CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Optimisation of battery operating life considering software tasks and their timing behaviour

Dissertation zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften

von

Dipl.-Math. Henrik Lipskoch

Gutachter:
Prof. Dr.-Ing. Wolfgang Nebel
Prof. Dr.-Ing. Frank Slomka

Tag der Disputation: 19.02.2010

Ackknowledgements

Everyone who was patient with me during this step of my life, I want to thank: my colleges, for many honest comments and plenty of discussions; my parents, my brothers, and their families, for their mental support; Mario Korte, for he looked over this work; Johannes Faber, for when we were running twice a week, he was never tired to talk over our progress on the street, in our research, and in life; a very special thanks goes to Thomas Weißmüller and to Nils Schröder, who made such a reliable and fully automated device enabling my battery measurements.

Abstract

Users of mobile embedded systems have an interest in long battery operating life. The longer a system can operate without need for recharge or battery replacement, the more will maintenance cost and the number of faults due to insufficient power supply decrease. Operating life is prolonged by saving energy, which may reduce available processing time. Mobile embedded systems communicating with other participants like other mobiles or radio stations are subject to time guarantees ensuring reliable communication. Thus, methods that save energy by reducing processing time are not only subject to available processing time but subject to the embedded system's time guarantees. To perform parameter optimisations offline, decisions can be taken early at design time, avoiding further computations at run-time. Especially, to compute processor shutdown durations offline, no extra circuitry to monitor system behaviour and to wake up the processor needs to be designed, deployed, or power supplied: only a timer is required.

In this work, software tasks are considered sharing one processor. The scheduling algorithm earliest deadline first is assumed, and per-task, a relative deadline is assumed. Tasks may be instantiated arbitrarily as long as this occurrence behaviour is given in the notion of event streams. Scaling of the processor's voltage and processor shutdown are taken into account as methods for saving energy. With given per task worst-case execution times and the tasks' event streams, the real-time feasibility of the energy optimised solutions is proven. The decision which energy saving solution provides longest operating life is made with the help of a battery model.

The used real-time feasibility test has the advantage that it can be approximated: this yields an adjustable number of linear optimisation constraints.

Reducing the processor's voltage reduces processor frequency, therefore, execution times increase. The resulting slowdown becomes the optimisation variable, either global, for all tasks, or local, individually per task. It is shown that a global slowdown factor can be computed from the linear constraints together with a linear objective in a linear program, whereas local slowdown factors are shown to require a non-linear but still convex objective.

Processor shutdown switches off the processor for a specific time; it blocks the processor from processing software. Duration and occurrence become optimisation parameters, and the linearised real-time feasibility test is used to decide whether a parameter setting is feasible or violates deadlines.

In this thesis, two shutdown policies are introduced. Periodic shutdown switches off the processor periodically, thus, besides duration, the period is to be optimised. Task-dependent shutdown switches off the processor with an inherited occurrence behaviour from a certain task, thus, besides duration, the parent task needs to be chosen, as well as the number of instances of the parent task to fall between two consecutive shutdowns.

Each of the presented methods for saving energy yields a different system configuration, out of which the best with respect to operating life is chosen with the help of a battery model. Discharge profiles serve as a coupling between information in the notion of event streams, task power consumptions, and the battery model. A battery model evaluation with measurements is presented in this thesis.

Zusammenfassung

Benutzer mobiler eingebetteter Systeme haben ein Interesse an einer langen Batterienutzungsdauer. Je länger ein System ohne Nachladen oder Batteriewechsel operieren kann, desto geringer werden Wartungskosten und durch Energiemangel bedingte Ausfälle. Energiesparmethoden helfen die Nutzungsdauer zu verlängern, können jedoch zu einer Reduzierung der verfügbaren Rechenzeit führen. Die Systeme können zeitlichen Anforderungen unterworfen sein, etwa um eine zuverlässige Kommunikation sicherzustellen. Somit ist der Einsatz von Energiesparmethoden neben der verfügbaren Rechenzeit auch an die zeitlichen Anforderungen gebunden. Indem Optimierungen zur Designzeit durchgeführt werden, lassen sich früh Entscheidungen treffen und weitere Berechnungen zur Laufzeit vermeiden. Speziell die Parameterberechnung für die Prozessorabschaltung zur Designzeit ermöglicht ein Auskommen ohne zusätzliche systemüberwachende Schaltkreise, die ihrerseits entworfen und mit Energie versorgt werden müssten: für die Abschaltung ist nur eine programmierbare Echtzeituhr nötig.

In dieser Arbeit werden Software-Tasks betrachtet, die sich einen Prozessor teilen. Für die Ablaufplanung wird ausschließlich der Algorithmus "nach frühester Frist zuerst" betrachtet. Es wird den Software-Tasks ein beliebiges Auftreten zugestanden, solange sich dieses mit der Ereignisstrommethodik formulieren lässt. Als Energiesparmethoden werden Prozessorspannungsskalierung und Prozessorabschaltung betrachtet. Mit Hilfe von als gegeben vorausgesetzten Task-Laufzeitoberschranken und auf Basis der Ereignisströme, wird die Echtzeitfähigkeit der energieoptimierten Lösungen geprüft. Die Entscheidung, welche Lösung sich am besten für eine lange Nutzungsdauer eignet, wird mit Hilfe eines Batteriemodells getroffen.

Der benutzte Echtzeittest hat den Vorteil approximierbar zu sein, d.h. er lässt sich linearisieren und für die Optimierung als Menge linearer Bedingungen nutzen.

Prozessorverlangsamung reduziert die Taktfrequenz und erhöht damit die Rechenzeit der Tasks. Der sich aus der Frequenz ergebende Verlangsamungsfaktor wird zur Optimierungsvariablen, entweder global, d.h. für alle Tasks gleichermaßen, oder lokal, d.h. für jede Task individuell. Ein globaler Faktor lässt sich mit Hilfe einer linearen Zielfunktion und den linearen Bedingungen aus dem Echtzeittest in einem linearen Programm optimieren. Dagegen, wie in dieser Arbeit gezeigt, benötigt die Optimierung für lokale Verlangsamungsfaktoren eine andere Zielfunktion, die nicht-linear, aber noch konvex ist.

Den Prozessor abzuschalten, heißt alle Berechnungen für eine bestimmte Zeit auszusetzen. Dauer und Häufigkeit der Abschaltung werden Optimierungsvariablen, und deren Echtzeitfähigkeit wird mit Hilfe des linearisierten Echtzeittests überprüft.

Zwei Abschaltverfahren werden in dieser Arbeit vorgestellt. Bei periodischem Abschalten wird, neben der Abschaltdauer, die Periodendauer optimiert. Task abhängiges Abschalten koppelt die Abschaltung an das Auftreten einer bestimmten Task. Neben der Abschaltdauer, werden Task und die Anzahl der Task-Instanzen, die zwischen zwei Abschaltungen fallen sollen, bestimmt.

Aus der Optimierung, entsprechend der in dieser Arbeit vorgestellten Energiesparmethoden, resultieren je nach verwendeter Methode unterschiedliche Systemkonfigurationen. Aus welcher sich die längste Batterienutzungsdauer ergibt, wird mit Hilfe eines Batteriemodells ermittelt. Entladeprofile dienen der Kopplung von Task-Spezifikationen und Batteriemodell. Eine Evaluation des Modells anhand von Messungen wird in dieser Arbeit vorgestellt.

Contents

List of Figures

List of Tables

1. Introduction

A traffic bridge often serves for many years millions of cars to pass rough terrain. When such a bridge is built, cranes lift and shift large structures. The cranes are controlled by persons who direct bridge components into the right positions. Communication between controller and crane is done via radio. If the radio fails or transmits twisted messages communication is interrupted: structures might get broken; people might get severe injuries. Damaged material and hurt humans become the consequences of unreliable communication.

One reason why communication eventually fails is a weak battery, as mobiles and radios usually are powered by batteries. Methods for saving energy prolong battery operating life, and the most energy surely is saved by switching the device off, but with the drawback that communication is lost: energy savings are subject to the radio's correct functionality. A forecast on battery operating life provides a guarantee on the system's minimal run-time, and an early forecast, at design time of the system, allows for an early maintenance estimation which helps to avoid shortages when the system is in use. Energy savings and a forecast on operating life will increase the system's reliability.

Modern embedded systems are implemented partly in software and in hardware. The software is divided into tasks, and the tasks share a processor. Each task may include communication. Timing constraints in form of deadlines are part of the system's specification to ensure reliable operation. Methods for saving energy are subject to these constraints, especially those methods which affect a software's processing time. In this thesis, the latter are considered. Further, a processor that provides several levels of speed and energy consumption, both controllable by software, is considered. The energy supply is supposed to be a battery. Parameters describing this setting are the basis for the work presented in this thesis.

Beyond deadlines, parameters describing embedded software include a description of the software's occurrence behaviour, which is the frequency tasks are instantiated to become ready for processing. This behaviour can be periodic; the period itself can be affected by deviations (jitters), e.g. caused by time-slot-based communication; behaviour can be sporadic, yet with a minimal time to pass between two consecutive task instantiations; or, for example, behaviour can be affected by time shifts, e.g. caused by a clock with skew. Further, parameters include upper bounds on execution times. The bound on a task's execution time and its occurrence behaviour form a task's timing behaviour. Both can be combined to compute upper bounds on the processing demand. The sum of all tasks' demands yields the overall demand, and thus the available idle time, which is to be exploited by methods for saving energy. For the work in this thesis, it is assumed that deadline, worst-case execution time, and occurrence behaviour are given for each task of the task set.

Parameters describing the processor include the processor's ability to perform voltage and frequency scaling, the ability to transit into low-power modes, and the amount of modes available. The processor's power consumption differs for various modes and for different software. Infor-

mation on power consumption and timing behaviour combine to an energy demand estimation: this is an aim of this thesis, therefore, power consumptions for different modes and software are assumed to be given.

Parameters describing the battery include brand and capacity. A battery model based on these parameters is capable of estimating the battery's operating life for a given energy demand. Therefore, these parameters are assumed to be given as well.

The problem to be solved by the work described in this thesis is increasing a battery's operating life by applying methods for saving energy on an earliest-deadline-first scheduled set of tasks, with each task not necessarily occurring periodically, and with hard real-time constraints in form of a relative deadline for each task.

Additionally, because different methods for saving energy result in different system configurations and the best configuration might be battery dependent, a battery model is supposed to be used to choose the most appropriate configuration.

The stated problem is to be tackled by the work presented in this thesis with the help of the theory of event streams. The theory follows the idea to cover all cases by only inspecting worst-cases: if an assertion holds for a worst-case, so does it for all cases included in the worst-case. A successful worst-case test will provide a reliable guarantee for all situations that might happen while the embedded system is run.

Note, the theory of event streams is also capable of handling preemptive fixed-priority systems.

The work is structured into eight chapters, each composed of sections, subsections, subsubsections, and paragraphs. Each chapter ends with a summary or a conclusion. Chapter 2 includes the foundations helping to understand the rest of the work. Chapter 3 is a discussion of works dealing with similar problems as the one described in this thesis; it gives a survey on battery models.

The concept behind the work presented in this thesis is shown and classified in Chapter 4. The reader will learn about the interaction of processor slowdown and the real-time feasibility test in Chapter 5; regardless of the occurrence behaviour, optimisation of processor slowdown is done per task and per task set, with the real-time test providing the optimisation constraints. The reader will find in Chapter 6 the idea to model processor shutdown as a hard real-time task, and the optimisation of its occurrence behaviour and execution time in several directions. Chapter 7 discusses properties of batteries affecting their operating life, enriched by experimental results; the chapter presents the chosen battery model and the coupling of task timing and power parameterisation to a battery model in the form of discharge profiles. The reader will find the descriptions of experimental setups and further experimentation results in Chapter 8.

A conclusion drawn in Chapter 9 summarises the results and the achievements presented in this thesis, further, it provides an outlook on future work.

An appendix contains further notable material beyond the thesis. The appendix is followed by a list of literature. An index containing frequently used notions and symbols forms the end of this thesis.

2. Preliminaries

Embedded software is used if a pure hardware solution would be too complex to design from scratch, to achieve flexibility, to allow extensions, or for prototyping reasons. The software may be divided into parts, called tasks, which are to be instantiated and executed independently on a processor. There are many reasons to divide the assignment of an embedded system into several tasks designed for parallel execution. First, an embedded system needs not to be assigned to one single objective. For example a system for traffic surveillance can additionally record weather observations; both applications are separate, but they share hardware resources.

Another reason to divide an objective into multiple tasks lies behind communication with other participants. Interrupt based communication will create jobs – task instantiations – when an interrupt is thrown and the interrupt handler is activated. The system waits passively for incoming interrupts and then processes the jobs. Jobs are allowed to be interrupted for certain occasions. The occasion then has a higher priority than the processed job.

Definition 2.0.1. Task and job distinguish like offline and run-time.

a) A task is defined as a set of commands to be executed by a processor. It is the implementation of a program.

b) A job is an instance of a task handled by the operating system, i.e. the dispatcher. The job is the unit to be scheduled by the operating system's scheduler.

Compared to object oriented programming, a task refers to the class, and the job refers to the object.

Already at design time of a software, times may be specified defining when the software has to provide results, with the semantic that if the software completes after these times, then its results are no longer valuable. These times are called deadlines and are a task's property.

Further task's properties are: upper bounds on job execution times – worst-case execution times; upper bounds on processor power consumption while it processes a job – worst-case execution power; and energy needed for processing – worst-case execution energy.

The total amount of needed processing time is estimated with the help of the job creation rate. This occurrence rate, or at least an estimation of this, belongs to a task's timing behaviour as well and defines a starting point for energy optimisations presented in this thesis.

The first section of this chapter starts with the scheduling problem. The next section introduces event streams to solve the analysis complexity. Event stream approximation is shown in a further section. With this preparation, the concept to determine upper bounds on processing demand is shown. After that, the sources of power dissipation are introduced as motivation and description for methods to gain lower power dissipation. The chapter closes with an introduction on batteries as suppliers for electric energy.

2.1. Scheduling

The scheduler's assignment is organising all jobs for processing. It is called by the dispatcher if a job is to be transferred from state "ready" to state "running". The scheduler chooses the next job out of a list, called "ready-queue", by a certain set of rules, called "scheduling policy". Rules are applied to job properties like estimated execution time, time the job was inserted into the queue, or a number, called "priority". The latter can be fixed offline, called "static priority", or to be determined online, called "dynamic priority."

The number determining the priority may be interpreted as the time after a job has to be finished, from creation on, a relative deadline. A scheduling policy chooses the job with the smallest number – the shortest time to finish. Suppose the time to finish is fixed offline: if the job's creation time is not added to it, we have the case of deadline monotonic scheduling; if the job's creation time is added, we have the case of earliest deadline first scheduling (EDF).

If these deadlines are to be met and must not be violated, we speak of a hard real-time system.

Definition 2.1.1. An offline analysis proving that all jobs will ever meet their deadlines is called hard real-time feasibility test.

The dynamic priority scheduling with the help of deadlines (EDF) is known to be capable to schedule the highest load, that is, the authors of [LL1973] showed in their work that a task set is real-time feasible if and only if the offline feasibility test with EDF succeeds, and because of this fact, the work presented in this thesis considers jobs being scheduled earliest deadline first.

The test on real-time feasibility has to prove deadline retention for all situations that may occur. Thus, any simulation not guaranteeing to inspect all critical cases is insufficient. Every time a deadline has to be fulfilled, the test has to check it; the more deadlines to check, the more to test; and the more jobs created, the more deadlines to check. The number of test points decreases whenever two different deadlines fall onto the same point – the number of requirements to test remains the same – but the number of test points increases when jobs are created asynchronous or the tasks' occurrence behaviours have less in common, for example, when there are two tasks that occur periodic but both periods do not share the same primes numbers.

The first reason increasing the number of test points is overcome by:

Definition 2.1.2. The synchronicity assumption assumes that all tasks of the task set eventually occur together at one point in time.

That is, there is a point in time when all tasks are instantiated: for each task there is one job created at the point of synchronicity.

Now, if all tasks occur periodic, the interval starting at the point of synchronicity and with the size equal to the least common multiple of the tasks' periods (*L*CM) will contain all job constellations that can occur. If we define the point of synchronicity 0, then the interval becomes $[0, LCM]$. The feasibility test has to check every deadline within this interval to provide a guarantee for the whole lifetime of the system.

The least common multiple is called hyper-period. It growths if periods share less large primes. If the tasks do not occur in strict periodic fashion, the hyper-period based test becomes insufficient: for example, if some tasks can be instantiated earlier than their period or later – though both deviations may be bounded – the number of job constellations that become possible

multiply by amount and size these deviations can occur with during the system's lifetime. In short: the interval $[0, LCM]$ will no longer contain all job constellations, it needs to be enlarged. Thus, the hyper-period based test becomes insufficient for a problem with non-periodic task timing behaviour.

Nevertheless, the hyper-period based test is generalised to non-periodic behaviour in case of preemptive systems: then solely the load becomes important. Maximal load is achieved if all tasks occur the earliest possible after the point of synchronicity, that is, if real-time feasibility is shown for a schedule with all tasks occurring synchronously and then earliest possible, other schedules with relaxed occurrence situations will be real-time feasible as well. The idea of considering only worst-case situations is followed by the theory of event streams.

Although the focus of this work is on preemptive real-time systems with dynamic priorities, it should be mentioned that the event stream model stated in the next section is also applicable for preemptive systems with static priorities, refer to [ABS2007].

2.2. Event Streams

As motivated in the previous section, considering only worst-cases reduces the computational complexity, still providing correct results, and the tested schedules are not theoretical but can occur at run-time. The theory of event streams follows the idea to consider only worst-cases but to abstract from schedules. Although the tested situations can occur at run-time, it is also possible they do not: this depends on the accuracy of the model parameters. The model of event streams is the first approximation from the system's real behaviour that is made in this thesis. The model is explained in the following.

The second approximation, which further reduces the computational complexity, is possible because of a property that event streams have. The second approximation is the topic of the next section.

First, let us define an event.

Definition 2.2.1. An event denotes a particular point in time when a task is triggered and instantiated: the time a job is created.

The following definition traces back to Gresser ([Gre1993]) though with another notation. The definition describes an abstraction from particular schedules to a list of worst-cases, represented by a sequence of ascending time spans – short: span[1] – with largest possible processing demand in terms of events.

Definition 2.2.2. For a task, an event stream is a sequence of real numbers where each number represents the amount of time to pass until as many jobs are created as the index of the sequence element belonging to that number denotes.
We write $(a^{(1)}, a^{(2)}, a^{(3)}, \dots)$ or $\left(a^{(n)}\right)_{n\in\mathbb{N}}$.
For all intervals $I = [t_0, t_1]$ on the time scale with $t_1 - t_0 = a_\tau^{(j)}$, at most j jobs of task τ will be created.

[1] In this thesis, the terms "span" or "time span" will be used. The reader may find other publications in which the terms "interval length", "time window size", or "time interval" are used for the distance between two points in time.

For all event streams, the number 0 is the element with index 1, because a single event corresponds to a point in time[2].

Example 2.2.3. The first three examples give a concrete transformation from task behaviour into abstraction. The last two illustrate common pitfalls.

a) Let τ_1 be a periodically triggered task, which is supposed to be once in every second. Then: between two events, there passes exactly one second; between three events, two seconds need to pass. The corresponding event stream is:

$$(a_{\tau_1}^{(1)}, a_{\tau_1}^{(2)}, a_{\tau_1}^{(3)}, \dots) = (0\,\text{ms}, 1000\,\text{ms}, 2000\,\text{ms}, 3000\,\text{ms}, 4000\,\text{ms}, \dots).$$

b) Let τ_2 be a periodically triggered task with the same period as τ_1, but now let there be deviations (jitter) such that the task can be triggered up to 100 ms earlier and later than the period. A sample trigger sequence is

$$(0, 1000\,\text{ms}, 2000\,\text{ms} + 10\,\text{ms}, 3000\,\text{ms} - 100\,\text{ms}, 4000\,\text{ms} - 50\,\text{ms}, 5000\,\text{ms}, \dots).$$

In case of a jitter, the minimal span for two events is period minus two times the jitter, for three events, it is two times the period minus two times the jitter, and for four events, it is three times the period minus two times the jitter. In our case, it is 800 ms for two events, 1800 ms for three events, and 2800 ms for four events, and we obtain as event stream:

$$(a_{\tau_2}^{(1)}, a_{\tau_2}^{(2)}, a_{\tau_2}^{(3)}, \dots) = (0, 800\,\text{ms}, 1800\,\text{ms}, 2800\,\text{ms}, \dots).$$

c) Let τ_3 be a task that is triggered by an external clock. Suppose, the clock should to give a tic every 600 ms, but proceeds faster: between two consecutive tics, it passes 99% of the time that passed between the two tics before. The manufacturer knows this information and has implemented a re-synchronisation every ten tics, after which the clock is correct for the following tic. Then between the first two tics, 600 ms will pass, and $600\,\text{ms} \cdot 99\% = 594\,\text{ms}$ pass between the second and the third tic. As consequence $0.99^8 \cdot 600\,\text{ms} = 548\,\text{ms}$ will pass from the next to last tic before clock correction. Thus, the event stream for the first ten events is

$$a^{(n)} = a^{(n-1)} + 0.99^{9-(n-1)} \cdot 600\,\text{ms} = \sum_{i=1}^{n} 0.99^{10-i} \cdot 600\,\text{ms}, n = 1, \dots, 10.$$

After the 10-th element the sequence repeats. We obtain for $10m + n \geq 2$:

$$a^{(10m+n)} = m \cdot \sum_{i=1}^{n} 0.99^{10-i} \cdot 600\,\text{ms} + \sum_{i=1}^{n} 0,99^{10-i} \cdot 600\,\text{ms}, 0 \leq n \leq 9.$$

d) The reader could assume that the occurrence specification in the event stream notation always corresponds to an existing schedule. This is not true, as Figure 2.1, page 22, depicts. The figure shows the transformation from a schedule into its event stream representation. For simplicity, all jobs are from the same task.

The intervals, which induce the elements $a^{(2)}$ and $a^{(3)}$, do not even intersect and the intervals with four and five events are shifted.

[2]Other works may introduce signal edges – e.g. rising times – as events, which need non-zero time to raise.

e) In a similar way, we could think that event densities – a density being the number of events divided by time to pass – decrease monotonically. As shown in this example, this does not hold.

In Figure 2.2, page 22, an event stream is drawn, which has the elements $a^{(1)} = 0$, $a^{(2)} = 1$, $a^{(3)} = 4$, $a^{(4)} = 5$, and in general for $n = 1,2,3,\cdots$: $a^{(2n-1)} = 4n-4$ and $a^{(2n)} = 4n-3$. Therefore, the density for an even number of events is less than the density for an odd number. Furthermore, the density approaches 2 for $n \to \infty$.

As demonstrated in the examples, event streams allow to model very complex task occurrence. If there is no specification describing how tasks occur, but the task set as a whole is available as implementation or some kind of automaton, then automatic extraction of event streams can be done, as shown in [WMT2005] and [BAS2006]. The latter uses event dependency graphs to reduce the complexity of the extraction algorithms. Time is annotated to basic blocks – sequential code-like bodies of loops or alternatives of decisions. All basic blocks of a program that are placed between two blocking commands are considered as one task. Examples for blocking commands are those that perform communication, for instance with other devices or other software. Via the annotated times and the blocking commands, an event sequence is generated: it represents the times the commands' receiver is invoked. With this event sequence the event stream for the receiver is calculated.

