Fast Calculation Of Permissible Slowdown Factors For Hard Real-Time Systems*

Henrik Lipskoch[1], Karsten Albers[2], and Frank Slomka[2]

[1] Carl von Ossietzky Universität Oldenburg
Henrik.Lipskoch@informatik.uni-oldenburg.de
[2] Universität Ulm
{karsten.albers,frank.slomka}@uni-ulm.de

Abstract. This work deals with the problem to optimise the energy consumption of an embedded system. On system level, tasks are assumed to have a certain CPU-usage they need for completion. Respecting their deadlines, slowing down the task system reduces the energy consumption. For periodically occurring tasks several works exists. But even if jitter comes into account the approaches do not suffice. The event stream model can handle this at an abstract level and the goal of this work is to present and solve the optimisation problem formulated with the event stream model. To reduce the complexity we introduce an approximation to the problem, that allows us a precision/performance trade-off.

1 Introduction

Reducing the energy consumption of an embedded system can be done by shutting down (zero voltage), freezing (zero frequency, e.g. clock gating) or stepping the circuits with a slower clock and lower voltage (Dynamic Voltage Scaling or Adaptive Body Biasing).
On system level, tasks as programmes or parts of those are assigned a processing unit. Here we are interested in tasks having a deadline not to miss and with some sort of repeated occurrence, that is those tasks are executed repeatedly as long as the embedded system is up and running.
The mentioned possibilities to reduce the overall energy consumption result in a delay or slowdown from the view of a program running on the system. It has a lower-bound for those tasks having deadlines to meet with the side effect, that the available processing time for other tasks running on the the same processor will be reduced. Thus any technique regarding energy reduction which influences the processing speed has to take these limits into account.
The problem we focus here is to minimise the systems total power consumption for a task set, where each task may have its own trigger and its own relative deadline, using static slowdown to guarantee hard real-time feasibility.
In the next section we will describe work on similar problems. In the model section we will specify our assumptions, describe the event stream model along with the demand bound function. We will improve the number of test points and then

* This work is supported by the German Research Foundation (DFG), grant GRK 1076/1 and SL 47/3-1. It has been published by Springer, DOI 10.1007/978-3-540-74442-9_48 (Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation, 2007)

show how this is incorporated into a linear programme solving the problem of calculating slowdown factors, while guaranteeing hard real-time feasibility. Before we conclude the paper, we will do an example and demonstrate the advantages, we gained with the developed theory.

2 Related Work

There are several works regarding the energy optimisation of hard real-time systems. Here we are interested in optimising without granting the circuits any loss of precision, without missing any deadlines, and we want a true optimal solution, if possible, thus we focus on linear program solutions.
Ishihara and Yasuura [1] incorporate in their integer linear program different discrete voltage/frequency pairs, an overall deadline for their task set to meet, but no intertask dependency and only dynamic power consumption. They consider mode switching overhead negligible and for every task a number of needed CPU-cycles, in worst-case analysis this corresponds to their worst-case execution time.
Andrei et al. [2] formulate four different problems, each regarding intertask dependency, a deadline per task, and per task a number of clock cycles to complete it, which can, again, be considered as worst-case time. The four problems vary in with or without regarding mode switching overhead and with or without integer linear programming for the number of clock cycles. The task set is considered non-preemptive. The authors prove the non-polynomial complexity of the integer linear problem with overheads.
A somewhat different approach have Hua and Qu [3]. They are looking for the number and the values of voltages yielding the best solution with dynamic voltage scaling. However, their problem formulation only explores the slack between execution time and relative deadline of a task with the assumption that the task system is schedulable even if every task uses the complete deadline as execution time. In hard real-time analysis this assumption does not hold. For example if each task has its own period a common way is to set its deadline equal to its period.
Rong and Pedram [4] have their problem formulated with intertask dependencies, leakage and dynamic power consumption, different CPU-execution modes, external device usage each with different execution modes. They state that mode switching overhead on the CPU is negligible especially when “normal” devices (e.g. hard disks etc.) are involved in the calculation. They also state the non-polynomial complexity of the mixed integer linear program. And their task graph is assumed to be triggered periodically with a period they use as overall deadline for the task set. The tasks are considered non-preemptive. The tasks themselves do not have an own deadline.
In [5] Jejurika and Gupta present a work on calculating slowdown factors for energy optimisation of earliest-deadline-first scheduled embedded systems with periodic task sets. The work does not consider static power consumption and therefore does not take switching off the CPU into account.
They first present a method to calculate a slowdown factor for a whole task set using state-of-the-art real-time feasibility tests. Then they develop a method with ellipsoids to have a convex optimisation problem. This test incorporates the feasibility test of Baruah [6], which turned out to be very computational intensive, because it requires to test all intervals up to the hyper period of the task system

