The scheduler preempts running jobs when the job's budget is used up. If a job completes before its budget is entirely used, the remaining budget which represents the generated slack is enqueued in a deadline sorted budget queue.

Second, the user space resource allocation was integrated in the iguana root task which has special privileges. One privilege added is the exclusive right to perform system calls to the scheduler. This ensures that no other task can make its way around the mechanism. Deadlines, budgets and periods can be allocated at build time or dynamically changed on runtime if the resource allocator permits the change.

Third, a module for arithmetic evaluation of the time model 3.1 and the energy model 3.2 was added to the kernel space scheduler. It evaluates the execution-time estimation for all possible frequency settings and then calculates the energy consumption at all feasible set points. Finally it chooses the set point related to the lowest energy consumption.

Figure 7 shows the software architecture for our implementation which is partially similar to the RBED implementation we received by courtesy of Brandt et al.

The hardware platform consists of a *Gumstix Connex* motherboard, an *Etherstix* network interface as well as an *Audiostix 2* sound card and a *Tweener* serial console

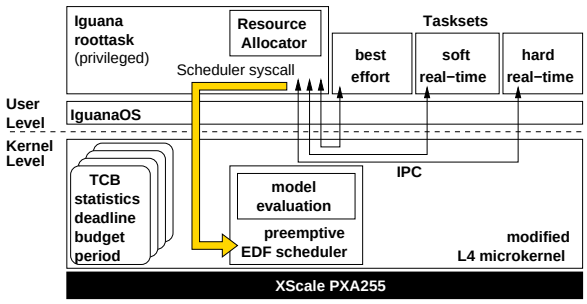

Figure 7: DVFS-RBED Software Architecture

driver. We have chosen this configuration to generate a typical static and dynamic power consumption of an embedded device. The *XScale PXA255* CPU frequency as well as the bus frequency and the memory frequency can enter 22 different setpoints in this configuration. The *Gumstix* platform does not support voltage scaling.

All power consumption was measured with a power- and energy-measurement device developed within the group. This device obtained an accuracy of better than a Milliwatt. The static power consumption of this hardware configuration in idle mode was measured to be $1.13W$. The network interface was measured to consume the majority of this ($0.8W$). While active, at the lowest frequency set point, the power consumption of the device is $1.45W$ and $1.57W$ at the fastest.

The CPU deployed in our experiments implements two types of frequency switches. The standard switch changes frequencies of CPU, bus and memory to one of the 22 different modes. These changes take a substantial time between $500\mu s$ and $600\mu s$. This time is caused by a combination of required operations, like putting the memory in self refresh mode, and the settling time of the phase-locked loop circuitry. The other switching operation implemented in the *PXA255* is called "turbo" mode switching. It allows fast switching between two different CPU frequencies inside frequency pairs, maintaining the memory and bus frequencies. Those turbo frequency changes are intended for peak processing requirements and happen synchronously without disrupting the memory controller or any peripherals. These switches require only a small number of nanoseconds.

Furthermore, the *PXA255* CPU implements a number of low power states which may be useful depending on the maximum acceptable interrupt latency. These low power states also range from single cycle clock gating to extremely deep sleep states which requiring a delay on the order of one millisecond to exit.

6.2 Best effort threads

One of the key points in the RBED scheduling algorithm is the seamless integration of hard real-time, soft real-time as well as best effort tasks. Best effort tasks can be described as tasks which do not necessarily have a periodic or frequent blocking point and do not raise any real-time requirements. Nevertheless, the goal is a guaranteed continuous progress regarding their execution even under high system load to maintain the system in a responsive state.

To keep the scheduling algorithm simple, periods, budgets and deadlines are assigned to all best effort tasks. Thus, these tasks are treated as real-time tasks with artificial deadlines. Depending on the system parameters and the priority of power savings over best-effort performance restricting the execution of best-effort tasks to a small share can save energy. If the execution of best-effort tasks is not restricted, the system may never go to low-power idle mode.

6.3 Lessons learned

Calculating the energy purely needed by one particular job shows, that faster execution saves energy because the execution time decreases faster than the power consumption increases for our particular platform. However, we believe there is a trend in modern processors, with static power taking an increasing share of the overall system power consumption. Since our experimental application is a device which is constantly turned on, the energy required by the device to stay turned on must be taken into account. Due to the high static power consumption of the device and compared to the low dynamic power consumption, the impact of the idle time is substantial. For our particular experimental platform it turned out, that the lowest possible frequency setpoint is always the most energy saving.