2.3. Approximating Event Streams

Event streams have a property allowing to weigh precision against the computational complexity while preserving demand overestimation. The definition of the approximation is preceded by a simple but important fact.

Lemma 2.3.1. *Let $\left(a^{(n)}\right)_{n\in\mathbb{N}}$ be an event stream, then for all $a^{(n)}, a^{(m)}$ the following holds:*

$$a^{(m)} + a^{(n)} \leq a^{(m+n)} \tag{2.1}$$

$$a^{(m)} + a^{(n)} + a^{(2)} \leq a^{(m+n+1)} \qquad \text{if } a^{(2)} \neq 0. \tag{2.2}$$

This property is known as subadditivity.

Proof. For two arbitrary natural numbers n and m, let us assume the contrary of the first statement. Then there exists in the schedule an interval $I_{m+n} = [t_0, t_1]$, with span $t_1 - t_0 = a^{(m+n)}$, which contains $n+m$ events. We now count n events from the beginning of the interval, t_0, up to a point in time x, at which we find the n-th event. We define $I_n := [t_0, x]$. Its length must be equal to or be greater than $a^{(n)}$, otherwise, $a^{(n)}$ would have been wrongly constructed. The interval from x up to the endpoint t_1 covers $m+1$ events. By the assumption, its length is less than $a^{(m)}$ which says that $a^{(m)}$ cannot be the shortest span to let m events happen: the element $a^{(m)}$ must have been wrongly constructed.

For the second assertion, we count from the start t_0 n events and reach again point x, and we count from endpoint t_1 towards the start t_0 m events and reach a point $y \in I_{m+n}$: it is y strictly greater than x. In the interval $I' :=]x, y[$ – both x, y excluded – there are no events, and it follows $y - x \geq a^{(2)}$. □

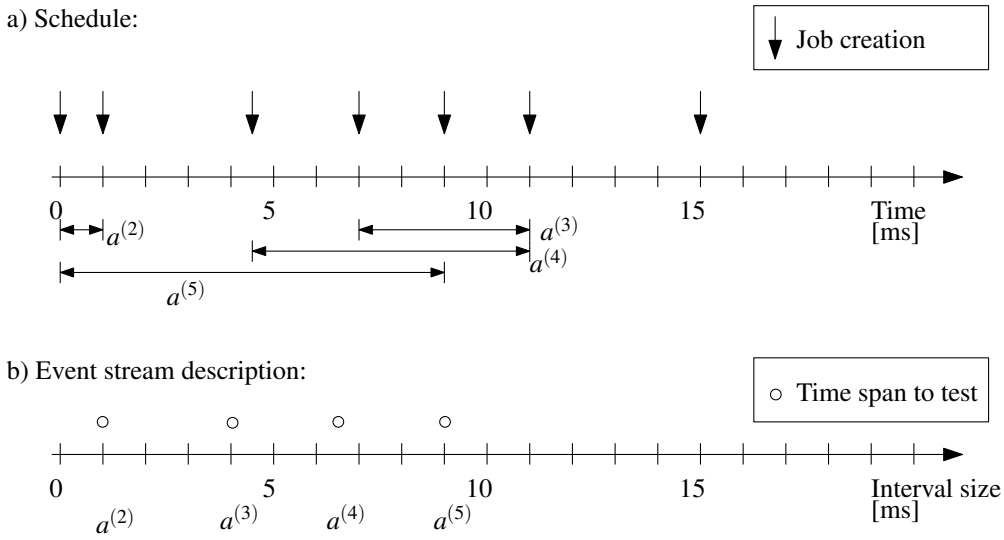


Figure 2.1.: Transformation From Explicit Schedule to Event Stream Description

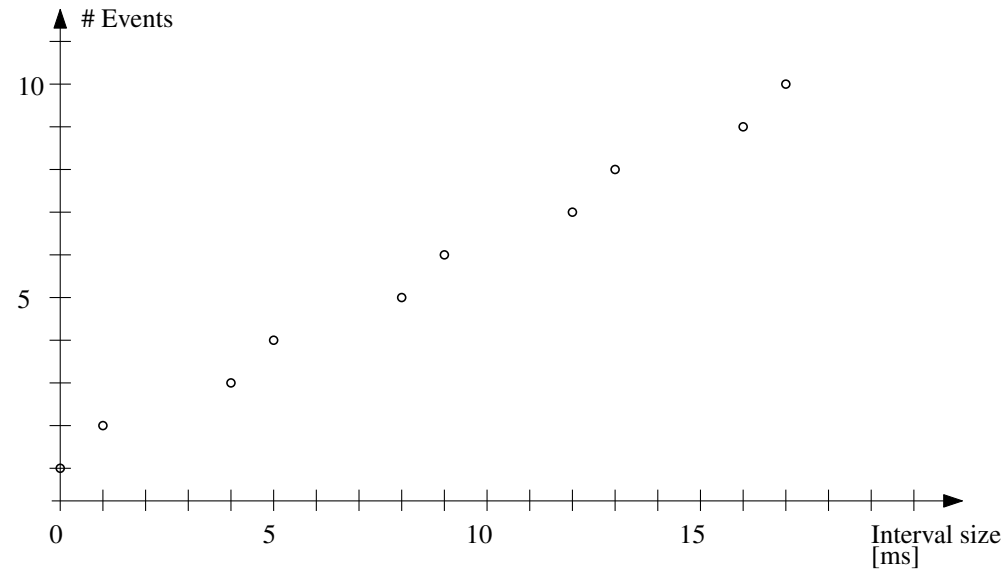


Figure 2.2.: Alternating Event Densities

This property establishes an approximation – a linearisation – which was first introduced in [AS2004].

Definition 2.3.2. Let $\left(a^{(n)}\right)_{n\in\mathbb{N}}$ be an event stream. Exists an index k and $s > 0$, such that

$$s \geq \frac{m-k}{a^{(m)} - a^{(k)}} \quad \text{for all } a^{(m)}, \quad m > k, \tag{2.3}$$

then we call s an approximation of $\left(a^{(n)}\right)_{n\in\mathbb{N}}$ at index k, and k is called approximation index.

Does the in-equation hold for a number s and for every index out of an index set I, then we call s an approximation of $\left(a^{(n)}\right)_{n\in\mathbb{N}}$ for the set I. Is the index set $I = \mathbb{N}$, then we call s an approximation of $\left(a^{(n)}\right)_{n\in\mathbb{N}}$.

In the case $a^{(2)} > 0$, it follows from (2.2) that $s := \frac{1}{a^{(2)}}$ is an approximation of $\left(a^{(n)}\right)_{n\in\mathbb{N}}$. For further illustration, the following examples are given.

Example 2.3.3. We want to develop approximations of $\left(a^{(n)}_{\tau_1}\right)_{n\in\mathbb{N}}, \left(a^{(n)}_{\tau_2}\right)_{n\in\mathbb{N}}, \left(a^{(n)}_{\tau_3}\right)_{n\in\mathbb{N}}$, taken from Example 2.2.3.

a) In the case of τ_1, we know that $a^{(n+1)}_{\tau_1} - a^{(n)}_{\tau_1} = 1000\,\text{ms}$ for all $n \in \mathbb{N}$, and this leads to $s = \frac{1}{a^{(2)}_{\tau_1}} = \frac{1}{1000\,\text{ms}}$ as an approximation of $\left(a^{(n)}_{\tau_1}\right)_{n\in\mathbb{N}}$.

b) In the case of τ_2, it holds $a^{(n+1)}_{\tau_2} - a^{(n)}_{\tau_2} = 1000\,\text{ms}$ for all $n \geq 2$, and with $s = \frac{1}{a^{(3)}_{\tau_2} - a^{(2)}_{\tau_2}} = \frac{1}{1000\,\text{ms}}$ we obtain an approximation of $\left(a^{(n)}_{\tau_2}\right)_{n\in\mathbb{N}}$ for the set $\{n \in \mathbb{N} : n \geq 2\}$.

c) In the case of $\left(a^{(n)}_{\tau_3}\right)_{n\in\mathbb{N}}$, the difference between two consecutive sequence elements increases and it resets to 548 ms every ten elements. Thus, a first approximation of $\left(a^{(n)}_{\tau_3}\right)_{n\in\mathbb{N}}$ with $s := \frac{1}{548\,\text{ms}}$ will hold for every index. But a restriction to certain index elements, namely $n = 9 + 10m$, $m \in \mathbb{N}$, yields a second approximation with $s := \frac{10}{5189}$. The second approximation yields no error at its index elements. However, the first yields a 5% error because it predicts ten events for an interval size of 4932 ms, in contrast to the event stream element with the index 10 which is 5189 ms.

In Figure 2.3, page 24, suggestions to approximate the event from Example 2.2.3(e) are depicted. The curves g_1, g_2, g_3, g_4 have all the same slope $s_g := \frac{1}{a^{(2)}}$; the curves h_1, h_2 have each the slope $s_h := \frac{1}{a^{(5)} - a^{(2)}}$. The curve family g_1, g_2, g_3, g_4 over-approximates. The curve h_2 underestimates test points at 1, 6, 11, and 16, thus it leads to infeasible results: the reason is that it started at the wrong event stream element. The curve h_1 starts at the correct sequence element and either hits test points without error or overestimates them.

The next statement supports the usage of the fast real-time feasibility test in this work.

Remark 2.3.4. *If for a task at most only a finite number of jobs are created at the same moment, then its event stream can be approximated: every event stream can be approximated if it contains a positive element with a finite index.*

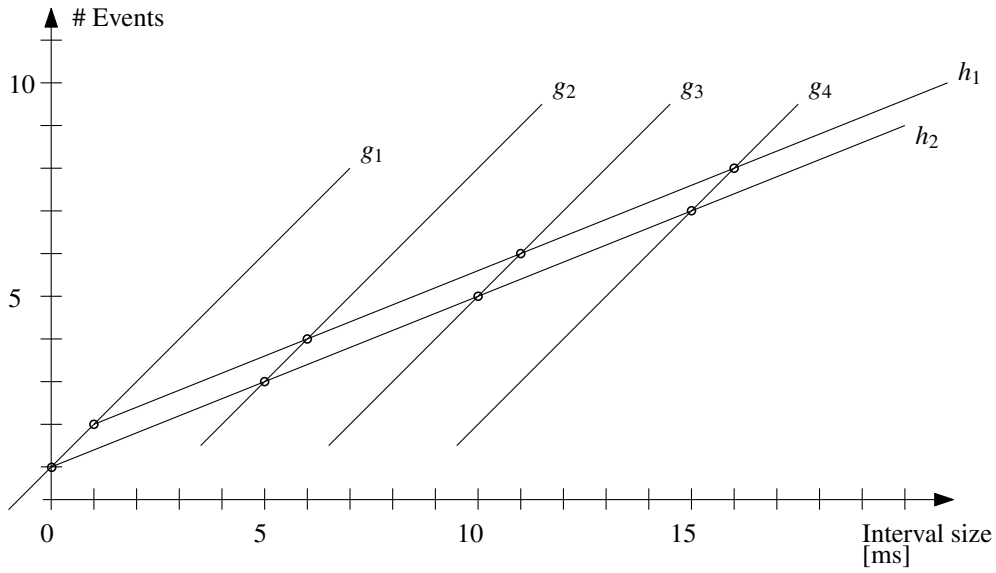


Figure 2.3.: Likely Overestimations of an Event Stream

Proof. Let $\left(a^{(n)}\right)_{n\in\mathbb{N}}$ be an event stream. Let $k \in \mathbb{N}$, $k < \infty$, with $a^{(k)} > 0$ and $a^{(k-1)} = 0$. Define

$$b^{(j)} := \begin{cases} a^{(j)} & \text{for } 0 < j < k, \\ a^{(k)} + a^{(1+j-k)} & \text{for } k \leq j < 2k, \\ m \cdot a^{(k)} + a^{(1+j-m\cdot k)} & \text{for } m \cdot k \leq j < (m+1) \cdot k, m > 1. \end{cases}$$

Then for every $b^{(j)}, j > 0$, according to Lemma 2.3.1, the in-equation $b^{(j)} \leq a^{(j)}$ holds.

The sequence $\left(b^{(n)}\right)_{n\in\mathbb{N}}$ can be approximated by $s := \frac{k-1}{b^{(k)}}$ at index $k-1$.

We conclude thereof the approximation of $\left(a^{(n)}\right)_{n\in\mathbb{N}}$. □

The remark is necessary to limit the application of the approximation. Since all practical situations and task implementations have a limited number of task instantiations at the same instant, the remark shows that the event stream approximation is practicable.

2.4. Demand Bound Function

Deadlines represent the constraints that a real-time feasibility has to test. The key idea behind the demand bound based real-time feasibility test is to restrict to those deadlines, that are to meet within a given interval. Deadlines beyond the interval are postponed to the test on larger intervals.

Clearly, because of preemption, it is only necessary to test those intervals having maximal processing demand compared to others of the same size.

Definition 2.4.1. Let Γ be a set of tasks assigned to the same processor. The demand bound function $\mathrm{dbf}^{\#} : \Gamma \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ denotes for every span $\Delta t \geq 0$ the maximal needed processing demand by instances of a task $\tau \in \Gamma$.
(See [BCGM1999])

Let us take the synchronicity assumption. The summed up processing demand of every task then yields the real-time test:

Theorem 2.4.2. *Let Γ be the set of tasks to be run on the same processor. Then:*

a) The function

$$\begin{aligned} \mathrm{dbf} : \mathbb{R}_{\geq 0} &\to \mathbb{R}_{\geq 0} \\ \Delta t &\mapsto \mathrm{dbf}(\Delta t) := \sum_{\tau \in \Gamma} \mathrm{dbf}^{\#}(\tau, \Delta t) \end{aligned} \tag{2.4}$$

is an upper bound on the task set's processing time: the task set Γ demands for intervals of length Δt at most $\mathrm{dbf}(\Delta t)$ time units of processing time.

b) The system is real-time feasible, if

$$\mathrm{dbf}(\Delta t) \leq \Delta t \quad \textit{for all } \Delta t \geq 0. \tag{2.5}$$

(See [BCGM1999])

For the calculation of the demand bound function, a per-task estimation on the processing demand is needed. This can be achieved in combination of the tasks' event streams and their supposed execution time.

Execution times of tasks running on special purpose processors can be estimated by software tools like *AbsInt* [Abs2009], *Cinderella* [LMW1999], and *ChronEst* [Chr2009], where the former two are analyses yielding upper bounds on execution times, and the latter is a simulation. Depending on implementation and specialisation on processor models, the estimates hold or can be used on other configurations. This is not part of this work. Here, a per-task maximal – never to be exceeded – execution time is assumed to be given and called worst-case execution time.

For each span, let us sum up the number of jobs to finish within and multiply each number of jobs of the same task with the task's worst-case execution time:

Let $I = [t_0, t_1]$ be an interval of length $\Delta t = t_1 - t_0$. The number of jobs of a certain task to finish within the interval I is bounded according to the task's event stream specification: the index of the largest sequence element not exceeding Δt gives the maximal number of created jobs within the interval I – let this number be j. Now, every job has a deadline, say d_τ, which is relative to the job's creation time: if the j-th job starts at time t_1, then the job is to finish until $t_1 + d_\tau$. Thus, it is j the maximal number of jobs of task τ to finish within any interval of size $\Delta t + d_\tau$.

Let us define this more general:

Definition 2.4.3. Let Γ be the set of tasks to be run on the same processor and let $\left(a_\tau^{(n)}\right)_{n\in\mathbb{N}}$ be the event stream of $\tau \in \Gamma$, then define the delayed event function

$$\begin{aligned} m : \Gamma \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} &\to \mathbb{R}_{\geq 0} \\ (\tau, \delta, \Delta t) &\mapsto m(\tau, \delta, \Delta t) := \max\{j \in \mathbb{N}_0 : a_\tau^{(j)} + \delta \leq \Delta t\}, \end{aligned} \tag{2.6}$$

with $a_\tau^{(0)} := -\infty$.

The delayed event function is a non-negative step function which gives the maximal number of events occurring within any interval of size $\Delta t - \delta$.

Example 2.4.4. For a task τ,

a) let $\delta = 0$, then the event function gives the maximal number of events generated within any interval of length Δt;

b) let $\delta = d_\tau$, then the event function gives the maximal number of jobs to complete within any interval of length Δt. This matches the definition of the event function η in [BCGM1999].

Now, let us multiply the number of jobs to complete by the amount of processing time one job maximally needs. This gives an upper bound on the processing demand:

Lemma 2.4.5. *Let Γ be a set of tasks to be run on the same resource. For each task τ, let d_τ be the deadline, c_τ be the worst-case execution time, and $\left(a_\tau^{(n)}\right)_{n\in\mathbb{N}}$ be the event stream, respectively. Then the function*

$$\begin{aligned} \mathrm{D} : \mathbb{R}_{\geq 0} &\to \mathbb{R}_{\geq 0} \\ \Delta t &\mapsto \mathrm{D}(\Delta t) := \sum_{\tau\in\Gamma} m(\tau, d_\tau, \Delta t) \cdot c_\tau, \end{aligned} \tag{2.7}$$

overestimates the processing demand for all intervals of size

$$\Delta t \in \bigcup_{\tau\in\Gamma} \bigcup_{j=1,2,3,\ldots} \{a_\tau^{(j)} + d_\tau\}.$$

The function D *is called approximated demand bound function.*
(See [AS2004])

Proof. Suppose the opposite: let $\Delta t > 0$ and $\mathrm{D}(\Delta t) < \mathrm{dbf}(\Delta t)$. Then there exists some task τ with

$$\mathrm{dbf}^{\#}(\tau, \Delta t) > m(\tau, d_\tau, \Delta t) \cdot c_\tau = \max\{j \in \mathbb{N}_0 : a_\tau^{(j)} + d_\tau \leq \Delta t\} \cdot c_\tau.$$

Suppose the max-term yields k jobs to finish, and suppose worst-case execution time c_τ and deadline d_τ are correct, then divide the processing demand, which the demand bound function calculates, by worst-case execution time c_τ. This yields more jobs to finish then the max-term:

$$\frac{\mathrm{dbf}^{\#}(\Delta t)}{c_\tau} > k.$$

The max-term subtracts deadline from interval size – it only recognises the number of jobs to finish. Therefore, the number of jobs to finish and the corresponding element of the event stream must have been wrongly determined. □

Note, in the lemma, the term "approximation" stands for the first performed approximation, which was done by means of event streams (Subsection 2.2).

The use of the event function in the lemma represents the most general event stream characterisation. The function becomes explicit by explicit given event streams, like those in Example 2.2.3. The additional element $a_\tau^{(0)}$ is for computational reasons, as without the element, the max-term would yield nothing for zero as input.

Example 2.4.6. In the case of a strictly periodic task system with individual deadlines, less or equal to the periods, the approximated demand bound function becomes:

$$\mathrm{D}(\Delta t) = \sum_{\tau\in\Gamma} \left(\left\lfloor \frac{\Delta t - d_\tau}{T_\tau} \right\rfloor + 1 \right) c_\tau,$$

here, for a task τ, it denotes T_τ the period, d_τ the deadline, and c_τ the worst-case execution time.

This is the real-time feasibility test suggested by Baruah *et al.* ([BRH1990], page 312f).

The following real-time test is a consequence of Lemma 2.4.5 and Theorem 2.4.2.

Theorem 2.4.7. *With the same assumptions as for the lemma above, a task set is real-time feasible, if*

$$\mathrm{D}(\Delta t) \le \Delta t \quad \textit{for all } \Delta t \in \bigcup_{\tau\in\Gamma} \bigcup_{j=1,2,3,\ldots} \{a_\tau^{(j)} + d_\tau\}. \tag{2.8}$$

(See [AS2004] and [LAS2006].)

Several possibilities exist to limit the test point quantity. In the case of a strict periodic task set, the test bound can be set equal to the hyper-period. After the hyper-period, because of the synchronicity assumption, the pattern of task occurrences will repeat. As noted earlier in this chapter, this way has disadvantages. On the one hand, the hyper-period can grow very large, especially when some periods are prime numbers or have large prime divisors. On the other hand, in presence of task jitter, the occurrence pattern will not repeat after the hyper-period, and in cases of more complex task triggers, like 2.2.3(c), it is not canonical to determine a hyper-period.

Here, the definition of a suitable test bound makes use of the approximation of event streams – the second mentioned approximation as of Subsection 2.3 – which is proven to always exist, according to Remark 2.3.4.

Theorem 2.4.8. *Let us assume the same as for Lemma 2.4.5. Additionally, for each task $\tau \in \Gamma$, let s_τ be an approximation at index k_τ of the respective event stream. Then, no deadline will be violated if*

$$\mathrm{D}(\Delta t) = \sum_{\tau\in\Gamma} h_\tau(\Delta t)\cdot c_\tau \le \Delta t \qquad \textit{for } \Delta t \le M, \tag{2.9}$$

$$\sum_{\tau\in\Gamma} s_\tau \cdot c_\tau \le 1,$$

$$\begin{aligned} &\textit{with} && M := \max_{\tau\in\Gamma}\{a_\tau^{(k_\tau)}\} \\ &\textit{and} && h_\tau := \begin{cases} 0 & \textit{for } \Delta t \le d_\tau, \\ m(\tau, d_\tau, \Delta t) & \textit{for } d_\tau < \Delta t < a_\tau^{(k_\tau)}, \\ k_\tau + (\Delta t - a_\tau^{(k_\tau)} - d_\tau)\cdot s_\tau & \textit{for } a_\tau^{(k_\tau)} \le \Delta t. \end{cases} \end{aligned} \tag{2.10}$$

In the case of a real-time feasibility test, we call the index of the sequence's element at which an event stream is approximated the test index.

A similar theorem was first given in [AS2004]. In the present form, it can be found in [LAS2007], which also contains the discussion on the hyper-period above.

Proof. For $x \le M$, the theorem's assertion is the same as Theorem 2.4.7. For $x > M$, $D(x)$ is a sum of linear functions: if derived the sum to x, we obtain the theorem's second statement. □

With the help of the theorem, we are able to determine the processing demand at linear complexity for all points in time. For example in [LAS2007], the number of test points was decreased from about 78.000 down to 20 without introducing errors. Therefore, the theorem enables determination of degrees of freedom for power saving methods at low complexity.

2.5. Energy Consumption of Software

Hardware devices consume electric energy and convert it into thermal energy and electric charge. It is the software which decides when this process happens. This section starts with a description of processor power dissipation and moves over to power dissipation that is caused by software, finally, the section describes ways to reduce power consumption. Methods for lowering power consumption applied at chip design time, like slowing down non-critical gate paths, are not applied in this thesis. The reader may find information regarding these kinds of optimisation in one of [NM1997], [PR2002], or [Pig2005]. It is assumed, that all on-chip, not later modifiable power optimisation has taken place at chip design time. Here, only those methods are taken into account, which can be modified at run-time, or be programmed before, e.g. via register values.

2.5.1. Power dissipation

The combination of execution time and the processor's power consumption determines the energy demanded from the source. The power divides into dynamic power dissipation, short circuit, and static power dissipation. In the following, the first two kinds are explained at a CMOS-inverter and the third at an NMOS-transistor. Both devices are common design elements for manufacturing digital circuits.

Figures 2.4 and 2.5 depict a CMOS-inverter consisting of an NMOS-transistor, T_N, a PMOS-transistor, T_P, and a capacitor, C_{load}, which corresponds to the circuitry the inverter has to drive. It is V_{dd} supply voltage potential. The potential V_{PB} is connected to the bulk of the PMOS-transistor, and it may be equal to V_{dd}, but alternatively, it may be different which is called body biasing and is described later. In the same manner, the potential V_{NB} is connected to the bulk of the NMOS-transistor, and in absence of body biasing, the potential equals ground potential. The

grey arrow in both figures depict the flowing current when the input changes from low to high potential. This is now described in detail.