in the case of a periodic task system (again, see [6]) in other cases.
To face the problem of computational intensity Albers and Slomka developed another test [7] with a fast approximation [8], having the possibility to do a performance/precision tradeoff, based on the event stream methodology by Gresser [9]. Here, in this work, we want to show how this approximation applies for the problem of calculating slowdown factors.

3 Model

We do not want to limit us to periodic triggered task systems. Therefore, we assume that a task can be triggered not only periodic, but periodic with a jitter or sporadic with minimal distance between two consecutive triggers or other forms. We assume the triggers of a task be given in the form of an event stream, see below. For the scheduling algorithm, we assume preemptive earliest deadline first scheduling.
When we speak of a task system we speak of tasks sharing one single processor concurring on the available processing time. Additional we assume it to be synchronous, i.e. all tasks can be triggered at the same point in time.
An invocation of a task is called a job and the point in time when a task is triggered an event.
Each task of our task systems is assumed to have a relative deadline d, measured from the time when the task is triggered, a worst-case execution time c, and an event stream denoting the worst-case trigger of that task. The latter one is described in the following subsection.

3.1 Event Streams

Motivated by the idea to determine bottlenecks in the available processing time, Gresser ([9]) introduced the event stream model, which is used in [7] to provide a fast real-time analysis for embedded systems. We understand a bottleneck as a shortest time span in which the most amount of processing time is needed, i.e. a time span with the highest density of needed processing time. The event stream model covers this, time spans with maximal density are ordered by their length (maximal among time spans having the same length: go through all intervals on the time axis having the same length and take the length of that one, that has the least available processing time within).
Achieved is this by calculating the minimal time span for each number of task triggers.

Definition 1. *Let τ be a task. An event stream $E(\tau)$ is a sequence of real numbers $a_1, a_2, \ldots$, where for each $i \in \mathbb{N}$ a_i denotes the length of the shortest interval in time in which i number of events of type τ can happen. (See [7] for a more detailed definition)*

Event streams are sub-additive sequences, i.e. $a_i + a_j >= a_{i+j}$, for all $i, j \in \mathbb{N}$. Albers and Slomka explain how to gather that information for periodic, periodic with jitter, and sporadic triggered tasks [7].

Example 1. Consider the following three tasks.

1. Let τ_1 be triggered with a period of 100 ms. Then the shortest time span to trigger two tasks is 100 ms. For three it is 200 ms and so on, thus the resulting event stream is $E(\tau_1) : a_1 = 0s, a_n = (n-1) \cdot 100ms$.
2. Then, let τ_2 be triggered sporadically with a minimal distance between two events of 150 ms. Then the shortest time span to trigger two tasks is 150 ms. For one task more it is 300 ms. The resulting event stream is $E(\tau_2) : a_1 = 0s, a_n = (n-1) \cdot 150ms$.
3. And let τ_3 be triggered periodically every 60 ms but can be triggered 5 ms before and after its period. Thus the shortest time span to trigger two tasks is 50 ms, which corresponds to one trigger 5 ms after one period and the next trigger 5 ms before the next period. The then earliest task after both can not be triggered shorter than 60 ms later, which is 5 ms before the over-next period, and this corresponds to a time length of 110 ms to trigger 3 tasks. Following this argumentation the resulting event stream is $E(\tau_3) : a_1 = 0s, a_2 = 50ms, a_n = 50ms + (n-2) \cdot 60ms$.

