# Slow Down or Race to Halt: Towards Managing Complexity of Real-Time Energy Management Decisions *

**Stefan M. Petters** **Muhammad Ali Awan**

[1]CISTER
ISEP-IPP
Porto – Portugal

{smp,maan}@isep.ipp.pt

***Abstract.*** *Existing work in the context of energy management for real-time systems often ignores the substantial cost of making DVFS and sleep state decisions in terms of time and energy and/or assume very simple models. Within this paper we attempt to explore the parameter space for such decisions and possible constraints faced.*

## 1. Introduction

Currently we observe a number of trends in the way embedded systems and in particular embedded real-time systems are deployed. Firstly, such systems have become ubiquitous and we become increasingly dependent on them. The computing capability of the processing cores is increasing at a dramatic pace leading to the change towards multi-functional and multi-criticality devices, where hard real-time and best-effort tasks share resources. Finally, while traditionally embedded systems were isolated single purpose devices, they have become often networked and/or mobile.

In particular mobile devices have inherently limited energy supply in terms of batteries. Also tightly packed multicore systems have increasingly thermal issues. In the past we have developed a method to accurately predict time and energy consumption under dynamic frequency and voltage scaling (DFVS) based on online measurements [Snowdon et al. 2007, Lawitzky et al. 2008, Snowdon et al. 2009]. However, a fundamental challenge is the substantial search space for online decision making. For example, our implementation [Lawitzky et al. 2008] of using DVFS in a RBED [Brandt et al. 2003] required 14 multiplications and eight additions per frequency set-point, leading for 22 set points available in our XScale [XScale 2004] platform to a maximum observed of 26,000 cycles. Our observations also indicated frequency and voltage scaling times of 500 to 600 $\mu$s.

On the other hand it has been shown that models used in theoretical DVFS work on real-time systems are substantially off the mark. This covers the following assumptions: The power model, where the power consumption is solely dependent on frequency and voltage as given in Equation 1 does not take application dependencies into account.

$$P \propto fV^2 \tag{1}$$
$$C = c * f \tag{2}$$

*This work was supported by CISTER FCT-608 and CooperatES QoS-Aware Cooperative Embedded Systems PTDC/EIA/71624/2006

A constant number of cycles required to execute a task as provided in Equation 2 does not take into account the number of wait states on external devices like main memory, which changes when the core frequency is changed. Frequency switch costs are frequently considered negligible in terms of time and energy, which it is clearly not. Often frequency and voltage is considered continuously scalable, while it in fact only changeable in discrete steps. In particular frequency is often only coarsely scalable for higher frequency set-points, which lower frequency set points can be site on a finer granularity. Individual issues have been tackled in mostly academic work, but a comprehensive approach taking all these factors into account is outstanding. For example, [Aydin et al. 2006] used a non constant number of execution cycles by considering an off-chip and an on-chip component of the execution time, but did not consider the overhead of transitioning into a different frequency set point and using an evaluation by simulation the problem is not exposed. [Cheng and Goddard 2005] used discrete frequencies, but a application independent power consumption, as well as assuming negligible voltage and frequency switch cost.

The problem is acerbated when sleep states are considered. The number of different sleep states available is processor dependent. This reaches from a simple clock gating, where the CPU core is separated from the system clock, to deep sleep states, which power off substantial parts of the chip. The former is available instantaneous (within a CPU core clock cycle), while the latter requires substantial time and energy to get into and out off.

Within this work we will identify all the parameters to be considered to make energy efficient frequency decisions in a real-time context, as well as looking into ways to reduce the search space for such decisions. In the following section we will introduce our system model before discussing implications for DVFS and sleep state decision taking on a static basis and taking knowledge about system dynamic slack into account.

## 2. System Model

In terms of power management models we draw on our previous work [Snowdon et al. 2007], with a few generalizations. While it is not essential for the discussion to use these models, they give an overall sense of the complexity of the task faced.