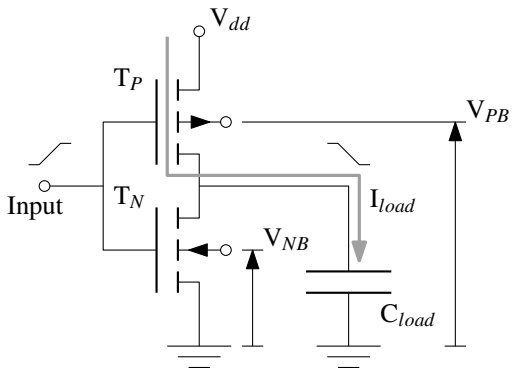


Figure 2.4.: CMOS Inverter, Loading Capacities

The first kind of power dissipation occurs while the inverter changes its state. Transformation of electric energy into thermal energy while capacities are charged or discharged is called dynamic power dissipation. It occurs whenever the processor changes its state, e.g. at a clock edge introducing a new cycle. Capacities represent driven gates, for example, connected transistors and wires. At both transistors shown in Figure 2.4, electric energy will be transformed into thermal energy, corresponding to $E = \frac{1}{2} C_{load} \cdot V_{dd}$. This thermal energy arises while loading the capacity and while unloading the capacity: dynamic power adds up to

$$P_{\text{dyn}} = \alpha \cdot C V_{dd}^2 f, \tag{2.11}$$

with clock frequency f, supply voltage V_{dd}, total circuit capacity C, and the load switching fraction α. Thus, dynamic power dissipation is proportional to frequency and the square of the supply voltage: if voltage or frequency decrease, so will the power dissipation.

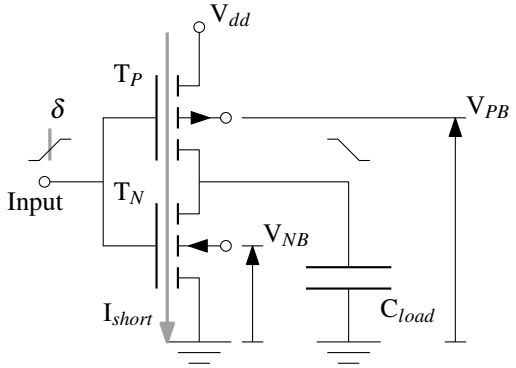


Figure 2.5.: CMOS Inverter, Short Circuit Phase

Gate state transition causes a short in the circuit lasting a small moment in time. This is the second kind of power dissipation, and it occurs because both transistors are conducting at the same instant, see Figure 2.5. While the input edge is rising, the open transistor starts closing and

the other one starts opening. This causes an electric current through both transistors from supply voltage potential down to ground, and the current lasts while both transistors have not reached their final state: sharper signal edges will reduce short circuit power dissipation.

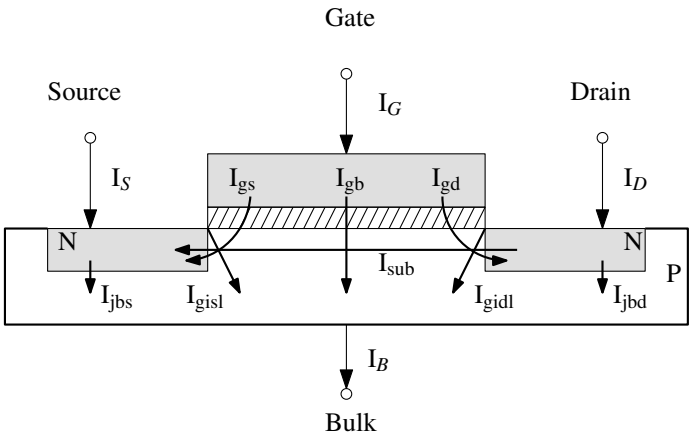


Figure 2.6.: Leakage Currents in an NMOS-Transistor, Based on: [RMMM2003]

Static power dissipation is the last kind of power dissipation. It occurs always when the processor is voltage supplied. Currents 'leak' through the material beside the desired flow of electrons. Figure 2.6 depicts all electric currents flowing in a metal-oxide semiconductor transistor. The currents are identified as:

a) I_{sub}: sub-threshold current, flowing although the transistor channel is closed.

b) I_{gs}, I_{gb}, and I_{gd}: tunnelling currents from the gate through the oxide into source, bulk and drain.

c) I_{jbd}, I_{jbs}: currents flowing through the p-n junctions in reverse direction.

d) I_{gidl}, I_{gisl}: parts of I_{jbd} and I_{jbs}, respectively, induced by the gate.

According to [MR2003], subthreshold leakage is the dominating leakage current for 90-nm technologies, but gate leakage becomes the same important for 50-nm – now 45-nm. In their report for 2007 [ITR2007], the ITRS considers leakage currents a major design challenge.

Several sources ([CYVA1996], [FP2002], page 399, and [FF2005], page 3-6) describe subthreshold leakage for an NMOS transistor similar to

$$I_{\text{sub}} = \alpha e^{\frac{V_{\text{GS}}-V_{\text{TH}}}{nV_t}} \left(1 - e^{-\frac{V_{\text{DS}}}{V_t}}\right), \tag{2.12}$$

with V_{TH} the threshold voltage, V_{GS} gate-to-source voltage, V_{DS} drain-to-source voltage, V_t thermal voltage ($V_{\text{DS}} >> V_t$), n a parameter depending on depletion and oxide capacitances, and with α a constant depending on used material and transistor dimensions. In reference [FF2005], the factor α is multiplied by V_t^2, that is, a temperature dependency is included as well. Noticeable is that the today major leakage current exponentially depends on the distance from threshold

voltage to Gate-to-Source voltage: the supply voltage. Additionally, reference [FF2005] states an exponential relation of supply voltage to gate induced leakage currents and reverse junction currents (*cf.* pages 3-10, 3-11). We conclude: a reduction in supply voltage will reduce power dissipation due to leakage.

In the surroundings of the processor, other sources of power dissipation can be found. Summarised as I/O-power dissipation, the power to drive peripherals can be considered. This highly depends on the peripherals to drive and can therefore not suitably be determined with the processor as solely base. Additionally, driven circuitry has own voltage levels, which must not be modified: modifications may lead to violated electrical specifications for interconnections or misinterpreted signals in peripheral circuits. In this work, this kind of power dissipation is not considered for the optimisation of battery operating life. Any optimisation or design space exploration in this direction is assumed to be done beforehand.

2.5.2. Power Consumption of Software

Already on first sight, we see a major influence of software to an embedded system's energy consumption: run-time. If we think of a radio as a processor, then the software defines the duration of sending; in cases of software-defined radio, the software defines the radio's frequency and intensity. The major components to determine the electric energy of a single transmission are intensity, duration, and frequency. A single transmission corresponds in this context to process one job of the software task 'send message'. For a common processor, duration and frequency become major factors influencing a task's power consumption, this is dealt with in the next subsection. In the following, let us look at intensity.

For a multi-core architecture, intensity can be seen as how each core is utilised. Utilisations can be useful to optimise power consumption, and the authors of [ESP2006] follow this approach. But utilisation is no slack indicator for hard real-time tasks because a deadline can be tight and the utilisation can be 50%, both at the same time. Therefore, this abstraction level is of no use in this work.

A processor splits into several parts, and different instructions can use different parts: this is the intensity addressed here.

The energy demand to process an instruction differs from demands of other instructions. The foregoing instruction can also have an effect as it steers the state in which the processor is left for the next one, this is called inter-instruction overhead. Research has been done in these matters and is reported in the following paragraphs.

Data dependencies can be calculated as well, but data is usually hardly to predict and because of this it is considered to be arbitrary.

A descend to instruction level was done in [TMW1994]. The authors research the influence of different instructions on the power consumption for general purpose processors: the Intel 486DX2, and the Fujitsu SPARClite 934. They provide an instruction level power model. For every instruction (assembly language), they provide base cost and include inter-instruction overhead by adding an additional constant value, which lies in a 5% range of the base cost. Their approach also considers cache misses, which results in penalty values for estimated misses. They report an error of about 3% from their instruction based estimation to the later measured value of the whole program.

On special purpose processors, inter-instruction overheads can have considerable variations for different instruction pairs. A factor of 10 was reported in [LTMF1995], Table 4. For the measured microcontroller, the instructions' base cost was reported as high as the inter-instruction overhead (ibid[3], Table 3). Again, only a small error was observed, when comparing the instruction based estimation with measurements of the whole program.

Processor soft-cores are hardware descriptions realising a universal processor for configurable hardware such as field programmable gate arrays (FPGA). Even for those soft-cores, instructions can be distinguished into different categories, yielding estimations with small errors compared to measurements of a program as a whole: the authors of [OP2004] report an error of about 7.8%. They use hardware simulation and report a computational complexity that is high.

The idea to sum up energy quantities for each instruction to obtain reliable energy estimations of the whole program has two drawbacks. First, the approach does only hold, if the program's execution path can be reliably foreseen. Otherwise the difference between worst-case and average case will have large variations, and errors will be inherited: the problem is similar to the estimation of worst-case execution times.

The second drawback is that base cost for every instruction and overhead values for every instruction pair need to be measured. An idea to circumvent this problem is functional level modelling, which is presented in [JLSM2003]. The authors propose to determine energy estimations by analysing which part of the processor is involved during execution. By measurement, each part is assigned a value denoting its power consumption. Multiplication of the part's power value with its usage time yields the amount of energy used for execution. All these energy portions summed up yield an estimation on the energy consumption of the task observed. The functional level approach is reported in [LJSM2004] to have a maximal error of 8%, considering different processors and benchmark programs.

A similar approach is achieved by assigning different states of power consumption to the resource and adding edges between the states. An edge markings denotes the cost for transition. These automata are called power state machines, refer to [FP2002], page 375. If a usage profile is known for each task and state or can be obtained, then the switching cost and the overall energy estimation can easily be determined. The calculation is an addition, as in the paragraph before, but now it includes transition cost as well.

2.5.3. Methods for Power Saving

There are two kinds of methods for saving energy that affect the available processing time and that are modifiable at run-time. On the one hand, the clock frequency can be regulated together with supply voltage, and on the other hand, the duration the processor is in use can be shortened by shutting it down in parts or in the whole.

The change of the clock frequency is based on the formula for the circuit delay δ_{circuit}

$$\delta_{\text{circuit}} = \frac{V_{\text{dd}}}{(V_{\text{dd}} - V_{\text{th}})^{\alpha}}, \tag{2.13}$$

with V_{dd} as supply voltage and V_{th} as threshold voltage, α a (hardware) technology dependent parameter in the range of $1 < \alpha < 2$, see [MM2002], page 270. A decrease of the supply voltage

[3] Ibid means same reference.

will increase the time for a signal to be delivered, which is the same as to lower speed. An increase of the threshold voltage achieves the same. The following two paragraphs describe either aspect.

The first idea is known as dynamic voltage scaling (DVS) and was first studied on system level to slowdown a processor in [IY1998]. It saves energy, because lowering the processor's supply voltage will lower its power consumption, as is derived from Subsection 2.5.1. A lower supply voltage has a lower processing frequency as consequence.

The second aspect is called adaptive body biasing (ABB), first studied on CMOS-level in [KFM+1996]. Here, the bulk potentials of the transistors are modified to increase their threshold voltage and thereby to reduce power dissipation caused by leakage currents. (See [FP2002], page 405.). As observed in Equation (2.13) increased threshold voltage results in decreased frequency.

Both techniques do not touch a task's context: all register values are retained. Both reduce power dissipation by reducing processing speed. Under the requirement that deadlines are retained, any spare time may be exploited, thus, a lowest frequency is the primal optimisation goal.

Although scaling of supply voltage and body biasing reduce power, and thus reduce temperature and leakage, both methods only affect the processor when it processes software. If the processor is idle, the processor can yet be set to lowest operating frequency, but power consumption will be kept at a certain constant non-zero level.

A processor that is switched-off when it is idle will have a power consumption of zero: that is instead of adjusting the frequency of the idle processor, send it into a low power mode with reduced power dissipation. It comes at the cost of reduced processing ability, and a penalty for back-transition into normal mode. A processor can have multiple low-power modes, and the mode to use is yet to be determined.

Table 2.1.: Categorisation of Low-Power Modes

Mode	Context is	Program timer	Wake-up transition	Sleep transition
run	kept	no	–	–
sleep	kept	no	a)	–
stop	kept	yes	a), b)	–
standby	discarded	yes	a), b), c)	–
standby (context retained)	stored	yes	a), b), c), d)	store context

Consider the processor partitioned into a timer, interrupt wires, registers holding task context information, and a processing unit covering the rest. The low-power modes can now be categorised: first, the clock of the processing unit is stopped; second, the voltage of the processing unit is set to zero; third, the registers are switched off. In the latter case it is important to know if register contents need to be restored after wake-up, as this will further increase transition times. The categorisation of power modes can be seen in Table 2.1. The third column denotes whether a timer has to be programmed beforehand to ensure proper wake-up. The fourth column denotes

what has to be done to bring the device up again; the letters have the following meaning:

a) The clock frequency needs to be stabilised.

b) The supply voltage needs to be stabilised.

c) The registers (and other logic) need to settle.

d) The software context needs to be restored.

The fifth column states what has to be done before the processor may safely be switched into low-power mode.

Table 2.2.: Electric Currents and Transition Times of the STM32F103 Microcontroller's Power Modes

Mode of operation	Supply current	Transition time to *run*-mode
run @ 72 MHz (peripherals on)	50 mA	-
run @ 72 MHz (peripherals off)	32.8 mA	-
sleep @ 72 MHz (peripherals on)	30 mA	1.8 μs
sleep @ 72 MHz (peripherals off)	7.5 mA	
stop	24 μA	3.6 μs / 5.4 μs
standby	1.7 μA	50 μs

Example 2.5.1. For the STM32F103 microcontroller (see [ST 2007]), Table 2.2 shows values for supply currents in different power modes and transition times for mode changes. The controller can be shut down in parts or in the whole, in three stages. Stage one is called *sleep*-mode and disconnects the processor from the clock generator, which allows a fast transition back to *run*-mode because the clock generator is still operating. Stage two is called *stop*-mode and additionally disconnects the processor from supply voltage. The last stage is called *standby*-mode and also stops the clock generator. In all cases, the back-transition to *run*-mode can be gained by special pins waiting for signal change or by interrupt, for example by an external power supplied real-time clock (RTC) that has been programmed before.

Besides the example, some microcontrollers allow to shut down on-chip peripherals, like analog-to-digital converters or USB-interfaces.

The processor must be woken up by an external signal. Here, in contrast to voltage scaling, where the processor changes the speed at run-time, the processor is the passive component. Thus knowledge on how long and how deep the processor can be shut down safely is needed.

Break-even time

In cases where mode transitions imply significant cost, this has to be incorporated into an optimisation method which is to reduce the energy consumption. The decision whether to shutdown the processor depends on the break-even time, that is the duration after which the energy saved is as high as the mode-transition cost.

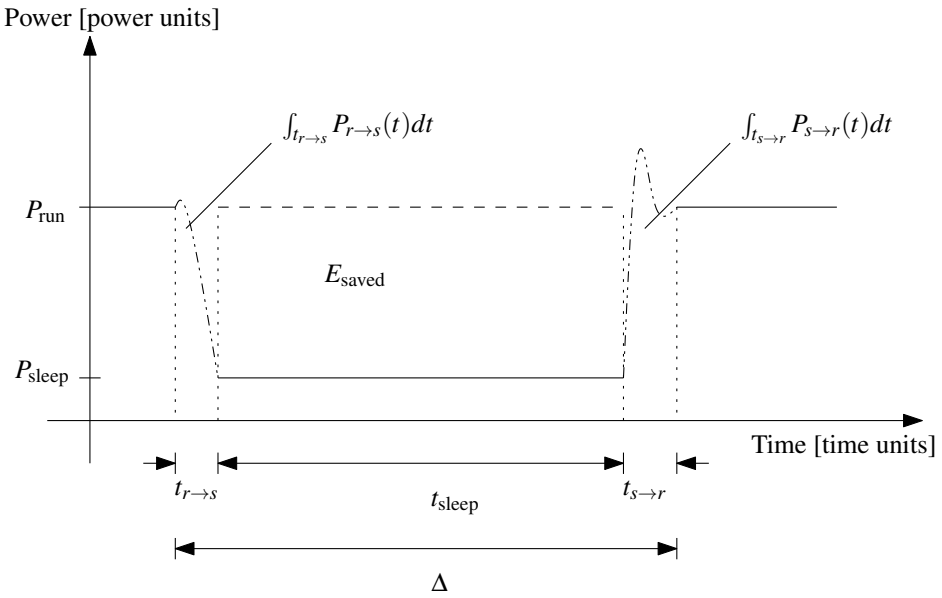


Figure 2.7.: Power Consumption by Time Diagram for Duration of Mode-Switch

In Figure 2.7 the energies for one mode switch are shown. Let us summarise the time needed for switching back and forth with $t_{\text{switch}} := t_{r\to s} + t_{s\to r}$, and let us summarise the energy needed with

$$E_{\text{switch}} := \int_{t_{r\to s}} P_{r\to s}(t)dt + \int_{t_{s\to r}} P_{s\to r}(t)dt.$$

For the energy saved, it must hold:

$$E_{\text{saved}} \geq \max\{0, E_{\text{switch}} - P_{\text{run}} \cdot t_{\text{switch}}\}. \tag{2.14}$$

Inserting $E_{\text{saved}} = (P_{\text{run}} - P_{\text{sleep}}) \cdot t_{\text{sleep}}$ yields

$$t_{\text{sleep}} \geq \max\left\{0, \frac{E_{\text{switch}} - P_{\text{run}} \cdot t_{\text{switch}}}{P_{\text{run}} - P_{\text{sleep}}}\right\}.$$

Adding t_{switch} to both sides gives

$$\Delta \geq t_{\text{be}} := \max\left\{t_{\text{switch}}, \frac{E_{\text{switch}} - P_{\text{sleep}} \cdot t_{\text{sleep}}}{P_{\text{run}} - P_{\text{sleep}}}\right\} \tag{2.15}$$

as necessary requirement for the duration of shutdown, Δ, with the break-even time t_{be} as lower bound.

The break-even time is a central point. Another one is guaranteeing hard real-time feasibility, which breaks several of the online algorithms known in the literature.

When we consider low-power modes, the following questions are to be covered by algorithms for energy optimisation:

a) When to transit into low-power mode?

b) How long to stay in low-power mode?

c) Which knowledge is required for determination of the two questions above?

d) When to wake-up? Deadline guarantees?

e) Which low-power mode should be used, if multiple are available?

f) Are there guarantees on energy saved?

Procrastination

To assign each job the processor as soon as possible and to schedule all jobs earliest deadline first can lead to a scattered schedule: one or several jobs are executed, a pause arises and the processor is idle, a job is created and executed, the processor becomes idle again, another job is created and executed, the processor becomes idle once more ... If now the second to last mentioned job has its deadline after the last one is created, then it is possible to delay the execution of the next to last job in order to execute both jobs consecutively. This is called procrastination.

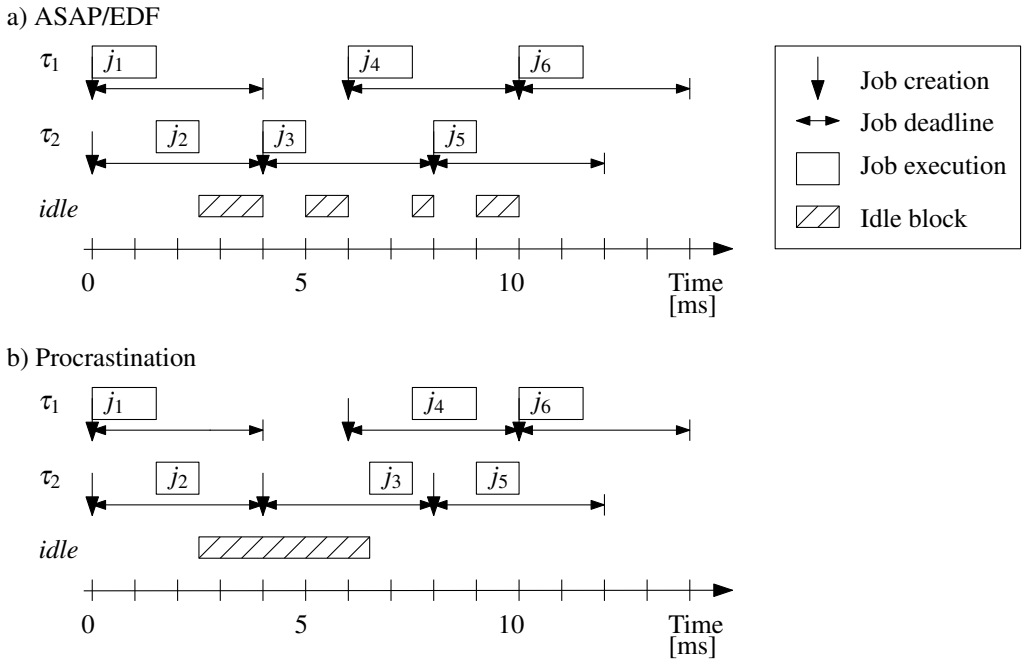


Figure 2.8.: Procrastination

An example is shown in Figure 2.8. It shows two schedules which have all in common, except the job start times. Above in the figure, the ASAP/EDF schedule is shown – execute each job as soon as possible, earliest deadline first: each job is executed as close to its creation time

as possible, and a delay is only applied when another job has an earlier deadline. Below in the figure, the procrastination schedule is shown: in order to enlarge and merge scattered idle intervals, jobs are delayed as much as their deadlines allow.

Without procrastination, we have four idle intervals. If we apply processor shutdown, we have two problems: first, the intervals are short, and we have to test if each matches the break-even time; second, we have four times to switch into and out of low-power mode.

Procrastination now delays the jobs j_3, j_4, j_5 such that they are executed consecutively and such that j_5 completes when j_6 is created. Four idle intervals have been merged into one. We need to switch into and out of low-power mode only once, and we have the advantage to use a "deeper" mode because the longer idle interval can match higher break-even times, which belong to more advanced low-power modes.

In the example, procrastination was possible because we knew when the jobs will be created. A common processor cannot assume this. As we will see in Subsection 3.2.2, some references assume a job list, that is a list of future job creation times – a look-ahead into the future. Other references compute a safe delay: each job which is created while the processor is idle can be delayed safely by this amount. Nevertheless, ideal processor shutdown with procrastination requires an optimisation over a very long life time of the embedded system – complete knowledge of the future.

2.6. Batteries as Energy Sources

The construction of an embedded system that is to operate mobile and mains independent includes the choice of an energy supply. This is a central pillar: is it unreliable, the embedded system eventually fails. Therefore, there is a need to provide guarantees on a battery's operating life that hold even for worst-cases. Naturally, these assurances need to be as least pessimistic as possible, and this motivates the optimisation of operating life.

Hereby, a battery's operating life is understood as the time one charge of the battery can be used until the device is said to be unloaded. How many times a battery can be recharged is called cycle life, and this is not directly subject here, instead it is assumed that cycle life increases when operating life is prolonged.