3.2 Demand bound

To guarantee the deadline of a task one has to consider the current workload of the resource the task runs on. The demand bound function (see [10] and [7]) provides a way to describe this, and the np-hard feasibility test using this function can be approximated in polynomial time [7].
For the workload we now calculate the maximal amount of needed processing time within an interval of length Δt. If we allow the simultaneous trigger of different tasks, which was our assumption, this leads to synchronising the event streams, and that is to assume all the intervals, out of which we obtained the time lengths for our event streams, have a common start. Thus, the sum of the worst-case execution times of all events in all event streams happening during a time of length Δt, having their deadline within that time, gives us an upper bound of the execution demand for any interval of length Δt. Note, that we only have to process jobs with deadline within this time span. Formulated with the notion of definition 1 the demand bound function turns out as follows.

Definition 2. *The demand bound function denotes for every time span an upper bound of workload on a resource to be finished within that time span.*
(See for ex. [10])

Lemma 1. *Let $\tau_1, \dots, \tau_n$ be tasks running on the same resource, each with worst-case execution time c_i and relative deadline d_i, $i = 1, \dots, n$. And let $E_1, \dots, E_n$ be their event streams. Define $a_0 := -\infty$. Under the assumption all tasks can be triggered at the same time, the demand bound function can be written as*

$$D(\Delta t) = \sum_{i=1}^{n} \max\{j \in \mathbb{N}_0 : a_j \in E_i \cup \{a_0\}, a_j \leq \Delta t - d_i\} c_i.$$

(see [7])

Example 2. Consider the task set of example 1. Let the deadlines be 30 ms for the first, 20 ms for the second, and 10 ms for the third task; let the worst-case execution times for each task be 25 ms, 15 ms, and 5 ms, respectively. Out of these properties, we obtain the demand bound function, which is

$$D(\Delta t) = \left\lfloor \frac{\Delta t + 70ms}{100ms} \right\rfloor \cdot 25ms + \left\lfloor \frac{\Delta t + 130ms}{150ms} \right\rfloor \cdot 15ms \\ + \left(\max\left\{ 0, \frac{\Delta t - 10ms}{|\Delta t - 10ms|} \right\} + \left\lfloor \frac{\Delta t}{60ms} \right\rfloor \right) \cdot 5ms.$$

The next step proceeds with a match of the needed processing time below the available processing time. This is the feasibility test of the real-time system. Since one can process exactly t seconds processing time within an interval of t seconds, if every consumer of processing time is modelled within the task set, and thus the feasibility test results in proving

$$D(\Delta t) \leq \Delta t \quad \forall \Delta t > 0. \tag{1}$$

Lemma 2. *Let $\tau_1, \ldots, \tau_n$ be tasks and $E(\tau_1), \ldots, E(\tau_2)$ their corresponding event streams. A sufficient set of test points for the demand bound function is $E' := \bigcup_{i=1}^{N} \{a_i + d_i : a_i \in E(\tau_i)\}$.*

Proof. The demand bound function remains constant between two points $e_1, e_2 \in \tilde{E}$.