The worst-case execution time (WCET) of a task under DVFS is subject to various components. Some time is spent actively performing computations in the CPU core, some time is spent waiting for access to a bus, another part is contributed by the response time of memory, which can be changed by modifying the memory frequency, some time is spent waiting for I/O devices and so forth. Assuming all of these are subject to individual frequency settings, we get an overall execution time $C$ being a function of various application dependent coefficients $C_{coeff}$.

$$C = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{bus}}{f_{bus}} + \frac{C_{mem}}{f_{mem}} + \ldots \qquad (3)$$

In the original work, we have identified the coefficients using measurements of certain hardware events, but for generality sake we limit ourselves to identify the makeup of the base equations. Throughout the paper we assume $C_i$ to be the execution time at top speed in the understanding that for DVFS considerations the actual $C(\mathbf{f})$ of the frequency setpoint $\mathbf{f}$ has to be considered.

The energy consumed by a task on a given execution path is also subject to various components as identified in Equation 4. The first component is the energy spent in the CPU core with the core voltage squared. In relation to the XScale processor the circuitry in the CPU core is switched with 3 different frequencies and essentially covers the switching cost of the clock circuitry. The execution time dependence is encoded in the multiplication of $C$, but beyond that the coefficients $\gamma_x$ are only architecture and not application dependent. Certain operations are dependent on the core voltage, but not on the CPU core frequency. An example is a multiplication function, where the same number of transistors switches irrespective of the core frequency.

Certain other operations are independent of the core voltage and frequency. An example could be a DMA packet transfer to the network interface, which will have a memory frequency dependent and a memory frequency independent part. The memory frequency independent part might still be subject to a specific frequency, for example internal to the network interface, which might not be scalable. Similar the voltage of the memory would be part of the coefficients. In Equation 4 this is encoded in the parameters $\alpha$ and $\beta$ respectively. As final components are the static power consumption $P_{static}$ and the memory is subject to switching over the time $C$. Again the voltage of the memory is considered immutable and is part of the coefficient $\gamma_4$.

$$E = V_{cpu}^2(\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem}) * C + V_{cpu}^2 \alpha + \beta + \gamma_4 f_{mem} C + P_{static} C \qquad (4)$$

We have shown [Snowdon et al. 2007, Lawitzky et al. 2008] how the parameters can be obtained runtime. In later work [Snowdon et al. 2009] we have demonstrated an approximation scheme for non RT systems, but this is concentrating on managing the performance to battery life trade-off with no consideration for RT systems. We assume a number of sleep states, which have monotonically increased power reduction, as well as monotonically increased time required to enter $t^s$ and leave $t^w$ the respective state.

Beyond the model for the power management aspects we also assume a sporadic task model with $x$ independent tasks in which the minimum inter-arrival time $T_i$ of a task $\tau_i$ is known. We assume form of temporal isolation, which may be achieved using constant bandwidth servers (CBS) [Abeni and Buttazzo 1998] or RBED [Brandt et al. 2003]. Slack is caused by the difference in worst-case assumption used during analysis and actual behavior at run-time. In later section we assume slack management as an important tool. It identifies the room to maneuver for power management in rate based real-time environments [Lin and Brandt 2005, Lawitzky et al. 2008]. For ease of representation we also assume implicit deadline model, where a job has to be completed before a new release is initiated.

## 3. Static Decision Taking

The least costly decision base is to statically assign frequencies and/or sleep states per task to be performed at start or on completion of the execution respectively. Generally speaking we have to consider the following scenarios: In a *race to halt* solution tasks are executed as fast as possible and the CPU is sent to a sleep state on conclusion of a job of the task. Alternatively a task may be subjected to DVFS to consume less execution time.