Electrochemical cells are mainly divided into two categories. The notion of primary cells is used for ineffective or not rechargeable cells. They have a good shelf life, that is they have notably small charge losses during storage before first use. Rechargeable cells – electrically, or mechanically by anode replacement – are called secondary cells. Electrically rechargeable cells are the main energy source for mobile embedded systems.

2.6.1. Function Principle

A battery is based on a chemical reaction, which is called reduction-oxidation reaction. The redox reaction, as it is also known, transfers electrons from one molecule to the other. It is composed of two reactions, where each has uncharged molecules on the one side and ions plus electrons on the other side of the reaction equation. Both reactions occur at different places within the battery: one at the battery's positive electrode, the other at the negative electrode. A

battery's fundamental principle is to exchange electrons outside the battery via electrodes and an electric circuit, and to exchange reactants within the battery.

Each of the two reactions can react from left to right and the other way around. The reaction direction that produces electrons is called oxidation, and the other one is called reduction. One reaction is more likely to react in reduction direction – this reaction forms the cathode – and the other is more likely to react in oxidation direction – it forms the battery's anode.

The redox reaction in secondary batteries can be efficiently reversed by applying a voltage to the battery, and the voltage must have reverse polarity. This will cause an oxidation reaction in the battery's positive pole, which is the one that forms the cathode in discharge direction, and analogously, a reduction will take place at the former anode.

Both parts of the redox reaction depend on each other, such that both reactions have to be seen as an inseparable pair. The circuit, which the battery is supposed to drive, connects the battery's electrodes, and this enables the electrons to shift from one pole to the other: now the reactions can take place; ions form at the electrodes or are consumed. The ions are dissolved into some material called electrolyte, which surrounds anode and cathode. The electrolyte itself acts only as ion conductor and solution. Direct contact of anode and cathode is avoided by a separator, which is an isolating material that allows only ions to pass through it. Figure 2.9 depicts the elementary components of a battery.

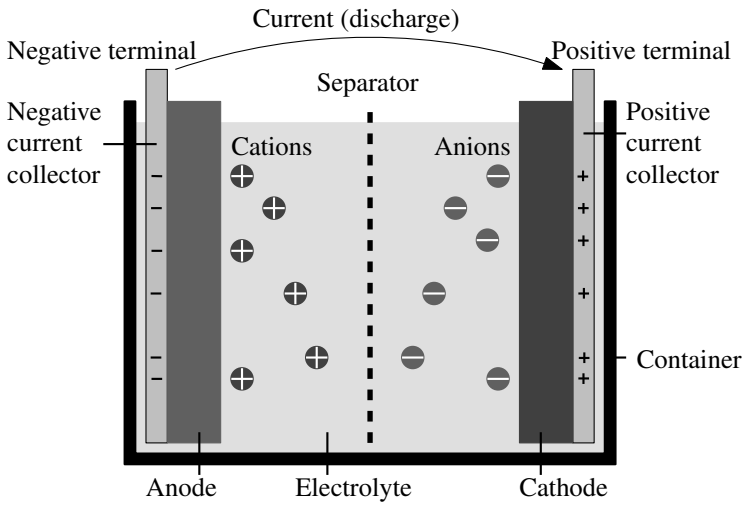


Figure 2.9.: Schematic of an Electrochemical Cell

Principle Discharge Reaction for Lead-Acid Batteries The anode of a lead-acid battery consists of lead, whereas the cathode consist of lead-dioxide. The third part of the reduction-oxidation reaction that takes place is solved in water: H_2SO_4, sulphuric acid. The water with the solved acid acts as electrolyte.

Anode reaction: $\mathrm{Pb} \rightarrow \mathrm{Pb^{2+}} + 2\,\mathrm{e^-}$
$\mathrm{Pb^{2+}} + \mathrm{SO_4^{2-}} \rightarrow \mathrm{PbSO_4}$
Cathode reaction: $\mathrm{PbO_2} + 4\,\mathrm{H_3O^+} + 2\,\mathrm{e^-} \rightarrow \mathrm{Pb^{2+}} + 6\,\mathrm{H_2O}$
$\mathrm{Pb^{2+}} + \mathrm{SO_4^{2-}} \rightarrow \mathrm{PbSO_4}$

Both, anode and cathode, consume solved SO_4 ions. Additionally, the H_3O^+-anions, which occurred by solving H_2SO_4 in water, react to water with the dioxide. The reaction stops when all SO_4 is consumed. The liquid part of the lead-acid battery becomes ideally H_2O. In this reaction, all material reacting at either electrode is either at the electrode or solved in water.

Note, although the SO_4-ions have to move through the electrolyte, this is not the case for the H_3O^+-ions: the protons build up a chain with the surrounding water molecules, where a proton moves to its next water molecule, out of which *another* proton moves to the next other water molecule. The same is true for hydroxide ions. Therefore, the H_3O^+-ions and the OH^--ions have a very good ion conductance in water.

Principle Discharge Reaction for Nickel-Cadmium Batteries The anode of nickel-cadmium batteries consists of cadmium, and the cathode is made of nickel oxy-hydroxide (NiOOH). The electrolyte is potassium hydroxide (KOH), which is solved in water. Because potassium alone is very reactive, it is important that these batteries do not run out of water.

Anode reaction: $\mathrm{Cd} + 2\,\mathrm{OH^-} \rightarrow \mathrm{Cd(OH)_2} + 2\,\mathrm{e^-}$
Cathode reaction: $2\,\mathrm{NiO(OH)} + 2\,\mathrm{H_2O} + 2\,\mathrm{e^-} \rightarrow 2\,\mathrm{Ni(OH)_2} + 2\,\mathrm{OH^-}$

The anode consumes hydroxide, which is produced at the cathode. Now, the hydroxide would have to wander through the electrolyte, and the reaction would possibly be slow unless anode and cathode are very close together. And of course the layer in between is thin for commercial or standard batteries, but because of the proton-exchange chain mentioned in the previous paragraph, this reaction can be very fast.

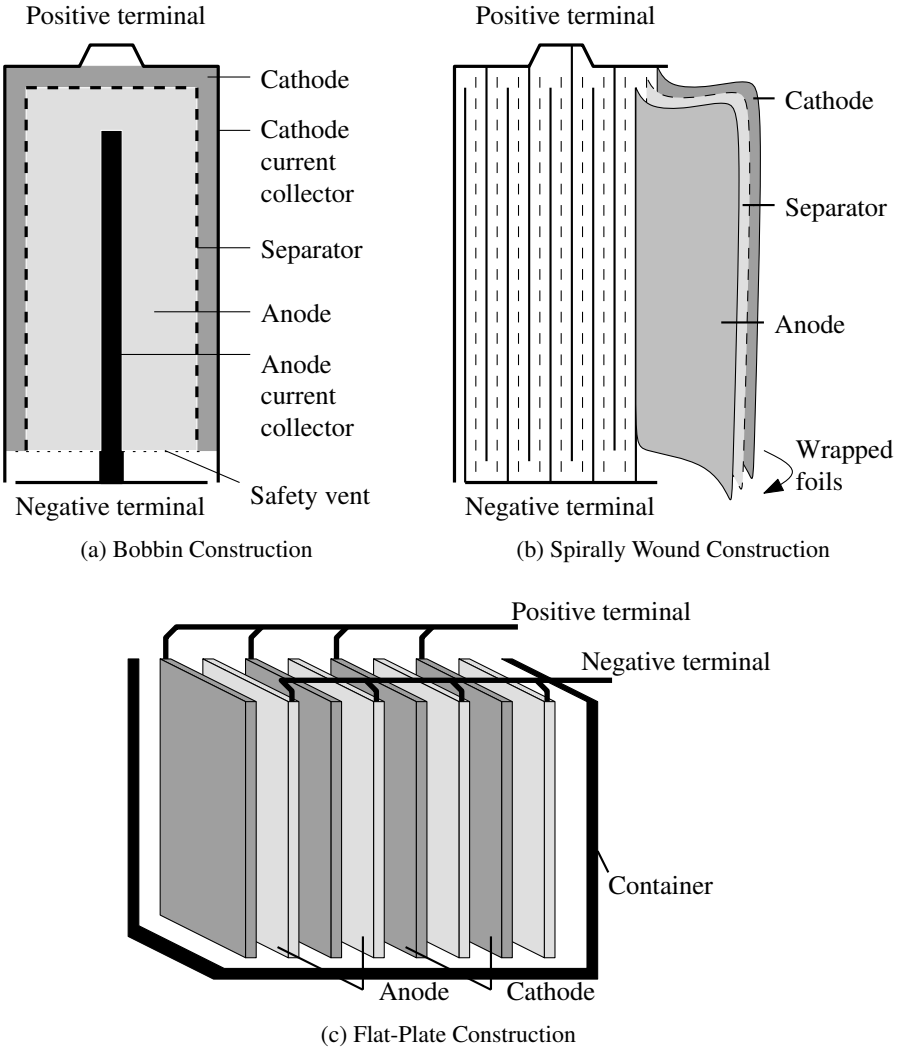
Principle Discharge Reaction for Lithium-Ion Batteries The lithium-ion battery has many variants. On the one side, there is metallic lithium as negative electrode, and on the other, there are lithium alloys as well as lithiated carbon, LiC_6. For the positive electrode, there are polymer cathodes, intercalation cathodes, soluble cathodes, and solid cathodes available. Reference [LR2002] lists an overview on page 34.3. In the following, the discharge process for a common material is shown: the LiC_6-$LiMn_2O_4$-battery with a liquid organic or solid polymer electrolyte. The structure $LiMn_2O_4$ is an intercalation cathode.

Anode reaction: $\mathrm{LiC_6} \rightarrow \mathrm{C_6} + \mathrm{Li^+} + \mathrm{e^-}$
Cathode reaction: $\mathrm{Mn_2O_4} + \mathrm{e^-} + \mathrm{Li^+} \rightarrow \mathrm{LiMn_2O_4}$

The anode material consist of a carbon grid that embed lithium in a reversible manner. On discharge, the lithium dissolves into the electrolyte leaving a 'hole' in the grid. The cathode material consists of metal layers that embed incoming lithium ions as well in a reversible manner (known as intercalation process, [LR2002], pages 35.4-35.6). That is, for this kind of battery, lithium ions have to wander from one electrode through the electrolyte to the other electrode: discharge and charge determine the direction.

This kind of battery requires very thin separator foils, which are in the range of 10 to 30 μm, [LR2002], page 35.29.

2.6.2. Construction



(a) Bobbin Construction

(b) Spirally Wound Construction

(c) Flat-Plate Construction

Figure 2.10.: Construction of Batteries

The notion of battery is generally used for the complete construction consisting out of one or more cells, which are placed in a container with electric contacts. Figure 2.10 illustrates three commonly used variants: the Bobbin Construction, depicted in Sub-Figure 2.10(a), is common for Alkali-manganese cells; the wound construction, Sub-Figure 2.10(b), is common for nickel-based cells as well as lithium-based secondary cells; and the flat-plate construction, Sub-Figure 2.10(c), is common for lead acid cells used in cars. However, wrapped foils are not only put in cylindrical cells but in prismatic (rectangular shape) cells as well.

Dimensions and used variant depend on the used battery chemistry. For example, the lithium based secondary battery from the previous paragraph is implemented with a spirally wound

construction. All components are foils of different thickness and porosity. Electrode thickness and porosity are different for each electrode because the speed of the oxidation reaction does not match the speed of the reduction reaction. This reason affords a higher porosity and thickness of one electrode to gain more surface area.

As an example for construction dimensions regarding a spirally wound cylindrical cell, reference [LR2002], page 34.34, reports for a Li/$LiNiO_2$ D-Size (a 60 mm long round cell with diameter of 32 mm) battery to have a positive electrode which is 0.38 mm thick, whereas the negative electrode is 0.10 mm thick. The cell contains 18 ml electrolyte in total – about 48 ml is the cells inner volume.

2.6.3. Battery Capacity

The amount of electric energy – electricity – that can be drawn from a battery could be understood as the battery's capacity, but this amount is different from that induced by the amount of material used, and therefore battery capacity is divided into the terms *theoretical capacity*, which denotes the amount of energy induced by the amount of material used, and *rated capacity*, which denotes the amount of energy that can be discharged under certain conditions. Unless noted otherwise, the rated capacity is generally understood as the battery's capacity.

Discharge conditions include temperature ranges specified by the battery manufacturer. Figures 2.11 and 2.12, depict the influence of temperature on available capacity. In Figure 2.11, the temperature yielding the highest capacity is 20°C; in Figure 2.12, the maximal temperature to expose the battery at – 70°C – leads to highest capacity. Obviously, the latter is beyond human internal temperature and thus rather for theoretical use than for a close-to-body application. Nevertheless, battery capacity is temperature dependent such that the temperature belongs to the rating conditions.

Further, discharge of secondary batteries is subject to an end-voltage, which is called cut-off voltage. The voltage at the battery's terminals must not fall below cut-off otherwise the battery can take damage, such that it may fail or cannot be recharged properly. Respecting the cut-off voltage will keep a rest of energy within the battery.

If we consider a battery as an element of an electric circuit, it can be divided into further sub-components. The battery's capacity divides into charge as result of the redox reaction and charge of the metal components, which also represent a capacitor. The battery contains a resistor as well: partly represented by the already mentioned metal components, partly represented by the ion conductivity of the electrolyte. Note, the conductivity depends on the reaction principle, as seen in the Subsection 2.6.1: "proton chains". For completeness, the battery as a whole may have an inductive component. All three components may be used to create an equivalence circuit representing the battery, and all these form the battery's impedance.

Two more notions are introduced for batteries, both offering better product comparability. One is the gravimetric energy density, and it denotes the capacity per unit weight; the other is called volumetric energy density, and it denotes the capacity per unit volume. All parts of a battery, like contact plates, container, electrodes, anode and cathode materials, electrolyte, separator etc. are included in both volume and weight, and therefore, metal anodes together with metal-oxide cathodes are the most preferred materials to increase both densities because each functions as electrode and active material together.

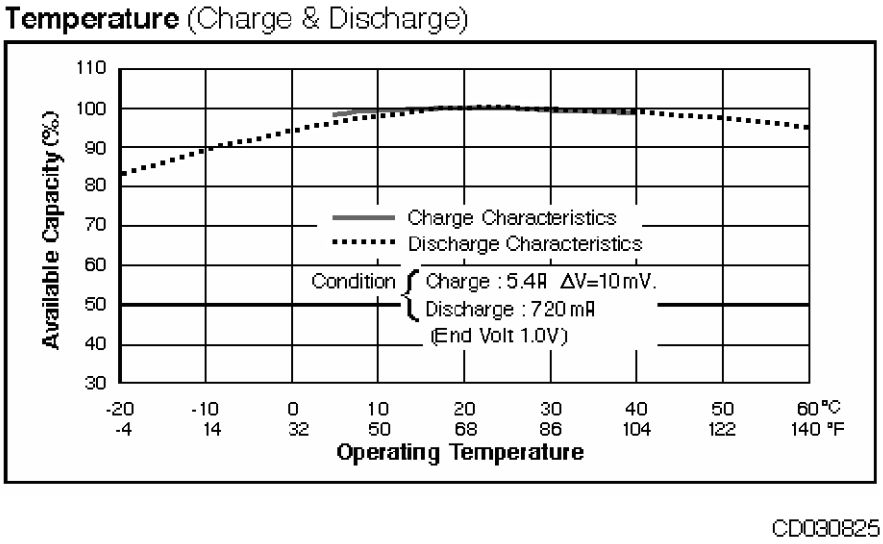


Figure 2.11.: Temperature - Capacity Curve of a Common Nickel-Cadmium Cell, SANYO CP 3600 CR [SAN2007]

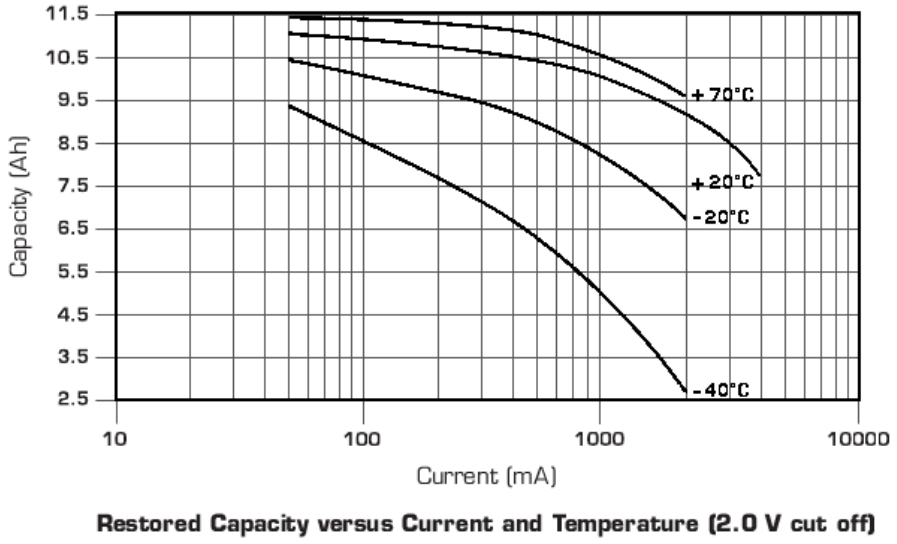


Figure 2.12.: Temperature - Capacity Curve of a Common Lithium Cell, SAFT LM 33600 [Saf2006]

2.6.4. Discharge

Cut-off voltage and impedance in combination yield the conclusion that the rated capacity is affected by the discharge current as well. This is the topic of this subsection.

Let R_{internal} be the battery's internal resistance, then, at discharge time t, the power

$$P(t) = I(t)^2 \cdot R_{\text{internal}}(t).$$

dissipates. The internal resistance changes by discharge, hence the time dependence. Any chemical energy transformed to heat cannot be transformed to electric energy, however, the battery capacity changes with the temperature: the generated heat may lead to increased capacity. Nevertheless, the argumentation with the internal resistance is the argumentation from an electrical engineer's view: the battery may increase its temperature because of reaction energy, the battery may also loose temperature if reactions are endothermic.

Without any losses, the battery's capacity will remain constant for any discharge current, with the presumption that discharge time will decrease by the factor with which the current increases. In general, the capacity does not remain constant, and time and current are not reverse proportional.

For predicting correctly how long a battery can be discharged at a certain discharge current, W. Peukert found out the following for lead-acid cells (See [Peu1897]):

$$\text{const.} = I^{\text{pc}} \cdot t, \tag{2.16}$$

where t denotes the overall discharge time, I the discharge current, and pc is called the Peukert coefficient. The latter depends on construction and chemistry of the battery: it varies between 1 and 2, and it is different for each battery. The constant will be denoted by C_{norm} and called normalised capacity, unit As (to get the units correct, first divide the current by 1 A, then compute I^{pc}, and last, factorise this by 1 A).

The formula also holds for nickel-based cells. For lithium-based cells, this is uncertain: for example, some high rate lithium-based cells have increased capacity for an increased discharge current, see [Pan2008] (2.7 Ah at 40 A current vs. 2.5 Ah at 10 A current, ambient temperature 30°C).

Because capacity is not directly proportional to current and time, and because it depends on temperature as well, batteries are given a rated capacity in terms of ambient temperature, discharge current, and time to reach cut-off voltage. Discharge current and time to reach cut-off voltage form the discharge rate. For example, data sheets for lead-acid batteries specify the 20-hour discharge rate, which is a current: a fresh battery can be discharged for 20 hours with this current until its terminal voltage reaches cut-off potential. For the discharge rate, the symbols C_n and C/n are used, where n represents the number of hours for discharge, and C stands for the rated capacity (unit: ampere-hours). For rates which define to discharge the battery within an hour or less, the symbol nC is used, n serves now as a factor of the rated capacity.

All of these symbols do not keep units correct: for a battery with 2.5 Ah rated capacity for 10 hours discharge, the discharge current at the half hour rate would be written $2C = 2 \cdot 2.5Ah$, which is not the unit for electric currents. To keep units correct, the standard IEC 61434 defines $I_t[A] = C_n[Ah]/(1[h])$, with t being the time for discharge and n the corresponding divisor of the rated capacity. With the standard, the example becomes $I_{\frac{1}{2}\,\text{h}} = 2 \cdot 2.5\text{Ah}/1\text{h} = 5\text{A}$.

2.7. Summary

This chapter started by explaining the division of an embedded system into hardware and software. It was shown how real-time feasibility of the software can be ensured with the theory of event streams, their approximation, and an approximating real-time feasibility test based on event streams.

It is the software that defines the processor power consumption, because running software causes the processor to dissipate power. The software's execution time determines the duration; the software's hardware requirements determine the parts of the processor used.

It was shown that the processor dissipates power when it switches it's internal states – dynamic power dissipation; and that it already dissipates power when it is switched on – leakage. Power dissipation depends on the processor's supply voltage, which may be reduced at the cost of lower processing speed, or instead the supply may be switched off at the cost of an unavailable processor. The unavailability is classified by the processor's different low-power modes.

Finally, mobile providers of electric energy, batteries, were presented. A battery's components of the reduction-oxidation reaction determine the cell voltage. The construction of a battery influences energy per unit weight and per size. The capacity of a battery is not a constant: it varies with the current drawn from the device, and this behaviour is modelled by the Peukert-formula for constant currents.

3. State of the Art

This chapter describes briefly previous work regarding embedded hard-real time systems. In the first section, the theory of event streams is put in context with other real-time feasibility tests. The second section covers previous work dealing with energy reductions for hard real-time systems. The last section gives an overview over existing battery models for predicting operating life, and their granularity.

However, the reader will find an overview over related work and how it compares to the work presented in this thesis not before the next chapter.

3.1. Hard Real-Time

Several models for hard real-time tasks exist. This section steps through commonly used task models for feasibility testing.

In [LL1973], the authors model their tasks with periods and deadlines equal to the periods. The authors introduce deadline driven and preemptive scheduling, which is known as earliest deadline first scheduling (EDF), and prove that if and only if there exists a schedule using EDF, then the system is hard real-time feasible. Their test is linear in the number of tasks.

The authors of [BRH1990] assume EDF scheduling and model their tasks with periods and arbitrary deadlines. They assume synchronicity and provide a test at each time point where a deadline has to be met. The test starts from time point zero, the start of all tasks, up to $\Delta t_{\max} := \frac{U}{1-U}$, where U is the long-term utilisation. The test bound $\Delta t_{\max}$ is only feasible if $U < 1$, or $\Delta t_{\max}$ is less than the hyper-period. Although a test point reduction, they prove their test to be NP-hard.

Linear complexity in the number of tasks is provided by [Dev2003]. The author models EDF scheduled tasks with periods and deadlines less than or equal to periods. The test is only sufficient as was shown in [AS2005].

Hitherto, previous work considered the real-time test to be performed on schedules, although these are specific worst-case schedules, the tests are performed on the time scale. In the Preliminaries we have seen the scale of increasing interval sizes, and to use this scale was the author's idea in [Gre1993]. He started not to diagram the processing demand for specific time points but the demand for interval sizes – we have seen the theory of event streams in the Preliminaries. The idea was picked up by few research groups, each of which is stated in the following.

The authors of [RE2002] restrict their notation of event streams to cover tasks occurring periodic, periodic with jitter, or sporadic, as well as tasks occurring in bursts. The restriction is made because of computational reasons. The transformation to this special notation, known es event model interfaces (EMIF), is affected by accuracy losses, which occur if a task's occurrence behaviour does not fall into one of the covered categories. In [RE2008], the authors widen the

expressiveness to hierarchical models. In both works EDF scheduling is assumed and yet an approximation has not been presented.