The values of E' can be bounded above and the remaining set will still be a sufficient test set. If the event streams contain only periodic behaviour, it is feasible to use their hyper period, since this is defined as the least common multiple of all involved periods it grows as the prime numbers contained in the periods grow (cf. $p_1 = 31 * 2 = 62, p_2 = 87 : H = p_1 * p_2 = 5394$, whereas $p_1' = 60$ and $p_2' = 90$ will yield $H = 180$). Another test bound exists [6] covering also non-periodic behaviour. It depends on the utilisation U: $\Delta t_{\max} = \frac{U}{1-U} \cdot \max\{T_i - d_i\}$ and it cannot be used here, because in slowing down the system, we will increase its utilisation (more processing time due less speed) and thus formulas, similar to the one mentioned, will result in infinite test bounds (which is the reason that such formulas are only valid for utilisations strict less than 1). Instead of using such test bounds, we improve the model in another way.

Definition 3. *A bounded event stream with slope s from k on is an event stream E with the property*

$$\forall\, i \geq k, a_i \in E : \ \frac{1}{a_{i+1} - a_i} \leq s. \tag{2}$$

The index k is called the approximation's start-index.

Because of the sub-additivity the pair $(s,k) = (a_2, 2)$ always forms a bounded event stream, this is used in [8], but in changing the index, we may change the precision, as the following example shows:

Example 3. Let there be a jittering task with period 100 ms and a jitter of 5 ms. Then approximating with $a_2 = 90$ ms will have a significant error. Starting the approximation at index 2 with $a_3 - a_2 = 100$ ms will end up in no error at all!

We summarise this information more formal in the following lemma.

Lemma 3. *Let task τ have a bounded event stream E with slope s from k, then an upper bound on its demand is*

$$D_\tau(\Delta t) = \begin{cases} c \cdot \max\{j : a_j \in E, a_j + d \le \Delta t\} & \Delta t < a_k + d \\ c \cdot \left(k - 1 + \frac{\Delta t - a_k - d}{s}\right) & \Delta t \ge a_k + d. \end{cases} \tag{3}$$

Note that the growth of the function has its maximum between 0 and $a_k + d$, because for values greater than $a_k + d$ the function grows with $\Delta t / s$, which must be less or equal the maximal growth according to the sub-additivity of the underlying event stream.

The definition reduces our set of test-points depending on the wanted precision.

Theorem 1. *Let $\tau_1, \ldots, \tau_n$ be tasks, $c_1, \ldots, c_n$ their worst-case execution times, and let $E_1, \ldots, E_n$ be their bounded event streams with slopes $s_1, \ldots, s_n$ and approximation start-indices $k_1, \ldots, k_n$.*
Define $\tilde{E} := \bigcup_{i=1}^{n} \{a_{i,j} + d_i : a_{i,j} \in E_i, a_{i,j} \le a_{i,k_i - 1}\}$.
A sufficient feasibility test is then

$$\forall \Delta t \in \tilde{E} : \qquad D(\Delta t) \le \Delta t \tag{4}$$

$$\text{and} \qquad \sum_{i=1}^{n} \frac{c_i}{s_i} \le 1. \tag{5}$$

Proof. In Lemma 2 we stated that the demand bound function is constant between the test-points of $\tilde{E}$. For values greater than $A := \max\{a \in \tilde{E}\}$ we approximate the demand bound function by a sum over straight lines for each task, cf. lemma 3:

$$D(\Delta t) \le D' := \sum_{i=1}^{n} g_i(\Delta t) \forall \Delta t > A,$$

$$\text{with:} \qquad g(\Delta t) = c \cdot \left(k_i - 1 + \frac{\Delta t - a_{i,k} - d_i}{s_i}\right)$$

The growth of D' has its maximum between 0 and A, because this is true for the elements of the sum (compare note in lemma 3). If the function D' is below the straight line $h(x) := x$ for values between 0 and A, then it will cut h for values greater than A if and only if its derivate there is greater than 1. That results in proving:

$$\begin{aligned} 1 &\ge \frac{\partial}{\partial \Delta t}(D'(\Delta t)) \qquad (\Delta t > A) \\ &= \frac{\partial}{\partial \Delta t}\left(\sum_{i=1}^{n} g_i(\Delta t)\right) \\ &= \sum_{i=1}^{n} \frac{\partial}{\partial \Delta t}\left(c \cdot \left(k_i - 1 + \frac{\Delta t - a_{i,k} - d_i}{s_i}\right)\right) \\ &= \sum_{i=1}^{n} \frac{c_i}{s_i}. \end{aligned}$$

If a task system's demand bound function allows some "slack", that is it does not use the full available processing time, we are interested what happens to the calculation if we introduce another task into the system. The argumentation is clear: it has to fit into the rest available processing time. To be more formal we state the following lemma, which basically expresses, that we do not have to recalculate the demand bound as a whole but only for the new test-points.