In a first step we rule out sleep states which lead to a potential violation of a deadline even when considering only a single task. This can be motivated by a job release of a task being triggered just after a transition into a given sleep state has been initiated. Once initiated the transition into the sleep state has to be completed, before a wakeup is triggered by the ISR corresponding to the task release. This implies a blocking time of up to $t_n^s + t_n^w$ for sleep state $n$. Since it does not matter which task has initiated the sleep state transition we can used the rule expressed in Equation 5 to express this. Any sleep state violating Equation 5 may not be used, as it may cause a deadline miss regardless of scheduling algorithm or schedulability analysis used.

$$\min((T_1 - C_1), \dots, (T_x - C_x)) \leq t_n^s + t_n^w \tag{5}$$

In a second step we examine whether it is economical to use such a measure. One observation in this context is that all tasks in the system have to follow the same rule, as to which sleep states are economical. This is caused by the general assumption that while one task initiates a sleep state, the processor may be transitioned into the waking state by any other task as we do not assume in this section any knowledge of past releases of any given task.

As such we compute the average expected idle time $\bar{G}$ produced by the system. Let $l$ denote the hyper period and $r_i$ represents release time per task $\tau_i$ as given in equations Equation 6, Equation 7 respectively. During a hyper period $l$ of the average-case inter-arrival time's $\bar{T}_j$ of all tasks we have a total of $r_i$ releases per task $\tau_i$. We also need to compute how many releases of tasks will happen while another task is executing, as this means that the previous job completion will not lead to a transition into a sleep state.

$$l = \mathtt{lcm}(\bar{T}_1, \dots, \bar{T}_x) \tag{6}$$

$$p_i = \frac{\sum_{\forall j \in \{0,\dots,x\} \setminus i} r_j * \bar{C}_j}{l} \tag{8}$$

$$r_i = \frac{l}{\bar{T}_i} \tag{7}$$

$$r_i^* = (1 - p_i) * r_i \tag{9}$$

As such we expect $r_i^*$ to be the number of releases of $\tau_i$ which happen during idle intervals. Equation 9 determines the $r_i^*$. Where $p_i$ given by Equation 8 compute the probability of one release of $\tau_i$ happening while any other task is executing.. The number of idle time task releases $r_i^*$ will be a real number. However, for the average idle time computation this is of no concern. Now $\bar{G}$ can be computed by Equation 10 where $\bar{C}_i$ denotes the average-case execution time. In this we assume to get approximately $\sum_0^x r_i^*$ transitions into a given sleep state.

$$\bar{G} = \frac{l - \sum_0^x r_i * \bar{C}_i}{\sum_0^x r_i^*} \tag{10}$$

In order to check on the economic switching to sleep states we need to consider the break even time $t_n^e$ of a sleep state. As the transitions into and out of a sleep state do not produce progress in the computation, but consume energy and time, the system needs to spend at least the break even time in that state.

The break even time can be estimated using the energy quantum to transition into and out of the sleep state $E_n^s$, $E_n^w$, the power consumed while in a given sleep state $P_n$, and the idle power consumption of the system $P_{idle}$. Here Equation 11 quantify the energy of sleep states along with energy of switching cost. Now a valid sleep state which saves on energy when switching to this sleep state needs to satisfy Equation 12.

$$P_{idle} * (t_n^e + t_n^s + t_n^w) = E_n^s + E_n^w + t_n^e * P_n \tag{11}$$

$$\bar{G} \leq t_n^s + t_n^w + t_n^e \tag{12}$$

Additionally, the energy consumption of the system needs to be checked against the energy consumption under DVFS, as well as against the general schedulability test used in the system. For the latter only the switching time required to enter and leave sleep states need to be considered.

## 4. Slack Management

In this section we assume that online slack information is available. In general we only assume the difference between WCET and actual execution time is known, while the slack caused by sporadic release of tasks is much harder to identify at runtime. A particular concern is that tasks may obtain slack after being preempted and thus already having some of their execution completed. This is of relevance as it adds another dimension in the decision space. In this section we will not assume that the previous release times of all other tasks is known and considered.