With full expressiveness, as in the definition of [Gre1993], the authors of [AS2004] introduce a notation designed to contain periodic parts of the general event stream, and they call these "event stream elements." They give an approximation that reduces the number of test points; at the same time they have a less computational error than the method of [Dev2003]: the error can approach zero by adjusting the approximation. The event stream elements are parameterised by a period and an offset. Setting the period to infinity and using a non-zero offset allows to model any non-periodic behaviour at the cost of one extra element. This extends to a drawback in cases of drifting tasks, refer to Example 2.2.3(c), page 20: the occurrence behaviour would cause 10 periodic elements of which nine would have a non-zero offset. Further on, consider a drifting task triggered by a drifting task, in this case, an over-approximation of the task behaviour would be the only chance to avoid one element for each possible interval size.

In [ABS2006], the same authors introduce a hierarchical model as compact description for tasks triggered by other tasks having their occurrence behaviour described as an event stream. If the parent is periodic, this will massively reduce the number of event stream elements as can be seen in their work.

Although on first sight designed for EDF scheduled task sets, their work has been extended in [ABS2007] to preemptive static-priority systems, again with approximation.

A very general approach for compositional analysis of tasks mapped to a processor is done in [TCN2000]. A demand curve, which represents a task's demand for given interval sizes, a service curve, which represents the available processing time for given interval sizes, and a capacity curve – the difference of both, to show remaining processing time – are motivated. A not yet explicit construction of the curves is given in [CT2005]. Named arrival function, the authors follow the same concept as Gresser's event streams. The arrival function is not explicitly given, but defined the same as event streams: for one or several tasks and for each interval size, the function returns the maximal needed processing time. Although the function can be expressed in the same manner as the notation of event streams stated in the Preliminaries, the authors restrict themselves to an approximation of their function using three lines (see [CT2005]), and causing non-recoverable, non-adjustable approximation errors. Further, this causes approximation errors in the capacity curves and other resulting functions. Since they only use demand curves without presenting how they are created, this general approach is not restricted to EDF scheduling.

Something uncovered in the work presented in this thesis is the variable execution demand, which allows tasks having multiple worst-case execution times. Variable execution demand was propagated in [MKT2004] and [BCGM1999]. For this thesis, the worst-case execution times are assumed to be constant regarding the interval size.

Summary

All related task models handle EDF scheduled preemptive tasks and assume a worst-case execution time for each task to test the real-time feasibility of the task set. On the one hand, we have periodic task models of [LL1973], [BRH1990], and [Dev2003], with deadlines equal to periods, arbitrary deadlines, and deadlines less than periods, in this order. The latter comes on the benefit of a linear complexity for the test. On the other hand, we have task models that allow

for non-periodic behaviour and arbitrary deadlines: [Gre1993] and [TCN2000], both are general and abstract; [RE2002], restricts generality to gain specialised explicit forms; and the authors of [AS2004] retain the expressiveness of [Gre1993], and present an explicit form.

The feasibility test presented in [LL1973] can verify only strict periodic tasks, with deadlines equal to periods. The test presented in [Dev2003] can verify periodic tasks with deadlines less than periods, but it is only sufficient. Both tests have a complexity that is linear in the number of tasks. The test shown in [BRH1990] can verify periodic tasks with arbitrary deadlines, and needs to evaluate every job deadline within a certain schedule; it is shown to be NP-hard.

All non-periodic task models involve a comparison of processing demand against available time on an ascending set of interval sizes, which are limited by the same test bound as before, and have the same complexity as the test in [BRH1990]. To reduce the number of test points, reference [CT2005] provides an approximation with three lines, which is inaccurate; in contrast, [AS2005] includes a test with adjustable precision and with linear complexity in number of tasks times the number of approximation steps, as we have seen in the Preliminaries (Theorem 2.4.8).

None of the authors model context switching overhead, neither do they take other forms of state transition overhead into account. Either would have been useful to determine the efficiency of a processor shutdown policy.

More general, it is to say that the energy demand of a real-time system is not taken into account by the discussed real-time tests.

3.2. Power Dissipation

Related in this field is previous work that can handle hard real-time constraints. Optimisations of energy consumption at the price of deadline violation are disregarded. Previous work that provides offline analysis of energy consumption is related as it can show a designer advantages and disadvantages before he or she puts the designed system into real-world. Although this thesis covers tasks as independent units, which are especially not refined into parts, intra-task methodology is also related if it applies to basic blocks with dependency constraints, which could then be seen as tasks with dependencies.

This section first deals with related work that explores modification of voltages while the processor is running, second it covers related work that explores low-power modes and regards mode transitions.

3.2.1. Voltage Scaling

We step now through the achievements on slowing down processing resources at the gain of reduced power dissipation. Intra-task optimisations are considered as well, as they can be understood as analysis on dependent tasks, with all tasks having the same occurrence behaviour.

In [IY1998], the authors use a task model where all tasks have the same period, which is also seen as common deadline. The tasks themselves are without dependencies. Their method to reduce power dissipation uses DVS without considering the cost for transition.

The authors of [ASE+2004] (also available as [AEP+2007]) model their task system as a graph: nodes represent tasks; edges represent precedence constraints; each branch is taken. The

tasks are allowed to have individual deadlines and are not preemptive. There is one period for the whole graph: at the period, the root node is triggered, and the root node triggers all dependent tasks, such that jobs are executed without pause in between. Slack will only appear between the last job of a graph and the job representing the next graph's root node. The used energy model includes DVS and ABB, modelled with and without transition cost, and voltages are scaled continuous as well as discrete. The discrete voltage selection problem considering transition overheads is reported to be NP-hard.

A similar approach as before follow the authors of [RP2006]. They also use a task model with a common period for all tasks, and the period is taken as common deadline as well. Tasks are allowed to have dependencies and are considered non-preemptive. The tasks' resource usage is included in the optimisation problem. For energy optimisation, DVS without transition cost and dynamic power management with transition overhead for the resources is used. The authors report their problem to be NP-hard.

The authors of [JG2004] model periodic tasks with individual deadlines scheduled earliest deadline first. For energy optimisation, they use an own ellipsoid algorithm together with Baruah's feasibility test: a frequency setup is determined and then tested with the feasibility test; by interval bisection, this is repeated until a solution is found and a given precision is reached. They model DVS without transition overhead.

The authors of [RHE+2006] use the event model interface to model EDF scheduled tasks. Their previous work, [RJE2005], is used to compute a minimal yet feasible processor frequency to run all jobs with. Similar to the aforementioned, by interval bisection, they repeatedly invoke their real-time feasibility test to test their frequency configuration. They model DVS without transition overhead.

Instead of an optimisation, an offline heuristic is presented in [ISG2003]. The heuristic is reported to be within factor 2 of the optimum. The offline algorithm optimises for an arbitrary, but finite, set of EDF scheduled jobs – not tasks. Jobs may have an individual creation time ("release time"), worst-case execution time ("units of work", see the reference) and deadline, and the job set is considered preemptive. If the job set was real-time feasible, then after the energy consumption has been improved, the job set will still be feasible. For the improvement of the energy consumption, a processor with one sleep state is modelled with transition cost included, and DVS is modelled without transition cost. Because it only works on a non-empty set of jobs, the offline heuristic is for comparison only, and an online heuristic is presented that is to be used in a real system. The same assumptions for jobs are made, as the offline version. The online algorithm sets the processor to two different speeds: maximal and least possible speed. Maximal speed is set, if not all of the queued jobs can meet their deadline, otherwise the least possible speed (determined before system deployment) is set, and if there are no jobs in queue the processor is send to sleep.

Two aspects follow: probability based usage profile, and probability based execution time – each of which is not considered in this thesis.

In [SKL2006], tasks are arranged in a graph: nodes represent basic blocks; edges mark task precedences; branches represent alternatives, that is, each node is followed by exactly one branch. Basic blocks are considered non-preemptive, and one deadline is assumed for the whole graph. An occurrence model is not considered for the graph, and the graph is assigned exclusively to the processor. For the branches, a usage profile is assumed, which is supposed to

contain for each branch the probability that it is taken. Based on this, the average power dissipation is reduced by applying dynamic voltage scaling with transition overheads and a set of discrete voltages.

Similar assumptions are made in [SK2005]: tasks are arranged in a graph, with basic blocks as nodes, edges as precedences, branches as alternatives. The graph has one deadline, is assigned to the processor exclusively, and has no occurrence model. Again, the basic blocks are non-preemptive. Now, voltage adjustments are calculated for each edge: first, the voltage is assigned as if being on the longest execution path; second, the probability is determined to be on a shorter path, and then the voltage is adjusted. The latter is a prediction, and if the prediction shows to be false, the voltage is increased again.

Summary

Related work regarding processor voltage scaling splits into two parts. First, in [ASE+2004], [RP2006], [SKL2006], and [SK2005]: tasks are modelled non-preemptive, with precedence constraints and common period, and with common deadline, except for [ASE+2004]; systems are modelled with transition overhead for voltage scaling, except for [RP2006]. And second, in [IY1998], [JG2004], [RHE+2006], and [ISG2003]: tasks are modelled preemptive and with deadlines; except for [ISG2003], which presents a job set based online heuristic, all of these related works present an optimisation.

Related work that considers tasks to be non-preemptive allows for computing processor speeds individually for each task, or for basic elements a task consists of. All this is achieved for periodic only tasks with dependencies and a common period, and because problems are shown to be NP-hard, heuristics are used – also true for [ISG2003].

Related work that considers tasks to be preemptive allows for computing processor speeds individually for each task ([IY1998] and [JG2004]), or a common speed for the whole task set ([RHE+2006]). Tasks are modelled with a common period which also is the deadline ([IY1998]), with individual periods and deadlines less than periods ([JG2004]), or event model interfaces are used ([RHE+2006]). The complexity of the solutions is that of an integer linear program in the case of [IY1998], and the logarithm of the precision (interval bisection) times the complexity of the underlying real-time feasibility test for [JG2004] and [RHE+2006].

It remains open to provide an energy optimisation with voltage scaling for preemptive task sets with arbitrary occurrence behaviour and arbitrary (but fixed) deadlines. Further, the optimisation should be able to compute a common frequency for all tasks as well as task individual frequencies. And to reduce the computational complexity, instead of invoking a real-time feasibility test, the optimisation is to be integrated into the test.

3.2.2. Power Management

As was seen in the Preliminaries, Subsection 2.5.3, energy consumption can be reduced by shutting down parts or the whole system. The main difference to the previous section is now that the processor or parts of it become unavailable during low-power mode. Additionally, it affords non-negligible transition cost – time and energy – to wake up the circuits. From the previous subsection, we already know some related work which is able to deal with these aspects:

[ASE+2004], [RP2006], and [ISG2003]. In the following, related work is discussed which primarily focusses on processor shutdown.

Non-Hard Real-Time Capable Power Management

A performance-power trade-off is made by dynamic power management (DPM): "suitable" idle periods are used for processor shutdown, and if the prediction fails, jobs are discarded. The decision when a period is suitable is made by the DPM algorithm: it determines duration and the low-power mode to use. Though mostly computed online, approaches exist which perform an offline analysis to determine parameters for an online algorithm. Surveys on dynamic power management can be found in [BdM2000], [LM2001], or [GIS2003]. They categorise works on dynamic power management into time-out policies, predictive policies, and stochastic policies.

A time-out policy decides when to shutdown. The transition is taken after the monitored resource has spent a given time in idle-mode. This was first presented in [SCB1996]. The authors' experiments showed a high probability that an idle period is either short or long. Their approach computes the length of the time-out before the system runs, and the time-out is unchanged during run-time.

The authors of [HW1997] introduced an adaptive scheme for calculating the time-out. The idea is to observe the duration of idle periods over a time frame of specific length, and to use these observations to predict the duration of the next idle period. If the predicted value exceeds a certain limit, then the processor is sent to low-power mode, and a timer is programmed to wake up the processor again. If the prediction was correct, the programmed time is right before the next job needs to be processed.

Stochastic policies, a long list can be obtained from the surveying papers cited above, modify the approach mentioned before to adapt online knowledge. Policies include parameterised probability distributions, Markov-chains, refinement of those chains, and the observation of a certain amount of past idle-periods and past processing periods. In short, it is the aim to correctly predict the probability that a next idle-period will last longer than the break-even time.

Every work mentioned in this sub-subsection is either unaware, ignores, or tries to lower the latency which is the result of its application. None explicitly tries to retain any deadline or even provide a latency of 0.

Power Management Maintaining Deadlines

The following related work deals with shutdown of devices or the processor in a hard real-time environment.

Power management for devices is used in [SCI2001]. The authors make use of a resource request list obtained by jobs in the schedulers run-queue. With this they "know" about future device usage and can make proper use of the devices' low-power modes, without violating a job's deadline or the break-even time. The authors deny that their approach can be applied to hard real-time systems: an incoming – new – job will be affected by a latency caused by devices which have been shut down. The authors' algorithm needs to look ahead a certain amount of jobs to ensure devices are up in time and deadlines are met.

The authors of [CG2005] use power management for devices and voltage scaling for the processor. With respect to job deadlines, slowing down and speeding up the processor is used to procrastinate device usage in order to optimise device idle-periods for power saving. Power management is done with similar knowledge and the same drawback as mentioned a paragraph before. If no job requests a device, then it is shut down; if there are no jobs in queue, then all devices are shut down: an incoming job will block and will have to wait for its requested devices to get ready. The authors solve this by requiring an extra maximal "blocking time" in their task model for testing real-time feasibility. The used test cannot handle more complex behaviour than periodic tasks.

In [LJ2000], the authors present an algorithm which computes a static schedule for periodic and sporadic tasks. With the schedule being static, it is easy to estimate transition cost. The schedule's end time is equal to the hyper-period of the periodic tasks, then the schedule repeats. For sporadic tasks, time slots are inserted, and these slots may be moved when the system is online to further improve the schedule. Sporadic tasks may have hard or soft deadlines, and all tasks may have precedence constraints based on data dependencies, then all tasks in a dependency graph share the same period. Tasks with jitter are not taken into account: these must be modelled as sporadic tasks with two times the jitter minus the period as minimal distance, which is pessimistic and reduces processing time for allocating more tasks. Energy consumption is reduced by applying dynamic power management and voltage scaling. The latter is preferred because of on an early argument that neglects leakage ([HKQ+1998]).

The authors of [LRK2003] present a modified EDF scheduler. It procrastinates jobs to extend idle periods. Only periodic tasks with deadlines equal to their periods are considered. The procrastination time δ_ω for a job of task ω is calculated via

$$\frac{c_\omega + \delta_\omega}{T_\omega} + \sum_{\tau \in \Gamma \setminus \{\omega\}} \frac{c_\tau}{T_\tau} = 1, \tag{3.1}$$

with c, T being worst-case execution time and period. The aim of this equation is to keep processor utilisation below 100%. On the one hand, if the tasks' periods in the fraction are replaced by the tasks' deadlines, the formula can handle tasks with deadlines shorter than periods, but on the other hand, the result will be very pessimistic: in cases with small deadlines compared to periods, no procrastination will take place as the sum then is greater than 1. Further, the authors assume external circuitry, e.g. a special purpose device, to keep the job queue, to compute procrastination time, and to shut down and switch on the processor (ibid, page 3).

The work cited in the foregoing paragraph is extended by global slowdown in [JPG2004]. The authors use exactly the same approach for procrastination, and also consider only periodic tasks with deadline equal to period. And again, they assume external circuitry to perform procrastination and to control the processor's power states.

The work presented in [NQ2004] provides lower energy consumption than the two previously mentioned works. The authors include periodic tasks with deadline lower than period, but not higher. Their algorithm works on a job set, which is said to contain a number of jobs with their creation times, deadlines (from the corresponding tasks), and execution times. To save energy, job slow down, that is adjusting the voltage for each job, and processor shutdown are applied, and to merge scattered idle periods, jobs are procrastinated. The computation is supposed to be

done offline and online: because the computed times for the procrastination of jobs are fixed for the job set, they have to be recomputed when the job set changes, that is whenever jitter or other non-periodic behaviour takes place. An external circuit is required to monitor the job set, to perform recomputation, and to control the processor's power states.

Summary

In contrast to voltage scaling, energy saving solutions involving power management account for non-negligible transition overhead. We have seen approaches which are not feasible for hard real-time systems; and those which are, consider tasks to be periodic – or sporadic in the case of [LJ2000], which is the only one to compute a static schedule. In [SCI2001] and [CG2005], device power management for preemptive jobs with deadlines was presented. Both solutions imply "blocking times" for incoming jobs. In [LRK2003] and [NQ2004], processor power management for preemptive EDF scheduled tasks was presented.

All real-time feasible solutions require an extra circuit for job queue monitoring, computation of procrastination, and control of mode transitions for the processor. All optimisations are performed online, at least with recomputation. The task model is always periodic, and with deadlines equal to period in the case of [LRK2003], or less than the period in the case of [NQ2004].

It remains open to provide an energy optimisation using low-power modes that is capable of arbitrary task occurrences, and arbitrary deadlines. It remains open to reduce the hardware demand to a minimum: a simple timer for wake up. Last, to allow for a prediction of operating life, it remains open to compute the energy demand of the future system offline.

3.3. Battery Modelling

If the lowest energy consumption yields the longest battery operating life, then an optimisation to a smooth and low power profile would be the best result, and no battery model would be necessary. But this is an assumption, thus we have to take a look into battery modelling.

We have to determine the achieved operating life for each power profile. Using Peukert's formula this can be done in first approximation. Better values are supposed to be obtained by more detailed battery models. While creating new models, researchers found out other important factors that determine battery operating life and incorporated those into new models. These models are discussed during the following of this section.

The authors of [RVR2003] give a survey from a computer scientist's view on models for lithium-based secondary cells. The overview structures battery models into physical, empirical, abstract, and mixed models: physical models are made by electrochemists and give a deep insight into cell reactions and processes; empirical models try to match measured data via parameter fit; so called abstract models are circuit equivalents for batteries (e.g. SPICE implementation), equivalent hardware entities (e.g. VHDL implementation), and Markov chain approaches; and so called mixed models have experimental roots and parameters representing physical properties. The following categorisation augments this view, but it omits a distinction between empirical and mixed models, because model parameters are always fit against experimental data from simulations or measurements. Here, analytical models and coulomb counters

form the last subsections, both deal with battery models that are not a simulation, but aimed at providing a closed formula for the prediction of a battery's operating life.

3.3.1. Physical Models

Physical models include the most detail. All models are based on differential equations. The electrochemical process – the redox reaction and its participants – influences the physical structure of the resulting cell, and the structure has an influence on the process. The modelling of interaction and the process involves several chemical and physical properties:

a) A detailed model of the electrode surface or at least its area, where surface or area are combined with the current density to determine the electron flow through the electrode.

b) For anode and cathode a separate reaction rate depending on surface area, current density, and electrolyte concentration.

c) Material kinetics representing the distribution development in the solvent

d) Temperature, which itself may be modelled non-isothermal, that is in dependence of the location within the cell.

This amount of detail affords deep knowledge of electrochemistry and a large set of parameters (see below). The latter needs to be given from a battery manufacturer as he has built the product. Thus, the intended use of detailed physical battery models is to design and build batteries, see for example [DFN1993].

Reference [DFN1993] describes a physical model of the lithium/polymer/insertion cell. The cell's electrodes are made of a lithium compound, and its electrolyte is a polymer. Lithium ions move back and forth in the electrolyte, insert into, and move out of the lattice structure of the electrode material (lithium insertion, hence the name). This model has been generalised in [FDN1994b] and resulted later in the Dualfoil simulator, which since version 5 (2005) includes simulation of nickel-metal-hydride cells, refer to [New2009]. The simulator's lithium version has been used to develop and adjust some of the analytical models cited below. Note, concerning electrode dimensions and battery chemistry, the simulator uses about 50 parameters.

The authors of [GMS$^+$2007] developed a set of differential equations to model a lithium-anode/hybrid-cathode (carbon-fluoride, silver vanadium oxide) primary cell, which is intended to power supply implantable medical devices. In order to simplify the model, the authors neglect almost all effects at the anode, and thus concentrate on cathode reactions, but still involve more than 20 parameters.

In [LDWG2005], a simplified model for the nickel-hydrogen cell is presented. Assembled as a battery, it is to be used in spacecraft applications and meets currents of about 0.5C (the two hour discharge rate). The model includes hydrogen pressure within the cell, the thermal process, and the electrochemical process. The latter itself is parameterised using 11 parameters, and four of those "may vary from one battery to another due to different contents and pore structures". That is, parameters vary with the battery, and an in-depth knowledge of the battery is necessary to determine these variations.

An overview on nickel-metal-hydride cells is given in [PSW2002]. The authors describe the chemistry to be a replacement for nickel-cadmium cells regarding environmental issues, and report a three times higher energy density compared to nickel-cadmium. To model cell geometry, such as electrode thickness and area, the authors require 13 parameters. Those are obtained by either asking the manufacturer or disassembling the battery. Also included in the model are parameters like the area of the electrodes' surfaces per unit weight and parameters regarding side reactions to improve model accuracy.

A spirally wound lead-acid battery is modelled in [HL1999]. In contrast to their related work, the authors stress the necessity of a three-dimensional model regarding cell geometry: because of the structure of the spirally wound construction and the porosity of the electrodes' surfaces, a two-dimensional model will not fit. Therefore the authors' battery model includes at least 27 parameters.

In summary, all physical models bring the most accuracy. But the accuracy comes at the cost of many parameters to model battery construction, chemical solutions and gradients, or temperature gradients, and almost all of the list are hardly to determine. The authors create physical models for a better understanding of cell processes, and to replace experiments by simulations in order to reduce the number of experiments.

3.3.2. SPICE-Models/Electric Circuit

Cell behaviour which is modelled with partial differential equations still needs an implementation as a program for computer simulation. This can be done via several programming languages or environments, for example the Dualfoil simulation program is written in the FORTRAN language, but approaches like that require preparation of the input data. Circuit models are designed to describe and verify the behaviour of electric consumers. If now cell behaviour is also described as a circuit model, then circuit designers can directly access the cell model. The following references provide circuit models for electrochemical cells.

Reference [BKN1999] presents a detailed one-to-one mapping of partial differential equations and their solutions onto resistors, capacitors, and transformers as circuit equivalents, which are backed by transfer functions and data tables. Their model implements a nickel-cadmium cell and uses 31 "most important" (ibid, page 149, Table 2) parameters.

A similar idea is followed in [ETM2008]. The authors implement a lead-acid cell on the basis of a resistance network and current sources. Here, in contrast to the work cited before, the cell is partitioned into a matrix of electric nodes, each representing a cell volume. A cell volume itself is implemented as a current source, and the characteristic is described separately for each cell. Each characteristic is a local integral solution of the differential equation that describes the electron flow within the original cell model (a physical model). Although the use of solved integrals speeds up simulation, the parameters to solve them still need to be known, and the model needs to be evaluated for other cell chemistries and battery designs.

The author of [Gla1996] presents a battery model for a nickel-hydrogen cell. The model allows to combine one or more cells to one battery. The model splits up into two parts: one part implements the cell's chemical-energy to electric-current conversion, and vice versa respectively – both conversions are derived from reaction equations; and the other part describes dynamic cell behaviour which is supposed to match experimentally determined battery responses to AC loads.

An early work which models cells with a supposed-to-be equivalent circuit is [Hag1993]. The implementation is restricted to major effects, and all adjustments are derived from experiments. The circuit includes cell resistance and rated capacity, and capacity decrease is modelled discharge dependent. Necessary input data is a list of pairs that represent a relation of state-of-charge versus open-circuit voltage. In [Hag1995], the author extends his model to nickel-metal-hydride cells.