The algorithm in Figure 5 which evaluates the time model and the energy model must be performed in fixed point arithmetic because floating point processing would cause too much overhead in general and on the *XScale* architecture in particular, since it does not implement a floating point unit.

Another lesson learned is that a microkernel requires extremely careful implementation of this approach. The first microkernels developed gave the kernel design a bad name because of the large number of context switches are necessary and which expose deficiencies in context switching costs as poor system performance. Years of research and development have lead to the L4 microkernel which was designed for very high *interprocess communication* (IPC) performance. This ap-

proach was successful and L4-based kernels are widely deployed in modern embedded systems. While the success of current L4 microkernels is based on fast context switches and IPC. This advantage is threatened by the overhead of Figure 5 on each context switch.

Our choice, the *PXA255* processor was based on its ability to change between run mode frequency and turbo mode frequency without a substantial switching time. However, the *PXA270* processor implements a half-turbo mode. This processor would have been the better choice in hindsight. Finally, the choice of the *Gumstix* platform results in the inability to use voltage scaling. Deploying voltage scaling would lead to better energy savings, but may add a switching overhead.

7 Conclusion

Within this paper we have shown the integration of real-world power management with a real-time scheduling approach and have reported on the lessons learned from this work.

In the future, we will expand this work beyond the scope of this paper. Core issue in this area is the extension to communicating task sets. The modeling of systems with non rate-based applications (e.g. bursty workloads), is another requirement for real-world deployment. We will investigate the modeling and integration of these tasks within our framework. Furthermore we want to study the effects of deadlines which are shorter than the period on the RBED algorithm in general and on our DVFS extension in particular.

Modelling sporadic tasks as periodic tasks might not be the energy optimal solution. Further investigation is necessary when completely unused reserved time of sporadic tasks can be freed for power management use. As mentioned above, our task model assumes that I/O completions start a job and another blocking operation marks the end of a job. Depending on the hardware, this model may not be sufficiently general. Further investigation of the effects of intra-job blocking on our model remains future work.

8 Acknowledgements

We would like to thank Scott Brandt, Suresh Iyer and Jaeheon Yi for allowing us access to their RBED implementation and documentation which we used to guide our own implementation.

References

[1] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson, "Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes," in *24th RTSS*, (Cancun, Mexico), Dec 2003.

[2] C. Lin and S. A. Brandt, "Improving soft real-time performance through better slack management," in *26th RTSS*, (Miami, FL, USA), Dec 2005.

[3] D. C. Snowdon, G. van der Linden, S. M. Petters, and G. Heiser, "Accurate run-time prediction of performance degradation under frequency scaling," in *3rd OSPERT*, (Pisa, Italy), Jul 2007.

[4] D. C. Snowdon, S. M. Petters, and G. Heiser, "Accurate On-line Prediction of Processor and Memory Energy Usage Under Voltage Scaling," in *7th Int. Conf. Emb. Softw.*, (Salzburg, Austria), Oct 2007.

[5] M. Fleischmann, "Microprocessor architectures for the mobile internet era," Apr 2002. http://www.marcfleischmann.com/talks/arcs2002.pdf.

[6] A. Weissel and F. Bellosa, "Process cruise control—event-driven clock scaling for dynamic power management," in *CASES*, (Grenoble, France), Oct 8–11 2002.

[7] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *Int. Symp. Low Power Electron. & Design*, pp. 174–179, Aug 2004.

[8] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times," *Trans. CAD ICAS*, vol. 24, pp. 18–28, Jan 2005.

[9] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *18th SOSP*, (Lake Louise, Alta, Canada), pp. 89–102, Oct 2001.

[10] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," in *24th RTSS*, (Washington, DC, USA), p. 52, Comp. Soc. Press, 2003.

[11] A. Dudani, F. Mueller, and Y. Zhu, "Energy-conserving feedback EDF scheduling for embedded systems with real-time constraints," in *LCTES'02*, (Berlin, Germany), pp. 213–222, ACM, 2002.

[12] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *27th RTSS*, (Rio de Janeiro, Brazil), pp. 313–322, Comp. Soc. Press, Dec 2006.

[13] OK-Labs, *OKL4 Website*. URL http://www.ok-labs.com/products/okl4.