Similar to the previous section we assume that infeasible sleep states, which fail to guarantee that all deadlines are met because of the extensive transition time, have been removed. Opposed to the previous section, where a global decision on whether to scale or sleep can be taken, the system has more degrees of freedom for this. We assume the amount of slack passed to as task $\tau_i$ from previous tasks is labeled $S$. We will first discuss the scenario where a task receives $S$ at release time and there are no other tasks in the ready queue. In this scenario the DVFS solution has to be computed using this $S$. In order to avoid the costly computation of the optimal frequency setpoint, which has to take into account the switching overhead etc, we assume this can be obtained with a number of comparisons $S$ against pre-computed values, deciding on the frequency setpoint to be used. Furthermore the number of setpoints can likely be reduced, depending on the type of application. In our example hardware platform, a memory intensive application will use generally higher memory and bus frequencies when compared to a CPU bound application.

In the case of sleep modes, the algorithm can not only consider the slack being passed in, but also the expected slack generated by the task in question, $(C_i - \bar{C}_i)$ or even the average idle time $\bar{G}$ generated in the previous section. Since the expected slack generated by the task is not changing we can consider that as a constant in the trade-off between DVFS and using sleep modes.

When slack is passed in after preemption by a higher priority task we have to distinguish two scenarios. If the previous decision was to opt for a *race to halt* than the situation got actually more in favor of continuing the approach. However, if the previous decision was to opt for DVFS than the situation needs to be reassessed. In a first approximation the same trade off rules as used in the previous case may be used, ignoring any but the slack passed in after the preemption. When keeping track of the amount of execution completed it may happen that the actual execution time used so far is already more than the average-case execution time $\bar{C}_i$ has been executed in which case the trade-off would be more tuned towards DVFS.

In the case where there are more tasks in the ready queue, the situation becomes slightly more complicated. In this case we see the following possible scenarios. Establishing from the tasks in the ready queue the cumulative average-case slack generated by the tasks can be used to drive the decision. It is a fairly minimal extra effort, as the expected slack to be generated is inserted and removed, the same way a task is added and removed from the ready queue.

Another approach we envisage is to keep track of the number of jobs executed without any idle interval to reason about the expected idle interval after all jobs in the ready queue have been completed. The reasoning for this behavior is that the presence of many jobs executed in succession indicates that the next releases of the corresponding tasks will only happen at a later time. Note, that this is only a heuristic, which does not require to keep track of exact points of jobs releases in the past. However the exploration of these issues is future work.

## 5. Conclusion

In this paper we have explored a number of parameters that we need to consider for energy efficient DVFS and sleep states decision in a real-time context. Our proposed approach aims to reduce the design space for making such decision and lighten the runtime overhead for making energy efficient decisions. Future research will tackle work in the area of online DVFS and sleep state decisions and possible heuristics which allow for an efficient trade-off between the two. In particular we will explore how more information about past releases of tasks can be incorporated without causing prohibitive decision making overhead. Furthermore we will explore the impact the proposed methods will have on schedulability analysis methods.

## References

Abeni, L. and Buttazzo, G. (1998). Integrating multimedia applications in hard real-time systems. In *19th RTSS*, pages 4–13.

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

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

Cheng, H. and Goddard, S. (2005). Integrated device scheduling and processor voltage scaling for system-wide energy conservation. In *2005 WS Power Aware Real-time Comput.*

Lawitzky, M. P., Snowdon, D. C., and Petters, S. M. (2008). Integrating real time and power management in a real system. In *4th OSPERT*, Prague, Czech Republic.

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

Snowdon, D. C., Le Sueur, E., Petters, S. M., and Heiser, G. (2009). Koala: A platform for OS-level power management. In *4th EuroSys Conf.*, Nuremberg, Germany.

Snowdon, D. C., Petters, S. M., and Heiser, G. (2007). Accurate on-line prediction of processor and memory energy usage under voltage scaling. In *7th EMSOFT*, pages 84–93, Salzburg, Austria.

XScale (2004). *Intel PXA 255 Processor Developer's Manual*. Intel Corp. URL http://www.xscale-freak.com/XSDoc/PXA255/27869302.pdf.