Following the same principles as before, the author of [Gol1997] further extends the model to be able to simulate a lithium-ion cell. The augmented model includes heat generation within the cell and heat transportation out of the cell: ambient temperature, heat capacity, and heat conductance of the cell become necessary parameters. Further on, the model includes capacity losses caused by cycling. As the one cited in the paragraph before, this is still a simple model, and the author stresses that "simulation is a tool, it is not a substitute for building and exercising real circuits.": this forms the expectation that such simplified models will not provide more insight in battery technology as was known before using them.

Circuit equivalents of batteries are available in various degrees of detail. All models use an interface, which is directly usable with electric circuits that describe the load. The design of an electric circuit is an essential step when an embedded system is designed. Thus, circuit equivalents help the design of the whole system because consumer and provider of electric energy can be simulated in the same environment.

3.3.3. Abstract Models

Because circuit models can grow very large for modern embedded systems, hardware description languages (VERILOG, VHDL) are used. These implementations describe registers, busses, logical units, and other components which would be too computational intensive to produce them as circuit equivalents. These languages are provided with an own simulation environment, which is less computational intensive than a circuit simulator simulating the same component.

With a circuit model as basis, in [BCM$^+$2000], the authors present a discrete-time battery model and implement it in VHDL. This model is convenient for hardware designers as long as they use hardware description languages to design their systems. As hardware designs grow, this approach eventually suffers from the same drawback as the circuit equivalents of batteries: increased design complexity changes the development environment of the embedded system. Additionally, because the model is based on a circuit equivalent, it inherits the errors: the authors speak of an error about 15% in their circuit model. On the one hand, a change of the circuit basis and a redesign of the VHDL model can improve accuracy, but on the other hand, this will result in increased complexity.

Another way of abstraction for the reduction of computational complexity and the improvement of accuracy is presented in [CR1999]. Instead of a hardware description language or a circuit equivalent, the authors use a Markov-chain to model battery behaviour. Each state of the chain represents a state-of-charge of the battery. Forward transitions, which direct from states of high charge to states of low charge, represent the discharge. The probability to take a transition increases with the current drawn. Back-transitions model a so-called Relaxation-Phenomenon, that is, if a battery is allowed rest periods – no or very little discharge – then its conversion efficiency is supposed to increase: a back-transition is taken; a state of higher charge than before is

entered. As back-transitions are intended, the battery can recharge while being not discharged. Although the probability to shift back to the full state tends to zero, it is still possible – if one only waits long enough. This behaviour contradicts at least battery self-discharge.

3.3.4. Analytical Models

Research has also developed other means of modelling battery behaviour than to map electrochemical effects onto simulator components: research has developed closed formulas to predict battery operating life.

The Peukert-formula discussed in the Preliminaries, Subsection 2.6.4, is one example. It takes the average load current as input and produces a prediction of the operating life as output. Its single parameter, the Peukert coefficient, has to be determined by measurement for each battery type. With this single parameter, the model provides the fastest and easiest way to predict operating life. It has a drawback: it underestimates available capacity, as experiments in [DS2006] show. Especially, if two different loads have the same average, then the Peukert formula handles them the same. This is not true for a real battery.

A similar approach as above follow the authors of [PW1999]. They consider the load current non-constant, and model it by a distribution function. They present two models for the battery's efficiency to convert energy: one depends linear on the drawn current, and the other depends quadratic on the drawn current; the authors do not combine both. This is inappropriate because Peukert coefficients are between 1 and 2: the conversion efficiency corresponds at the least to a polynomial with a linear and a quadratic term.

A stochastic battery model is described in [SC1997]. The authors use multiple discharge curves to fit the parameters of a Weibull probability function. The approach is shown for a lithium primary battery. According to the authors, six measurements on a real battery are needed to build the model, then, an operating life can be estimated for a constant load and temperature. It is left out, how this applies to non-constant temperatures and loads.

The Dualfoil simulator is used in [RV2001] to develop a formula to predict the operating life of lithium cells. Before using the formula, two parameters need to be estimated by either measurements or simulations. A staircase function that denotes the ranges of the load over the whole discharge time serves as a profile for the load. An algorithm decides with a profile as input which part of the profile cannot be processed: the algorithm prints the operating life and stops when the battery is said to be empty. The algorithm is needed because a closed formula is not developed: the authors do not make the variable for operating life explicit "since equation (15) is hard to solve for L" (ibid, page 490) – L is the variable which represents battery operating life.

The Dualfoil simulator is also used in [RP2003] to develop a formula to predict the operating life of lithium cells. The formula computes the time that remains from a given state-of-charge on. The state-of-charge is to be described by the previous discharge current in the same cycle, information about previous cycles, temperature, and the future discharge current. Here, the term 'current' always refers to average current, and it remains unclear how this model fits for time varying currents, e.g. stepwise, the profile of a dynamic voltage scaled processor.

In summary, the Peukert-formula states a non-linear dependency of discharge current to the electric energy – electricity – that can be discharged. Constant loads are assumed in [PW1999],

[RP2003], and [SC1997], where the first of the list assumes either a linear or a quadratic relation of current to dischargeable electricity – contrary to Peukert. The only model that accounts for non-constant loads is described in [RV2001], but does not provide a closed form: all discharges have to be computed for the whole lifetime of the system, and then inserted into some test algorithm, which outputs the point in time the battery is discharged.

3.3.5. Coulomb counters

The stated problem in this thesis is to optimise operating life. In contrast to model a battery's run-time behaviour, an estimation of operating life based on a few characteristics of the current profile drawn from the battery might provide accurate results.

The physical law for electricity that is conveyed by a time dependent current $I(s)$ is

$$\int_t I(s)ds = C,$$

with duration of the current t in seconds, and conveyed electricity C in As or C (coulomb). This law is used in battery monitoring devices to count the electricity that is discharged from the battery, such devices are [Her2004], [ST 2009], [Max2003]. This method assumes that the battery converts chemical energy into electric energy with the same efficiency for all currents: it is assumed that operating life depends linear on the discharge current.

The Peukert-formula states a non-linear dependency of current to obtained capacity, and already Rossander and Forsberg, see [RF1900], proposed to account currents in a non-linear fashion for estimating the operating life. The idea is to break a non-constant current into constant components and to account for each component the corresponding electricity that is discharged. Their law is feasible but a very pessimistic underestimation, see [DS2006][1].

3.3.6. Summary

Battery models divide into a class of hardware (battery hardware) simulators, and into closed formulas with the single objective to compute operating life. Available simulation granularities are physical models, circuit equivalents, hardware equivalents (implementations in hardware description languages), in descending order of insight and precision. Analytical models and electricity counters provide closed forms for estimating a battery's operating life.

For the simulators, physical models include the most detail regarding chemical processes and including mechanical properties; these simulators are suitable for battery design. Circuit equivalents or hardware equivalents restrict to certain battery properties which causes computational errors. For the closed forms, constant loads are assumed for [PW1999], [RP2003], and [SC1997]. The model of [RV2001] can handle non-constant loads, but the authors do not solve their equations to output operating life directly. Models for simulation and equations for operating life are parameterised: the parameters must be adjusted by experiments. Electricity counters are the simplest form for estimating the operating life; they account the current linear in relation to conveyed electricity.

[1] The authors do not cite Rossander and Forsberg, but study a similar idea.

It remains open to provide an explicit solution for non-linear currents: measurements are necessary to quantify the effect of non-constant discharge, at least for short pulses caused by an embedded radio application.

3.4. Summary

This chapter describes work related to the work presented in this thesis. Works regarding real-time feasibility, energy saving by voltage scaling, energy saving by processor shutdown, and battery models were discussed.

Existing real-time feasibility tests provide fast computation for EDF scheduled periodic task sets, but concerning jitter and arbitrary but fixed deadlines, only few tests provide an analysis, and one is the approximated test presented in the Preliminaries. None of the tests includes any awareness of energy issues.

Related work regarding the adjustment of a processor's supply voltage includes non-preemptive periodic tasks with precedence constraints, and EDF scheduled preemptive tasks. Algorithms for non-preemptive tasks allow for computing task individual frequencies, as do algorithms for preemptive tasks, but these require tasks to be periodic and with deadlines less than periods. One algorithm exists that computes a frequency common for a set of preemptive non-periodic tasks having the event model interface. All optimisations for preemptive tasks invoke a real-time test several times to repeatedly check the computed configuration. It remains open to compute task individual frequencies for tasks with arbitrary occurrence behaviour and arbitrary deadlines, and to integrate this computation into a real-time test in order to reduce complexity.

All real-time feasible solutions that exploit low-power modes restrict to task sets that are periodic with deadlines equal to periods, except for one solution that can handle deadlines less than periods and jitter, but with online recomputation whenever a jitter occurs. All optimisations are performed online and require an extra circuit for monitoring the job queue, computing procrastination, and controlling mode transitions. It remains open to provide a solution that can handle arbitrary occurrence behaviour along with arbitrary deadlines. Such a solution should require only minimal extra hardware, and be offline to allow for an estimation of the system's energy demand.

Battery models with high detail are available, but unfortunately these models require deep insight in order to use them correctly, and additionally, they require the load to be described for a whole discharge cycle. This is also true for the only analytical battery model that can handle non-constant loads. The other models that provide a closed form to compute a battery's operating life can handle only constant loads. It remains open to provide an explicit solution for non-linear currents: at least to quantify the effect of non-constant discharge that an embedded system such as a mobile phone causes (short pulses).

The next chapter shows the underlying concept of this thesis, the chapter will provide a tabular comparison of voltage scaling and processor shutdown by juxtaposing the contribution of the work presented in this thesis.

4. Concept Presentation and Classification

This chapter presents the concept behind this work. The first section introduces task graphs, and with the help of this concept, the levels of abstraction used in this thesis are described. The second section states necessary knowledge and assumptions made. The work is classified by the last three sections, of which the first shows the objective which the work in this thesis is going to optimise, the second describes the conceptual work flow for the optimisation of battery lifetime, and the last closes the chapter with a comparison of the work presented in this thesis with the work of other researchers.

4.1. Task Graphs, and Levels of Abstraction

For the moment, let us understand a task as a concrete action, not to be refined. Suppose the task interacts with or depends on other tasks and call this relation a process or task graph. Now, we label the edges with markings – these define the context of the relation. With this notion in mind, let us step through the following paragraphs, which each illuminate a certain aspect of the relations between battery, processor, and software.

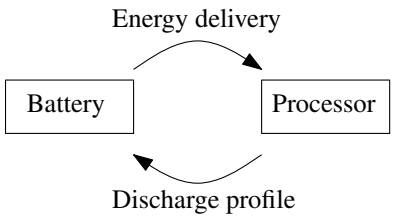


Figure 4.1.: Battery and Processor, Discharge Relation

Discharge Process Figure 4.1 shows the two participating members of the discharge process considered in this thesis. The battery acts as provider of electricity which eventually stops delivering after its operating life has been exceeded. The processor acts as a consumer of electricity and provides a discharge profile out of which the battery is supposed to determine its operating life.

Although not considered in this thesis, two or more processors may be connected to one battery. Processors may influence each other: indirectly, because each draws a certain amount of energy that lessens the available energy for others; or directly, because their discharge behaviour causes fluctuations in the voltage or current, e.g. noise. Their discharge profiles are to be arranged then: this is to be done by superposition, that is, it is assumed that both processors act independent; or a synchronisation is assumed.

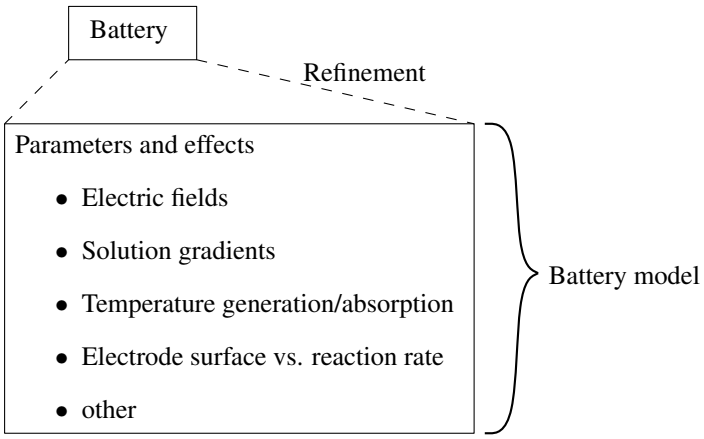


Figure 4.2.: Battery Process

Battery Process Figure 4.2 depicts a refinement of the discharge process taking place within the battery. But although the discharge process can be refined, the effects and parameters are supposed to be hidden, or incorporated into a battery model. The process gets the discharge profile as input and calculates operating life as output.

However, the battery model is supposed to also be accurate in terms of operating life regarding further conditions like ambient temperature.

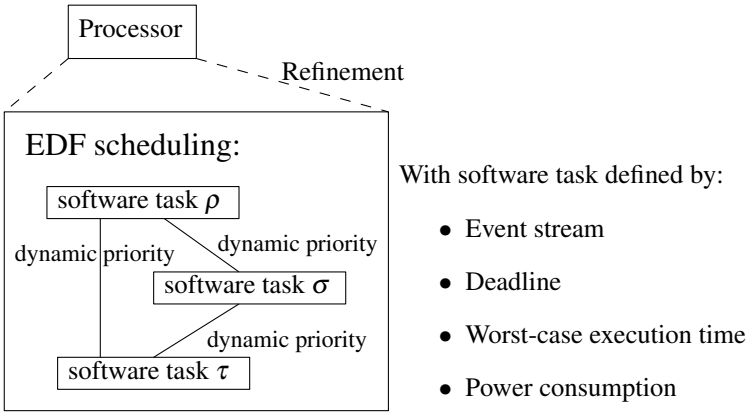


Figure 4.3.: Scheduling Process

Processor, Scheduling Process The refinement of the process defining the discharge profile is shown in Figure 4.3. Several software tasks share the same processor; dynamic priorities determine the running task which determines the processor's power consumption. Task switches, transitions into low-power modes, and frequency changes are scheduled and define the processor's discharge profile.

A software task is not refined into, but a task is determined by event stream, deadline, worst-

case execution time, and power consumption, whereof the first two parameters yield a task's dynamic priority.

4.2. Premisses

The work in this thesis makes several assumptions. These are used to mark a starting point for the described research, and to define knowledge to be known before the achieved energy optimisations and the estimation of operating life can take place.

Presented work in this thesis assumes a **single processor connected to one battery and with the software partitioned into tasks.**

Processor

The processor is assumed to be **voltage scaling enabled**. The processor is assumed to continue processing while the frequency changes: transition may be configured at run-time, and the cost for frequency change is considered to be negligible. Further, frequency-voltage pairs are supposed to be given for determining a discharge profile.

The processor is assumed to have one or more **low-power modes**. A mode transition may afford significant time and energy, and both are assumed to be known, as well as the processor's power consumption during the stay in low-power mode. When the processor is idle, this may result in less power dissipation: the power consumption of the idle mode is assumed to be known, but the cost for transition are assumed to be negligible.

A minimal extra circuit is required to wake up the processor out of any low-power mode. This is supposed to be a **programmable timer or alarm**.

Battery

Ambient **temperature** influences battery capacity to a high degree, as we have already seen in the Preliminaries. Temperature influences operating life, and life prediction will only be precise for a certain temperature. Temperature is assumed to be known and to be constant.

The battery must not be damaged, and the manufacturer provides information to avoid damage: battery **maximum ratings** are assumed to be known.

Though the processor's power consumption may not harm the battery, instantiation of some task may correlate with a (probably high) energy demand of some other component, for example a radio transmitter. Then, the component's energy demand is added to the task's demand, and the resulting discharge profile can be tested against the maximum ratings.

Tasks

The software running on the processor is assumed to be partitioned into units – tasks. The tasks are assumed to be **independent**. Their jobs are assumed to be **scheduled earliest deadline first**.

It is assumed that each job has to be finished a specific amount of time later than it is created. Is the job not finished in time, then its results are discarded. This **deadline** is assumed to be given for each task. All jobs from the same task have the same deadline.

A task may be instantiated periodically, partly periodically, or aperiodically, but this occurrence behaviour is not to change during run-time. The behaviour is assumed to be given in the form of an **event stream**.

Each job may need a different amount of execution time to finish. Two jobs of the same task may share the same upper bound on their execution time: this **worst-case execution time** is assumed, per task.

In analogy to worst-case execution time, a **worst-case execution energy** is assumed to be given for each task. Worst-case execution energy divided by worst-case execution time gives the average power consumption for each job of the same task.

Reduced processor frequency probably results in increased execution time. The work in this thesis makes use of a **linear relation of processor frequency to execution time**: decreasing processor frequency by a factor yields the same factorial increase of execution time.

4.3. Objective

With the premisses stated in the previous section, subject to the real-time feasibility of a fixed set of software tasks scheduled earliest deadline first, the following objectives are considerable:

a) Given battery and operating life: determine processor, minimise monetary cost.

b) Given processor and operating life: determine battery type and size, minimise size.

c) Given processor and battery: determine operating life, optimise life.

Solely the third one is tackled by the work described in this thesis.

This work is meant for a specific setup: the assignment of a set of software tasks to a processor to run on, and the assignment of battery to supply the system; power consumptions and execution times which are processor specific; and a model for computing the operating life which is battery specific. In this setup, discharge profiles serve for the coupling between the reduction of a processor's power dissipation and the computation of operating life. The objective is to maximise the time the system can run without the need to recharge or replace the battery.

Nevertheless, the tackled objective can be of use for the other two:

If there are multiple task implementations available, operating life can be maximised for each, and this eventually yields a best implementation that is to be deployed. With the maximal operating life in mind, the procedure to determine a best implementation is to be repeated for other processors: this will eventually yield several implementation-processor pairs that provide the desired operating life, and the cheapest solution can be chosen.

Thus, to solve the first objective with the third one makes a high motivation for fast, yet accurate, optimisation computations.

For the second objective, the assignment of a task set to a processor will yield a discharge profile. The profile contains information on the maximal current and on the average current that are drawn from a battery: both limit the number of batteries to use. The average current times the desired operating life gives a rough estimation on battery capacity: the battery's size.

Thus, if different battery types come with own battery models, the computed discharge profile will provide a suitable input. The faster the models can evaluate the input, the faster the designer will obtain an answer which battery type and size to use.

4.4. Conceptual Work-Flow

Hitherto, the attended level of abstraction was presented, and premisses and objectives were discussed. Here in this section, a sketch of the intended interplay of the various mechanisms is the subject.

The proposed flow is depicted in Figure 4.4. In the centre of which the domain of the optimisations presented in this thesis are placed.

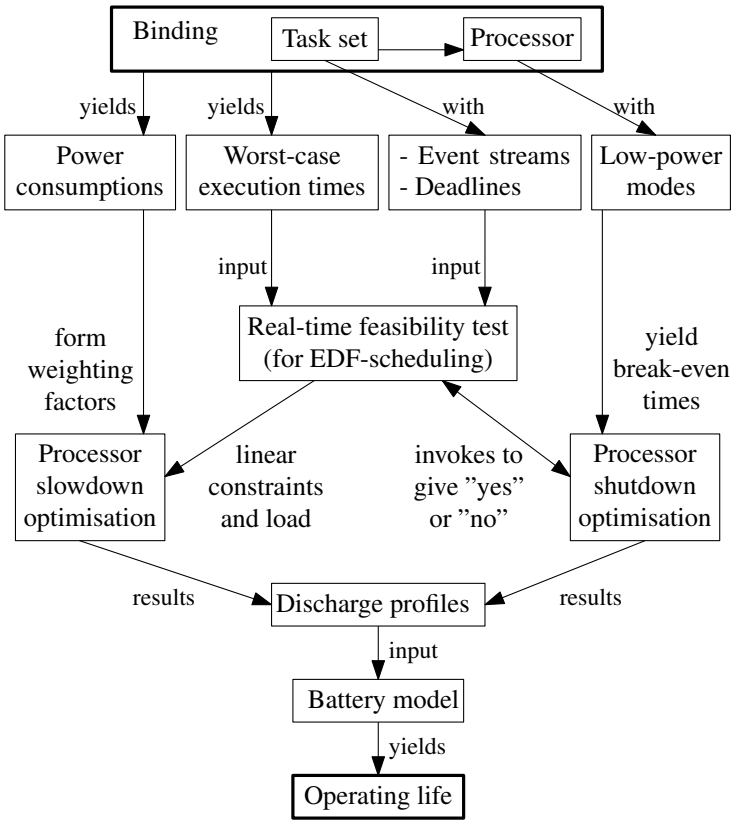


Figure 4.4.: Conceptual flow

The premisses were: a set of software tasks that is bound to a processor, and that is to be scheduled earliest deadline first; each task is determined by a deadline, an event stream, power consumption, and worst-case execution time; and the processor has low-power modes with known transition cost.

Now, worst-case execution times, event streams, and deadlines become inputs for the real-time feasibility test, in the approximated version. The test allows to model the real-time constraints in a linear way.

The linear constraints are used for the optimisation problems to calculate slowdown factors, i.e. least yet feasible processor frequencies, either global (for all tasks the same) or local (per task, not to be confused with per job). The former comes with a linear objective as we will see

in the following chapter; the latter optimisation problem incorporates the power consumption for each task together with the average processor load each task induces, and comes with a non-linear but convex objective.

Parallel to the slowdown optimisation, the shutdown optimisation takes place. Two shutdown methods are explored: periodic shutdown and task-dependent shutdown. The former transits periodically into low-power mode; the latter inherits the times for transition from the occurrence behaviour of another task that is to be determined. Either method models processor shutdown as a new task with execution time, event stream, and deadline. The deadline is set equal to execution time which is the same as giving the low-power task always highest priority to prevent preemption.

The configurations each shutdown method generates are tested with the real-time feasibility test, and if that evaluates to true, the duration for low-power mode is increased until the test evaluates to false. The best feasible parameters are taken as results.

Both optimisation steps yield two solutions which in total produce four different discharge profiles. Each profile is then evaluated with the battery model, and that profile is taken which leads to the longest operating life.

4.5. Comparison to Related Work

As stated in the chapter *State of the Art*, several researchers address the energy optimisation problem for embedded hard real-time systems. This section presents a table based overview how the research work described in this thesis fits in the work of other researchers.

Explanation of column headings that appear in both tables follow. Within each subsection further subsection specific headings are introduced.

Occurrence Behaviour This describes in symbols, which behaviour the work can handle. First, the reader should keep in mind that sporadic occurrence behaviour is always modelled as periodic with the period as minimal distance between two consecutive jobs of the same task. The ability to model periodic behaviour is denoted with symbol T, with or without subscript. Without subscript, the meaning is: all tasks may occur with one and the same period. A subscript τ means: all tasks are allowed to have individual periods. Task are allowed to jitter if symbol $\pm J_\tau$ is present. If the cell's entry is r_j, job release times, then the work is not on task basis, but on job basis, i.e. optimisation is performed on a schedule. If further r_j may be arbitrary but fixed, then the optimisation cannot handle future events beyond schedule, i.e. the entire set must be known with release times r_j and deadlines up to the lifetime or the end of the interval in which optimisation is supposed to take place. If symbols r and T appear together, then the optimisation is performed for a specific schedule and computes a look-ahead for future load on the basis of the tasks periods.

Deadlines This label describes deadline granularity and constraints. As in the paragraph before, deadlines may be: per task set, d without subscript; per task, subscript τ; or per job subscript j. Additionally, constraints may restrict deadlines to be equal to the corresponding period, $= T$, $= T_\tau$, less equal, $\leq$, or there may be no constraints.