Lemma 4. *Let Γ be a real-time feasible task system and let D' be it's demand bound function in the notion of the theorem. Let τ be a task with event stream E, deadline d and worst-case execution time c.*
Then the task system $\Gamma \cup \{\tau\}$ is real-time feasible if

$$D'(\Delta t) + \max\{j \in \mathbb{N} : a_j \in E, a_j + d \leq \Delta t\} \cdot c \leq \Delta t \qquad \forall \Delta t \in \{a_i + d : a_i \in E\}. \tag{6}$$

(cf. [7])

Proof. Since the function D' will by prerequisite not exceed the line $h(x) = x$, this violation can only occur at points when the task τ needs to be finished.

Clearly, if the introduced task has a bounded event stream with some slope s from some index k on, the set of test-points reduces to those induced by indices less than k.
We summarise the gained complexity reduction along with the accuracy of the test.

Lemma 5. *The complexity of the test for a periodic only task system is linear in the number of tasks, if for all tasks the deadlines are equal to the periods, no accuracy will be lost.*
The complexity of the test for a periodic task system with m tasks having a jitter is linear in the number of tasks plus m.

Two reasons for loss in accuracy exist. On the one hand, there is an error by the assumption of synchronicity. And on the other hand there is an approximation error due to linearisation.

3.3 Linear Programme

Our optimisation problem can now be formulated as a linear programme. Since our goal is to slow down the task system as much as it is allowed, the corresponding formulation (in the notion of the theorem) for the objective is then

$$\text{Maximize: } \sum_{i=1}^{n} \frac{\alpha_i \cdot c_i}{s_i}, \tag{7}$$

where α_i is the slow down for task i.
Note that a slowdown factor of < 1 means to speed up the task as this shortens its execution time and therefore we have to ensure the opposite:

$$\alpha_i \geq 1. \tag{8}$$

As stated in the theorem, the long term utilisation must not exceed 1, this gives us the constraint:

$$\sum_{i=1}^{n} \frac{\alpha_i \cdot c_i}{s_i} \leq 1. \tag{9}$$

Clearly, the optimum will never exceed 1.
Let $a_{i,j}$ denote the j-th element in the event stream belonging to task i. The constraint limiting the demand is then:

$$\forall\, \Delta t \in \tilde{E} \sum_{i=1}^{n} \max\{j \in \mathbb{N} : a_{i,j} + d_i \leq \Delta t\} \cdot \alpha_i \cdot c_i \leq \Delta t. \tag{10}$$

The max-term in the equation is calculated beforehand, because it does not change during optimisation, if it's value reaches a start-index k of some task then it is replaced by the equation 3 given in lemma 3.

4 Experiments

As first experiment we chose a rather simple example given in [11], describing seven periodic tasks, having their deadlines equal to their periods, on a Palm-pilot (see table 1). It has a utilisation of 86.1%. Calculation with the unimproved test results in slowing down task 7 by 3.075 with the help of 45 constraints. Exactly the same slowdown was calculated by our fast approach with only 7 constraints.