4.5.1. Voltage Scaling/Processor Slowdown

Works dealing with processor slowdown to allow voltage reductions are compared to the solution presented in this thesis. Table 4.1, page 66, shows how it fits in.

In the case of voltage scaling, there is the advantage that the processor does not need to be restarted after scaling its voltage; furthermore, it might be possible to continue processing while the transition from one voltage to the next is taking place. This advantage allows the program that initiates the voltage scaling to be present after the voltage scaling, thus no external monitoring circuit is needed to watch over the voltage transition.

Remarks on subsection specific column headings follow.

Precedence Constraints This indicates whether task precedence is maintained by deadline in earliest deadline first manner, or data dependency (one task depends on the data of another) may define task precedence. Both may be present as well. A data dependency always means that tasks are considered non-preemptive.

Complexity This stands for computational complexity of algorithms. It is $|\Gamma|$ the symbol for the number of tasks, $|\mathscr{J}|$ the symbol for the number of jobs, #Volt. the symbol for the number of available voltages, and #Dev. depicts the number of available devices (hardware devices like hard disks).

Complexity may be increased by an integer problem: this is marked by "INT". Algorithms may be dependent on a specified precision: this is denoted by "Prec.". Algorithms may depend on other algorithms: the dependency on a real-time feasibility test is marked by "feas. test", then the complexity includes the complexity of the real-time feasibility test.

The term #Dep.(Γ) in line [SK2005] refers to the number of task dependencies within the task set.

Scaling Granularity The granularity ranges from one frequency per task set, denoted by f, over one frequency per task, denoted by f_τ, to one frequency per job, denoted by f_j.

Transition Overhead This field denotes whether the overhead that occurs when the frequency is changed is incorporated: is the overhead included, then this is marked by "yes", and "no" otherwise.

Finally, let us discuss the complexities of the algorithms given in [JG2004], and [RHE+2006]. Both algorithms invoke a real-time feasibility test for a number of times depending on a given precision ("log(Prec.)"). The test that is used in these works has the complexity of exploring all interval sizes up to a maximal size of $t_{\max} = \frac{U}{1-U}$: this yields a complexity of $|\Gamma| \cdot \max_{\tau\in\Gamma}\{t_{\max}/T_\tau\}$.[1] If we replace the used test – provided this is possible – by the approximated test used in this thesis, then we would obtain a lower complexity of $\log(\text{Prec.}) \cdot |\Gamma| \cdot$ #Appx.steps, which is still greater than the complexity of the solution presented in this thesis.

[1] Note: $t_{\max} \to \infty$ if $U \to 1$.

Table 4.1.: Processor Slowdown/Dynamic Voltage Scaling, Comparison to Related Work

	Occurrence Behaviour	Deadlines	Precedence Constraints	Scaling Granularity	Transition Overhead	Complexity
[IY1998]	T	$d = T$	no	f_τ	no	(INT) $\|\Gamma\| \cdot$#Volt.
[ASE+2004]	T	$d_\tau \leq T$	data, deadline, non-preemptive	f_τ	yes	exp($\|\Gamma\| \cdot \#Volt.$)
[RP2006]	T	$d = T$	deadline, non-preemptive	f_τ	yes	poly($\|\Gamma\|$) ·#Volt., #Dev.
[JG2004]	T_τ	$d_\tau \leq T_\tau$	deadline	f_τ	no	$\|\Gamma\|$ $\cdot \log$(Prec.) $\cdot O$(feas.test)
[RHE+2006]	$T_\tau \pm J_\tau$, bursts, other with accuracy losses	d_τ	deadline	f	no	log(Prec.) $\cdot O$(feas. test)
[ISG2003]	r_j, arbitrary, but fixed	d_j	deadline	f_j	For sleep mode only	$\|\mathscr{J}\|$
[SKL2006]	T	$d \leq T$	data, non-preemptive	f_τ	yes	$\|\Gamma\| \cdot$#Volt.
[SK2005]	T	$d \leq T$	data, non-preemptive	f_τ	yes	$\|\Gamma\| \cdot$#Dep.(Γ)
This work	$T_\tau \pm J_\tau$, other without accuracy losses	d_τ	deadline	f, f_τ	no	$\|\Gamma\|$ · #Appx.steps

4.5.2. Power Management/Processor Shutdown

In contrast to voltage scaling, the processor is not able to process any software during the transition into low-power mode and for the duration of the low-power mode. Therefore, online methods shutting down the processor require extra circuitry that monitors and maintains job queues, and determines the time to awake. Reference [LRK2003] suggests an FPGA to be next to the CPU. External hardware must be designed and placed as well, further, external hardware dissipates power as these devices need to stay online.

From the battery point of view, shutdown causes a rather high discharge current variation, see for example Table 2.2, page 34. For a good operating life forecast, it may be desirable to have predictive times when shutdown takes place. The solutions presented in [LJ2000] and [ASE+2004], as well as the solution presented in this thesis are the only that provide this.

The comparison of related work dealing with processor shutdown together with the work presented in this thesis is given in Table 4.2, page 68.

The reader should note: in opposite to all cited references, the work described in this thesis yields predictable processor shutdown and at the same time does not need any external hardware besides a programmable timer.

Remarks on subsection specific column headings and changes to the previous structure follow.

Task Dependencies This term is left out because all works regard deadlines as dependencies.

External Circuitry Entries in this column describe devices needed besides the CPU. The notation "Timer" stands for a programmable alarm that can wake up the processor. The term "Schedule monitor" denotes an external device that maintains the job schedule and computes procrastination. The device may be based on reference data, e.g. task periods, or time slots that were computed beforehand.

Complexity The time to transit into low-power mode and the duration to stay in the mode can be calculated depending on the tasks' specifications or based on the situation at run-time. Therefore, algorithms dealing with processor shutdown may operate "online", "offline", or in combination, and therefore, the notion of complexity is divided into two parts:

- **Online-Complexity:** This term denotes the associated computational complexity of the actions that have to be done 'online' either by the processor or by external circuitry.
- **Offline-Complexity:** This term denotes the associated complexity of the actions that have to be done 'offline', that is, at design time by either the designer or his or her computer.

Table 4.2.: Processor Shutdown/Power Management, Comparison to Related Work

	Occurrence Behaviour	Deadlines	Online Complexity	Offline Complexity	External Circuitry
[SCI2001]	r_j arbitrary, but fixed	d_j	Monitor job queue: $\|\mathscr{J}\|$·#Dev.	none	Schedule monitor
[CG2005]	r_j, T_τ	d_j	1) Monitor job queue: $\|\mathscr{J}\|$·#Dev., 2) Calculate procrastination: $\|\Gamma\|$	none	Schedule monitor
[LJ2000]	T_τ	d_τ	Schedule sporadic tasks in spare slots	1.List schedule for whole hyper-period, LCM, 2.optimise, precompute spare slots 3.save schedule in memory $\|\Gamma\| \cdot \max\{\text{LCM}/T_\tau\}$	Schedule monitor
[LRK2003], [JPG2004]	r_j, T_τ	$d_\tau \leq T_\tau$	Monitor job queue and calculate procrastination $\|\mathscr{J}\| \cdot$ #Dev. $\cdot \|\Gamma\|$	none	Schedule monitor
[NQ2004]	$T_\tau, \pm J_\tau$	$d_\tau \leq T_\tau$	1) Monitor job queue and calculate procrastination, $\|\mathscr{J}\| + \|\Gamma\|$, 2) Update delay bound on period deviation (jitter), $\|\mathscr{J}\|$	Pre-compute delay bound on assumed schedule, $\|\mathscr{J}\|$	Schedule monitor
This work	$T_\tau \pm J_\tau$, other without accuracy losses	d_τ	Program timer, 1	a) Periodic shutdown: $(\log(\text{Prec.}))^2 \cdot \|\Gamma\|$ · #Appx.steps b) Task dependent shutdown: $(\log(\text{Prec.}))^2 \cdot \|\Gamma\|^2$ · #Appx.steps	Timer

4.6. Summary

Qualitatively, in short, the work presented in this thesis enriches the state-of-the-art by extending the application of methods for saving energy from strict periodic task sets with deadlines equal to or shorter to task sets having arbitrary occurrence behaviour (in the form of an event stream) and arbitrary (but fixed) deadlines – tasks are to be scheduled earliest deadline first and share a single processor. The extension is done for both methods: processor slowdown and shutdown.

Regarding processor slowdown, the work presented in this thesis uses a less complex algorithm to determine slowdown factors, i.e. processor frequency and voltage pairs, per task and per task set.

Regarding processor shutdown, the work presented in this thesis performs offline optimisations which allow for an estimation of power consumption already at design time, after task set and processor have been specified.

The presented solutions regarding processor shutdown are the first to require only a programmable timer as extra circuit to wake up the processor.

Finally, the division into four parallel energy optimisations allows for recognising special properties of the battery to use. This division allows for a battery depended evaluation whether smooth low power consumption, or power consumption with power peaks and pauses is better regarding elongated operating life.

5. Modelling Processor Slowdown

A decrease of the supply voltage will increase the time for a signal to be delivered: it slows down the processor which in turn increases the processing time of each job. The main objective, a minimisation of the energy consumption, is subject to usable processing slack. The slack is determined through processor utilisation and task deadlines, and both are computable using a real-time feasibility test. The combination of test and objective then yields real-time feasible slowdown factors, further, performing the optimisation with the tasks' worst-case execution times will yield feasible solutions for all cases.

In this chapter, first, a characterisation of the event stream based energy consumption is presented, second, objectives for energy optimisation with slowdown are derived, and third the objectives are brought together with the linear constraints formed by the approximated demand bound function. The chapter closes with a comparison of two different objectives for per-task slowdown; the comparison depicts the importance of correct objectives for this optimisation problem.

5.1. Energy Bound

Every running job lets the processor consume electric energy. The sum of all energies determines the energy needed to run the set of tasks: for each task, weight the maximal energy needed to finish a job by the task's occurrence rate, then compute the sum of all weighted energies. This energy is at least required by the battery.

Definition 5.1.1. Let Γ be a set of tasks assigned the same processor. The energy bound function $\mathrm{ebf}^{\#} : \Gamma \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ denotes for every span $\Delta t \geq 0$ the maximal needed processing energy to run jobs of a task $\tau \in \Gamma$ within.
(See [LAS2006])

Again, as with the demand bound function, a way to compute this function is seeked.

Define, analogously to worst-case execution time, the maximal energy needed to finish any instance of task τ as worst-case execution energy w_τ. We replace the worst-case execution time by worst-case execution energy in the formulas for the demand bound, then we obtain the approximated energy bound function.

For each span let us sum up the number of jobs to complete within multiplied by their corresponding worst-case execution energy:

Lemma 5.1.2. *Let Γ be the set of tasks to be run on the same resource. Let d_τ be the deadline for task τ, w_τ be the worst-case execution energy, and $\left(a_\tau^{(n)}\right)_{n\in\mathbb{N}}$ be the event stream, respectively.*

Then

$$\begin{aligned} \mathrm{W} : \mathbb{R}_{\geq 0} &\to \mathbb{R}_{\geq 0} \\ \Delta t &\mapsto \mathrm{W}(\Delta t) := \sum_{\tau \in \Gamma} m(\tau, d_\tau, \Delta t) \cdot w_\tau \end{aligned} \tag{5.1}$$

is an overestimation on the energy demand for all intervals of length Δt.
(See [LAS2006])

The proof is analogue to the former proof for the demand bound function.

The function W can be approximated linearly, in the same way as the demand bound function:

Corollary 5.1.3. *With the same assumptions as for Lemma 5.1.2, let h_τ be the linear approximation of the event stream of task $\tau \in \Gamma$. Then, for all intervals of length $\Delta t \geq 0$, an upper bound on the energy demand is*

$$\mathrm{W}(\Delta t) = \sum_{\tau \in \Gamma} h_\tau(\Delta t) \cdot w_\tau. \tag{5.2}$$

Hitherto, the energy demand is computed for a processor that does not consume any energy when it is idle – an ideal processor. A non-ideal processor dissipates the power $P_{\text{idle}} \neq 0$ when no job is running.

Say, the processor consumes the power $P_\tau(t)$ when it runs an job of task τ at time t. Average this over time, and let P_τ denote a task's average power consumption. Now, let us separate the energy w_τ into power P_τ and worst-case execution time c_τ: $w_\tau = P_\tau \cdot c_\tau$.

Further, let process all jobs of task τ at the same processing frequency, and define $\tilde{c}_\tau(\Delta t)$ to be the maximal processing demand of task τ within any interval of length Δt.

Let Γ denote the set of all tasks to be assigned the same processor, P_{idle} be the processor's idle power consumption, P_τ be the (average) power consumption of task τ, and c_τ be the task's processing demand, then, the total energy consumed within a time span Δt is

$$E(\Delta t) = \sum_{\tau \in \Gamma} P_\tau \cdot \tilde{c}_\tau(\Delta t) + P_{\text{idle}} \left(\Delta t - \sum_{\tau \in \Gamma} \tilde{c}_\tau(\Delta t) \right). \tag{5.3}$$

The first term on the right hand side depicts the energy consumed for processing. The second term stands for the energy consumed while the processor stays idle. The optimisation goal of all methods for saving energy is a minimisation of the sum.

5.2. Characterisation of Processor Slowdown

In this thesis, only those methods for energy reduction are considered that reduce energy consumption at the cost of idle time. That is idle time is decreased; long-term utilisation, symbol U, which is the average utilisation the task set generates, is increased.

Suppose there is an interval length which has the following property: if all intervals with shorter length are shown to be real-time feasible, so will all longer intervals be real-time feasible. Real-time feasibility tests integrating such a bound will have reduced computational complexity.

Such a test bound was determined by Baruah *et al.* in [BRH1990]. It is based on the long-term utilisation. The authors showed that

$$\Delta_{\max} := \frac{U}{1-U} \tag{5.4}$$

is a feasible test bound. Is the utilisation considerably high, then this bound grows large: if the utilisation approaches 1, then the fraction is unlimited – mind that for a non-periodic task set, the hyper-period does not lead to solution either. Therefore, these kinds of bounds are insufficient for energy optimisation.

The linear approximated real-time feasibility test does not need such test bounds, and therefore the following energy optimisations will make use of this test.

5.2.1. Voltage and Frequency Scaling

Mainly, in terms of Equation (5.3), processor slowdown – also called voltage and frequency scaling – reduces energy by quadratically decreasing P_τ, linearly increasing $\tilde{c}_\tau$, and thus linearly decreasing the equation's second term.

Consider P_τ from the first term of Equation (5.3). From the Preliminaries we know running jobs of different tasks on the same processor in general yields different power consumptions on the same voltage level. Regarding the physical law for power, $P = \frac{V^2}{R}$, we can conclude that the resistance R is dependent on the running job: we have R_j for job j.[1] Now, let us take worst-case assumptions and generalise per job resistances to per task resistances, R_τ, that is we remove data dependency and retain implementation dependency. Then we have a power consumption that is per task and determined by $P_\tau = \frac{V^2}{R_\tau}$.

Second, according to the relation of supply voltage and circuit delay, Equation (2.13), page 32, an increase of the frequency implies an increase of supply voltage. That is, the speed defines a lower bound on processor voltage and reducing speed will reduce voltage. Let us assume this relation is proportional: $V \propto f$. Then we have

$$P_\tau \propto \frac{f^2}{R_\tau}. \tag{5.5}$$

For a reduction of the power dissipation, it is not necessary to assume the power to be proportional to the product on the right hand side. However, beyond the relation that decreasing frequency decreases voltage, it is necessary, that the voltage reduction outweighs the increase in execution time in terms of energy:

$$P(V(f_1)) \cdot c_1 \leq P(V(f_2)) \cdot c_2, \quad \text{if } f_1 \leq f_2, \tag{5.6}$$

with c_1, c_2 the corresponding execution times at frequency f_1, f_2 respectively.

Although, as seen in the chapter *State of the Art*, voltage scaling has been done individually for each job or even within a job; in the following, voltage scaling is done individually for each

[1] Of course resistance and power are dependent on material and other hardware related parameters, but these are *common* for all tasks.

task τ – f_τ – and globally for all tasks – f_Γ. The latter closely relates to a computation of the overall processor utilisation.

The per-task fractions $P_\tau = V^2/R_\tau$ correspond to the task's average *instruction base cost* plus *instruction switch cost*, both proposed in [TMW1994] and [LTMF1995]. Analogously, P_{idle} equals *base cost* of the idle task – there is no instruction switch. With the assumption that the power values P_τ at maximal supply voltage V_{supp} are given for each task, the objectives for optimisation of energy consumption with slowdown are developed.

5.2.2. Global Voltage Scaling

To reduce the frequency equally for all tasks in a task set Γ is called *global slowdown*. A reduced processor frequency f_Γ becomes the goal, and the frequency reduction becomes the optimisation variable.

Let f be the original processor frequency, then the factor $\gamma := \frac{f}{f_\Gamma}$ describes the slowdown by the frequency reduction. Let $c_\tau(\Delta t)$ be the maximal processing demand of task τ at maximal speed over all intervals of length Δt, then the processing demand at reduced speed is $\tilde{c}_\tau(\Delta t) = \gamma c_\tau(\Delta t)$.

Let $\tilde{P}_\tau$ be the reduced power consumption for a task τ. With the help of the proportionality (Equation (5.5)), we obtain $\tilde{P}_\tau \propto \frac{f_\Gamma^2}{R_\tau}$.

For an interval of size Δt, the product of processing demand at reduced speed, $\tilde{c}_\tau(\Delta t)$, and reduced power consumption, $\tilde{P}_\tau$, is the maximal energy demand of jobs of task τ within the interval. For the slowdown γ, we obtain:

$$\tilde{P}_\tau \cdot \tilde{c}_\tau(\Delta t) \propto \frac{f_\Gamma^2}{R_\tau} \cdot \gamma c_\tau(\Delta t) = \frac{f^2}{\gamma^2 \cdot R_\tau} \gamma c_\tau(\Delta t) = \frac{f^2}{\gamma \cdot R_\tau} c_\tau(\Delta t) = \frac{1}{\gamma} \cdot P_\tau \cdot c_\tau(\Delta t). \tag{5.7}$$

That is, an increased slowdown will result in a decreased energy demand.

The Objective: Now, let us derive the objective with the help of Equations (5.3) and (5.7). For all $\Delta t \geq 0$, it is to minimise:

$$E(\Delta t) = \frac{1}{\gamma} \sum_{\tau \in \Gamma} P_\tau \cdot c_\tau(\Delta t) + P_{\text{idle}} \left(\Delta t - \gamma \sum_{\tau \in \Gamma} c_\tau(\Delta t) \right). \tag{5.8}$$

The term $P_{\text{idle}} \Delta t$ is independent from the optimisation variable γ and will be removed. To avoid an objective individual for each time span Δt, the processing demand for each task is averaged to $\overline{c}_\tau := \lim_{\Delta t \to \infty} c_\tau(\Delta t)$, the long-term demand.

Thus, the objective function g_{target} becomes

$$\text{Minimise:} \quad g_{\text{target}} := \sum_{\tau \in \Gamma} \overline{c}_\tau \left(\frac{P_\tau}{\gamma} - \gamma P_{\text{idle}} \right). \tag{5.9}$$

This will increase γ as much as possible, because the first derivative is non-zero:

$$\frac{\partial g_{\text{target}}}{\partial \gamma} = \frac{-1}{\gamma^2} \sum_{\tau \in \Gamma} \overline{c}_\tau P_\tau - \sum_{\tau \in \Gamma} \overline{c}_\tau P_{\text{idle}} \tag{5.10}$$

$$\neq 0 \quad \text{for all } \gamma > 0 \tag{5.11}$$

and the second derivative is positive:

$$\frac{\partial^2 g_{\text{target}}}{\partial^2 \gamma} = \sum_{\tau \in \Gamma} \overline{c}_\tau P_\tau \frac{2}{\gamma^3} \tag{5.12}$$

$$> 0 \quad \text{for all } \gamma > 0. \tag{5.13}$$

Together, both derivatives show that the objective function g_{target} is unbounded for $\gamma \to \infty$.

Since the objective is unbounded, and we have only one optimisation variable, we may replace the objective by a simpler version that will also increase the variable as much as possible.

An alternative objective for the optimisation problem is

$$\text{Maximise:} \quad \gamma, \tag{5.14}$$

which is linear.

We will make use of this objective for the global slowdown optimisation problem.

Resulting Optimised Energy Demand: When the energy has been optimised with global voltage scaling and an optimal slowdown γ has been computed, then Equation (5.9) computes the resulting energy demand of the system. The demand can be used to compare with other energy minimisations.

Note, P_{idle} does not necessarily change by changing the processor frequency as the processor may have an idle mode, which is independent of voltage changes.

5.2.3. Local Voltage Scaling

Reducing the frequency individually for all tasks, but globally for all jobs of a task, is called local slowdown. A minimal frequency f_τ is to be determined, subject to real-time constraints.

Consider the common slowdown factor γ of the previous subsection replaced by a slowdown factor γ_τ that is individual for each task $\tau \in \Gamma$. Following the same argumentation as before, we obtain a decrease in energy, if we decrease the frequency, now on a per-task basis, the decreased frequency f_Γ is replaced by task individual frequencies f_{γ_τ}.

With this setting, Equation (5.7) becomes

$$\tilde{P}_\tau \cdot \tilde{c}_\tau(\Delta t) \propto= \frac{1}{\gamma_\tau} \cdot P_\tau \cdot c_\tau(\Delta t). \tag{5.15}$$

The objective: As before, let us derive the objective with the help of Equations (5.3) and (5.15). Now, for all $\Delta t \geq 0$, it is to minimise:

$$E(\Delta t) = \sum_{\tau \in \Gamma} \frac{P_\tau \cdot \tilde{c}_\tau(\Delta t)}{\gamma_\tau} + P_{\text{idle}} \left(\Delta t - \sum_{\tau \in \Gamma} \gamma_\tau \tilde{c}_\tau(\Delta t) \right) \tag{5.16}$$

Again, the term $P_{\text{idle}} \Delta t$ is independent from the optimisation variables γ_τ and will be removed. And again, to avoid an objective that is individual for each time span Δt, the processing demand for each task is averaged to $\overline{c}_\tau := \lim_{\Delta t \to \infty} c_\tau(\Delta t)$, the long-term demand.

Thus, the objective function g_{target}, now for local slowdown, becomes

$$\text{Minimise:} \quad g_{\text{target}} := \sum_{\tau \in \Gamma} \overline{c}_\tau \left(\frac{P_\tau}{\gamma_\tau} - \gamma_\tau P_{\text{idle}} \right). \tag{5.17}$$

Convexity of a function ensures, that each local minimum is a global one. A function is convex if its second derivative is positive definite. Here, the second derivative computes to a matrix: we have to prove that all eigenvalues are positive, which can be easily verified if the only non-zeros are positive and appear in the diagonal of the matrix.

In our case, the objective's second derivative is a matrix that has only positive entries in the diagonal:

$$\frac{\partial^2 g_{\text{target}}}{\partial^2 \vec{\gamma}} = \begin{pmatrix} 2\overline{c}_{\tau_1} P_{\tau_1} / \gamma_{\tau_1}^3 & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & 2\overline{c}_{\tau_n} P_{\tau_n} / \gamma_{\tau_n}^3 \end{pmatrix}, \tag{5.18}$$

with $\vec{\gamma} = (\gamma_{\tau_1}, \dots, \gamma_{\tau_n}), \tau_i \in \Gamma$.