Table 1. Task set of the Palm-pilot

Task	Exec. Time [ms]	Period [ms]
1	5	100
2	7	40
3	10	100
4	6	30
5	6	50
6	3	20
7	10	150

For demonstration purpose, we chose a periodic task system with some tasks having a deviation in their period, whose maximal value is found as jitter in table 4. The example was taken from [12]. The task set has a utilisation of about 65.2%. We first applied the slowdown calculation without test-point reduction and tested up to the hyper-period of the task periods, which is 59,000,000, and the test resulted in 78562 constraints concerning the demand bound function. It yielded a slowdown for task 9 of about 9.7, a utilisation of exactly 1, which both are the same result as with the improved linear program we suggested with our developed theory having only 20 constraints regarding the demand bound function.
All linear programs were programmed in GNU MathProg modelling language and solved with glpsol, version 4.15 [13].

Table 2. Task set of processor one

Task	Exec. Time [μs]	Jitter [μs]	Deadline [μs]	Period [μs]
1	150	0	800	800
2	2277	0	5000	200000
3	420	8890	15000	400000
4	552	10685	20000	20000
5	496	9885	20000	20000
6	1423	0	12000	25000
7	3096	0	50000	50000
8	7880	0	59000	59000
9	1996	15786	100000	50000
10	3220	34358	10000	100000
11	3220	55558	10000	100000
12	520	0	10000	200000
13	1120	107210	20000	200000
14	954	141521	20000	1000000
15	1124	0	20000	200000
16	3345	0	20000	200000
17	1990	0	100000	1000000

5 Conclusion and Future Work

We have shown a very fast and yet accurate method for calculating static slowdown factors while providing hard real-time feasibility. In contrast to other methods it does not rely on periodic task behaviour, it's complexity does not increase when other forms of trigger, like sporadic with minimal distance between two consecutive triggers or periodic with a certain jitter, are part of the optimisation problem. In future work we want to embed criteria to allow modelling different system states such as sleep states and with our methodology we want to research in what cases a common slow down factor will be sufficient.

References

1. Ishihara, T., Yasuura, H.: Voltage scheduling problem for dynamically variable voltage processors. In: Proceedings of the International Symposium on Low Power Electronics and Design. (1998) 197–202
2. Andrei, A., Schmitz, M., Eles, P., Peng, Z., Al-Hashimi, B.M.: Overhead conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. In: Proceedings of the Design Automation and Test in Europe Conference. (2004)
3. Hua, S., Qu, G.: Voltage setup problem for embedded systems with multiple voltages. IEEE Transactions on Very Large Scale Integration (VLSI) Systems (2005)
4. Rong, P., Pedram, M.: Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system. In: Proceedings of the Asia and South Pacific Design Automation Conference. (2006)

5. Jejurika, R., Gupta, R.: Optmized slowdown in real-time task systems. In: Proceedings of the 16th Euromicro Conference on Real-Time Systems. (2004) 155–164
6. Baruah, S.K., Rosier, L.E., Howell, R.R.: Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Real-Time Systems **2**(4) (1990) 301–324
7. Albers, K., Slomka, F.: An event stream driven approximation for the analysis of real-time systems. In: Proceedings of the Euromicro Conference on Real-Time Systems. (2004) 187–195
8. Albers, K., Slomka, F.: Efficient feasibility analysis for real-time systems with edf scheduling. In: Proceedings of the Design Automation and Test in Europe Conference. (2005)
9. Gresser, K.: An event model for deadline verification of hard real-time systems. In: Proceedings of the Fifth Euromicro Workshop on Real Time Systems. (6 1993) 118–123
10. Baruah, S., Chen, D., Gorinsky, S., Mok, A.: Generalized multiframe tasks. Real-Time Systems **17**(1) (7 1999) 5–22
11. Lee, T.M., Henkel, J., Wolf, W.: Dynamic runtime re-scheduling allowing multiple implementations of a task for platform-based designs. In: Proceedings of the Design, Automation and Test in Europe Conference. (2002)
12. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems) **40**(2-3) (1994) 117–134
13. Makhorin, A.: GLPK linear programming/MIP solver. http://www.gnu.org/software/glpk/glpk.html (2005)