Now, with Equation (5.17) we have a convex optimisation problem, since the event-stream based feasibility test provides linear constraints.

An Alternative, Linear Objective: The achieved objective might lead to longer computation times because it is not linear. Now, let us ignore all products concerning P_{idle}, retain $\overline{c}_\tau P_\tau$ as weighting factors, and invert the fractions $\frac{1}{\gamma_\tau}$. This yields a linear objective for maximisation:

$$\text{Maximise:} \quad \sum_{\tau \in \Gamma} \gamma_\tau \overline{c}_\tau P_\tau. \tag{5.19}$$

This alternative will also result in a maximal increase for every γ_τ, but it sets different weights on the optimisation variables.

After the constraints have been added to the objectives, the optimisation problems are complete, and both objectives will be compared: first, in terms of remaining idle time – which should be the slack that is exploitable for optimisation; and second, in terms of resulting average power consumption, which should be the optimisation goal.

Resulting Optimised Energy Demand: When the energy has been optimised with local voltage scaling, and optimal slowdown factors γ_τ have been computed for each task τ of the task set, then Equation (5.16) computes the resulting energy demand of the system. The demand can be used to compare with other energy minimisations.

Again note, P_{idle} does not necessarily change by changing the processor frequency as voltage changes are applied only to tasks, and the processor may have an idle mode which is independent of voltage changes.

5.3. Constraints for Processor Slowdown

Since available processing time is not the only constraint to consider, another constraint is to retain all deadlines, a real-time feasibility test will yield all time-related constraints for the optimisation problems.

The feasibility test presented in the Preliminaries involves to compute the demand bound function – the processing demand. A comparison of interval size and processing demand yields the slack to exploit. With the help of event streams, we can determine the demand bound function.

However, as shown in the beginning of the previous section, in the case of processor slowdown, interval sizes to test cannot be bound on a long-term utilisation base. The linear approximation of the event stream based calculation of the demand bound function solves this problem.

The solution was already given in Theorem 2.4.8, page 27. Further, with the theorem, the computational complexity is reduced down to polynomial in the general case, and down to linear complexity when assuming bounded approximation indices.

A drawback this solution carries is its inaccuracy: the solution underestimates the available slack. Experiments will show how much this problem weighs.

5.3.1. Constraints for Global Slowdown

The first constraint is simple but important: we want the optimisation to reduce the processor frequency, and ensure this by

$$\gamma \geq 1. \tag{5.20}$$

The second constraint origins from the long-term utilisation, which must not exceed 1. With the notions of Theorem 2.4.8, for each task τ in the task set Γ, let s_τ be an approximation at index k_τ of the respective event stream. Then ensure

$$\gamma \cdot \sum_{\tau \in \Gamma} s_\tau \cdot c_\tau \leq 1. \tag{5.21}$$

The third and last constraint origins from the linearisation of the demand bound function. With the notions of Theorem 2.4.8, for each task τ in the task set Γ, let h_τ be the linear approximation of its event stream. Then ensure

$$\gamma \cdot \mathrm{D}(\Delta t) = \sum_{\tau \in \Gamma} h_\tau(\Delta t) \cdot c_\tau \leq \Delta t, \tag{5.22}$$

for all $\Delta t \in \bigcup_{\tau \in \Gamma} \bigcup_{j=1,\ldots,k_\tau} \{a_\tau^{(j)} + d_\tau\}$, the latter is derived from Theorem 2.4.7, page 27.

All constraints are linear, and the objective is linear: we have a linear optimisation problem.

5.3.2. Constraints for Local Slowdown

This subsection generalises all constraints stated in the previous subsection.

The first constraint ensures that the frequencies are reduced:

$$\gamma_\tau \geq 1. \tag{5.23}$$

Again, the constraints of the second kind origin from the long-term utilisation, but now each approximation is weighted by the corresponding slowdown factor. With the notions of Theorem 2.4.8, for each task τ in the task set Γ, let s_τ be an approximation at index k_τ of the respective event stream. Then ensure

$$\sum_{\tau \in \Gamma} \gamma_\tau \mathrm{s}_\tau \cdot c_\tau \leq 1. \tag{5.24}$$

And the constraints of the third kind origin from the linearised event stream based demand bound function, but now with a vector of optimisation variables. With the notions of Theorem 2.4.8, for each task τ in the task set Γ, let h_τ be the linear approximation of its event stream. Then ensure

$$D(\Delta t) = \sum_{\tau \in \Gamma} h_\tau(\Delta t) \cdot c_\tau \cdot \gamma_\tau \leq \Delta t, \tag{5.25}$$

for all $\Delta t \in \bigcup_{\tau \in \Gamma} \bigcup_{j=1,\ldots,k_\tau} \{a_\tau^{(j)} + d_\tau\}$, according to Theorem 2.4.7.

Let us rewrite the constraints of the second and third kind into matrix form. Let $\Delta t_1, \ldots, \Delta t_m$ denote all test-points. As before, let h_i, with slope s_i, be the linear approximation of the event stream of task i. As in the previous section, let $\overline{c}_i$ denote the average demand of task i, here, we can compute the value to be $\overline{c}_i = c_i / s_i$.

Then, the linear constraints form the matrix:

$$\begin{pmatrix} \overline{c}_1 & \ldots & \overline{c}_n & 1 \\ h_1(\Delta t_1) \cdot c_1 & \ldots & h_n(\Delta t_1) \cdot c_n & \Delta t_1 \\ \vdots & & & \vdots \\ h_1(\Delta t_m) \cdot c_1 & \ldots & h_n(\Delta t_m) \cdot c_n & \Delta t_m \end{pmatrix} \cdot \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_n \\ -1 \end{pmatrix} \leq \vec{0}. \tag{5.26}$$

The global slowdown problem can also use this matrix by setting $\gamma_1, \ldots, \gamma_n = \gamma$. The matrix form can be used as constraint matrix for all three objectives. This simplifies constraint computation.

5.4. Comparing Linear and Non-Linear Objective for Local Slowdown

For global and local slowdown factor optimisation, we found out two objectives for each. In the global slowdown case, it could be shown that both objectives will solve the problem. In the local slowdown case, both objectives may have in common that every variable is increased as much as possible, but each objective makes use of different weights for the optimisation variables.

This section evaluates with two examples how both objectives distinguish. Two values are of interest, first, the remaining idle time after optimisation as indicator for used slack, and second, the average power consumption after optimisation, which is the primary optimisation goal.

For first evaluation, an aircraft controller example is taken from reference [TC1994]. Since it does not specify power consumptions for each task, each task – including the idle task – is assigned a value of 1 mW; and for variation, one task after the other – excluding the idle task – is set to a high power consumption of 100 mW. This setting points out a prioritisation to slow down the task with high power consumption first and afterwards to slow down the remaining tasks.

Figure 5.1, page 79, shows the remaining relative idle time, in the long-term sense, and for each objective after optimisation. On the horizontal axis, it is depicted which task was given a high priority by setting its power factor to 100 mW.

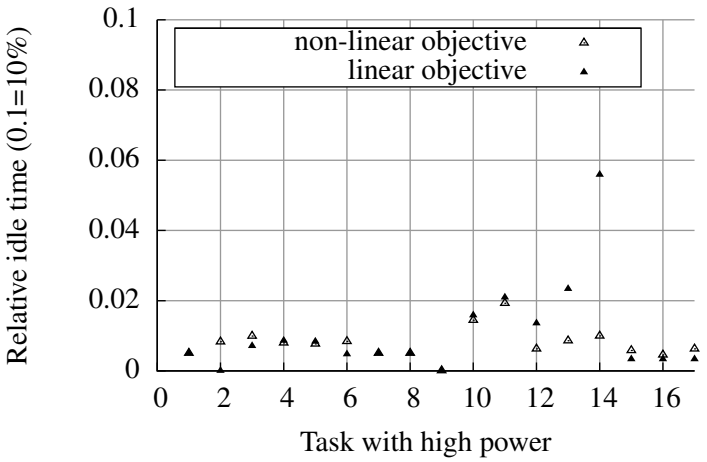


Figure 5.1.: Using Two Different Objectives to Obtain per Task Slowdown Factors, Remaining Idle Time for the Aircraft Controller Example

The figure shows that both objectives do not achieve similar results, although optimisation directions were forced by the power factor setting. The results do not even distinguish by a constant offset, instead, they alternate.

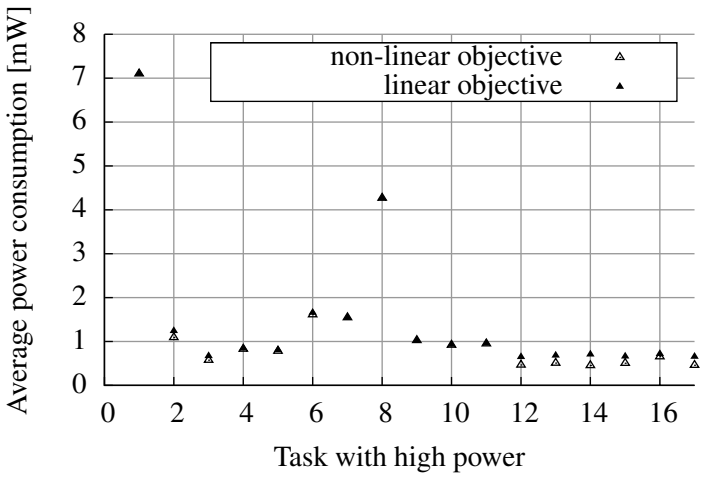


Figure 5.2.: Using Two Different Objectives to Obtain per Task Slowdown Factors, Resulting Average Power Consumption for the Aircraft Controller Example

Figure 5.2, shows now the achieved average power consumption. On the horizonal axis, it is depicted which task was given high priority by power factor adjustment.

First, the figure shows that the non-linear objective always achieves better results than the other. Second, together with Figure 5.1, it is shown that remaining idle time and average power consumption do not correlate. Thus, remaining idle time is not an indicator for power optimality.

The explanation for the last statement comes from the used real-time feasibility test. Of

course, either objective makes one out of the constraints given in Conditions (5.24), and (5.25) tight, but either objective tightens a different condition. Further, a tight condition does not indicate minimal idle time because the conditions depend on the approximation indices used: a condition can be tight because of over-approximation; the system's real idle time is not necessarily zero because some condition is tight. Therefore, for the figures, the optimisation variables were used to adjust the tasks' execution times, and afterwords, without approximation, the processing demand for a long interval size was calculated and diagrammed in the figure. Because the same interval size was used for all experiments with the aircraft controller, all results are comparable, regardless of objective and approximation.

Another example task set was taken for evaluating both objectives. This time, the PalmPilot example given in [LHW2002] was target of optimisation. The linear objective achieves 85.71 mW average power consumption, whereas the non-linear objective achieves 79.35 mW. This is already a significant gap. Now, the global slowdown method yields 81.36 mW in the average. This fastens the conclusion, that the linear objective for local slowdown points to a false direction.

5.5. Summary

This chapter characterised per-task and per-task set voltage scaling and presented their integration into an approximating real-time feasibility test.

As shown, global slowdown can be optimised by a linear program. Slowdown factors that are computed individually for each task were shown to require a non-linear objective. Both optimisation methods make use of the linearised approximation of the demand bound function. Both optimisations can use the same constraints: this eases computation.

In case the constraints cause a failing test due to approximation errors, the optimisation can be restarted with increased approximation indices – increased precision. However, the number of constraints depend linearly on the approximation indices, and therefore, the complexity of the problem is linear in the number of tasks times the approximation index for each task.

Furthermore, in case an increase of the number of approximation steps does not lead to feasible solutions, the task set can be interpreted to be infeasible. Then either the task set demands more processing time than available, or the binding of task set to processor is insufficient, and the designer of the embedded system may want to replace the processor.

The experiments in the last section show that remaining idle time is not a quality measure for judging energy optimisations with dynamic voltage scaling.

6. Modelling Processor Shutdown

A device that is put into low-power mode dissipates almost no power, depending on the sleep-state. But if parts or the whole processor are sent into a low-power mode, this leaves the disabled circuit unavailable for processing jobs. Further, the processor is passive: it needs to be woken up again. The back-transition requires a certain amount of energy and time, which will very likely produce a delay for the software to process next. Solutions for processor shutdown need to be verified by a real-time feasibility test.

This chapter first characterises processor shutdown to depict the difference to processor slowdown from the previous chapter, second, periodic shutdown is presented, and third, task-dependent shutdown is discussed.

6.1. Characterisation of Processor Shutdown

A difference to slowdown from the previous chapter is that a processor which is shut down is unavailable for processing. This fundamental difference to processor slowdown introduces two problems, first, the processor needs to be woken up, and because it cannot do the wake-up itself, extra circuitry for this special purpose is needed, second, the wake-up transition will take time and this time causes a delay before the next job is processed.

To avoid a delay, the time for wake-up needs to be predicted. Integration of wake-up time into the real-time feasibility test will produce results with no deadline violating delay.

If the transition into and out of low-power mode can interrupt a task because of, say, higher priority, then it needs to be considered how the processor can be woken up again and whether the low-power mode retains register contents, i.e. the current task's context. The result of these considerations might be an increased transition delay.

For the other way around, we want to incorporate the fact that the processor is blocked during low-power mode and to ease the estimation of the number of power mode switches, therefore we want to prevent preemption of the low-power mode:

Central Idea: A job with maximal priority will be processed without interruption. Maximal priority is ensured by setting deadline equal to execution time. For a common task, this cannot be done, because the execution time is hard to predict, but for processor shutdown with the duration determined at design time, this is an easy solution. Thus, all processor shutdown algorithms presented in the following sections will model the time to spent in low-power mode as a task λ with execution time, deadline, and occurrence behaviour, where duration is set equal to execution time and deadline.

6.1.1. Efficiency of Processor Shutdown Methods

The transition delay as well as the cost in energy for transition into and out of low-power mode require the duration to spend in low-power mode to have a minimal size: break-even time, see the Preliminaries.

The break-even time must be subtracted from the time spent in low-power mode, leaving over the time where energy is saved. Clearly, an algorithm for processor shutdown has the objective to increase this time as much as possible.

For processor shutdown, energy savings are subject to real-time constraints, as was the case for processor slowdown, and additionally, savings are subject to the number of transitions into low-power mode, as every transition reduces energy saving time.

The number of switches to and from low-power mode will provide an estimation on the effective energy that is saved.

Definition 6.1.1. Let $t_{\text{lp}}(\Delta t)$ denote the least time reserved for low-power mode within an interval of length Δt, and let #transitions(Δt) be the maximal number of transitions into and out of low-power mode occurring within an interval of length Δt. Then we define the effectiveness of a shutdown method as the fraction of time energy is saved within an interval of size Δt:

$$\text{Eff}(\Delta t) := \frac{1}{\Delta t}(t_{\text{lp}}(\Delta t) - \#\text{transitions}(\Delta t) \cdot t_{\text{be}}). \tag{6.1}$$

In the long-term, it is

$$\lim_{\Delta t \to \infty} \text{Eff}(\Delta t) \tag{6.2}$$

the time when the processor is in low-power mode *and* energy is actually saved: the effective low-power portion. This provides a means to compare different policies for processor shutdown.

6.1.2. Energy Consumed with Application of Processor Shutdown

Since processor slowdown and shutdown may yield different power consumptions, both, the slowdown and its variants, and the shutdown and its variants need to be compared.

Now, as both exploit available idle time, both can be compared on that basis, but keep in mind the conclusion of the last chapter that remaining idle time is no valid indicator. Therefore, processor slowdown and shutdown need to be compared based on energy consumption.

A first step is the computation of an energy bound according to Equation (5.3), page 72, now for the processor shutdown. Let us insert the time spent in low-power mode, t_{lp}, the maximal number of transitions occurring within any interval of length Δt, #transitions(Δt), the energy consumed while transiting, E_{switch}, the break-even time, t_{be}, and the power consumption of the low-power mode, P_{lp}, into Equation (5.3):

$$E(\Delta t) = \sum_{\tau \in \Gamma} P_\tau \cdot c_\tau(\Delta t) + P_{\text{idle}} \left(\Delta t - t_{\text{lp}} - \sum_{\tau \in \Gamma} c_\tau(\Delta t) \right) + P_{\text{lp}} \left(t_{\text{lp}} - \#\text{transitions}(\Delta t) \cdot t_{\text{be}} \right) + E_{\text{switch}} \cdot \#\text{transitions}(\Delta t). \tag{6.3}$$

This result allows to compute the energy needed, after optimisation with processor shutdown was done. Computation of the original equation for processor slowdown will provide a valid comparison based on energy consumption.

6.2. Periodic Processor Shutdown

A periodic task which has its deadline equal to its period will demand a uniform load. It is the goal of global processor slowdown to compute this load because the processor frequency can be reduced safely by this value. But the load can also be used to introduce a periodic task into the task system. Periodic processor shutdown exploits this advantage.

According to the following theorem, global slowdown and insertion of a periodic task with deadline equal to period are the same:

Theorem 6.2.1. *Let the lowest processor speed that preserves deadlines be denoted by α, and let this speed be relative to the maximal speed. Then the insertion of a task with execution time c, period T, deadline $d = T$, and with $\frac{c}{T} = 1 - \alpha$ yields a real-time feasible task set that cannot be slowed down anymore.*

Proof. It is α the lowest relative speed, in other words, the relation $\Delta t\,\alpha \geq \mathrm{dbf}(\Delta t)$ holds for all interval sizes $\Delta t > 0$.

Define $\beta := 1 - \alpha$, then for all $\Delta t' > 0$ with $\Delta t'\alpha = dbf(\Delta t')$ it follows $\Delta t' = \mathrm{dbf}(\Delta t') + \beta \Delta t'$. It was $\frac{c}{T} = 1 - \alpha$ which is equal to β, thus we have $\Delta t' = \mathrm{dbf}(\Delta t') + \frac{c}{T}$, which means the insertion of the new task yields feasible situations. □

The theorem is important, because it defines an upper bound on the load that the new task may introduce.

The new task represents the transition into and out of low-power mode, and the task will have deadline set to execution time, which is a stronger condition than in the theorem: the load the low-power task may introduce must probably be lower than the upper bound allows.

6.2.1. Periodic Shutdown of the Processor

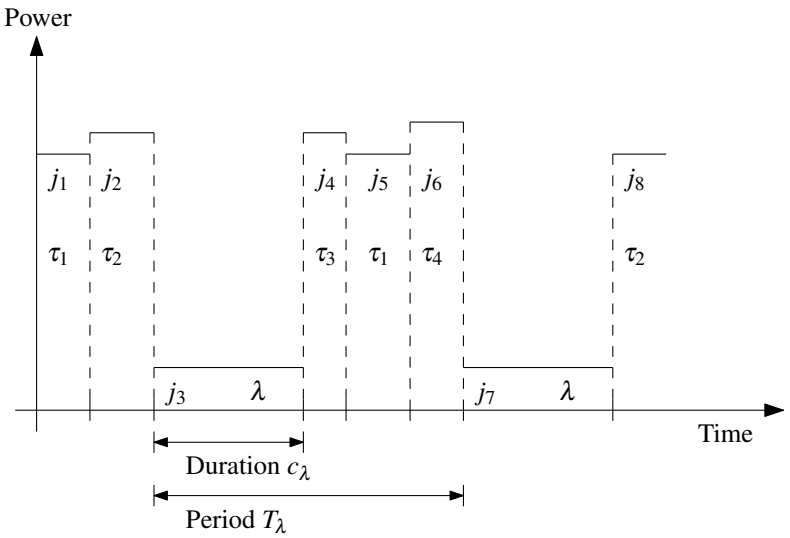


Figure 6.1.: Periodic Shutdown

An interval executed at low-power can occur periodically and therefore be scheduled and modelled as a task, λ. According to the least available idle time in busy situations, the execution time c_λ of this task is adjusted. To give the task highest priority – to prevent preemption – the deadline is set equal to the execution time. Then the period T_λ is minimised subject to the task set's real-time feasibility.

Figure 6.1, page 83, shows a sample time-line with applied periodic shutdown. The jobs j_i are enumerated, and each is marked with its corresponding task τ_k. The jobs marked with "λ" depict the low-power jobs where the processor is put into low-power mode for a duration of c_λ time units.

For the efficiency that the method induces, the period T_λ defines the number of mode transitions, and the execution time c_λ defines the duration spend in low-power mode. Let t_{be} be the break-even time for one mode switch, then we obtain as effectiveness

$$\text{Eff}(\Delta t) = \frac{c_\lambda}{T_\lambda} - \frac{\Delta t}{T_\lambda} \cdot \frac{t_{\text{be}}}{\Delta t}. \tag{6.4}$$

With the definition $\alpha := \frac{c_\lambda}{T_\lambda}$, it follows

$$\text{Eff} = \alpha \cdot \left(1 - \frac{t_{\text{be}}}{c_\lambda}\right). \tag{6.5}$$

6.2.2. Determination of the Parameters 'Period' and 'Duration'

Algorithm 6.2.1: Function `estimate`$(c_{\min}, c_{\max}, t_{\text{be}}, \Gamma, \alpha)$

Input: EDF scheduled Task set Γ, break-even time t_{be}, Utilisation α to use, minimal and maximal duration $c_{\min}$,$c_{\max}$

Output: Duration c_{solve}, Period T_{solve}, and Efficiency Eff

```
1  (Eff, c_solve, T_solve) := (−1, 0, 0);
2  while c_max − c_min > ε_c do
3      c := c_min + (c_max + c_min)/2;
4      T := c/α;
5      if feasible(Γ and low-power task(c,T)) then
6          c_min := c;
7          Eff_c := α · (1 − t_be/c);
8          if Eff < Eff_c then
9              (Eff, c_solve, T_solve) := (Eff_c, c, T);
10         endif
11     else
12         c_max := c;
13     endif
14 endw
```

The parameters 'duration' and 'period' are determined using the functions depicted in algorithms 6.2.1 and 6.2.2, pages 84 and 85. The latter repeatedly calls the former.

A real-time feasibility test is called to validate the feasibility of the parameter pair. The used test is the test from Theorem 2.4.8, page 27 (for the test see also [LAS2007]).

Algorithm 6.2.2: Function `apply_periodic_shutdown`(Γ, t_{be})

Input: EDF scheduled Task set Γ, break-even time t_{be}
Output: Duration c_λ and period T_λ of shutdown task, Efficiency Eff_{best}

1 $\alpha_{max} := 1$ - lowest_relative_speed(Γ);
2 $\alpha_{min} := 0$;
3 $c_{max} := \min\{\Delta t - D(\Gamma \setminus \{\tau\}, \Delta t > 0\}$;
4 $c_{min} := t_{be}$;
5 $\text{Eff}_{best} := 0$;
6 **while** $\alpha_{max} - \alpha_{min} > \varepsilon_\alpha$ **do**
7 $\quad \text{Eff}_\alpha := -1$;
8 $\quad \alpha := \alpha_{min} + (\alpha_{max} - \alpha_{min})/2$;
9 $\quad (\text{Eff}_L, c_L, T_L) :=$ `estimate`$(c_{min}, c_{max}, t_{be}, \Gamma, \alpha - \varepsilon_\alpha)$;
10 $\quad$ **if** $\text{Eff}_L > 0$ **then**
11 $\quad\quad (\text{Eff}_R, c_R, T_R) :=$ `estimate`$(c_{min}, c_{max}, t_{be}, \Gamma, \alpha + \varepsilon_\alpha)$;
12 $\quad\quad$ **if** $\text{Eff}_L < \text{Eff}_R$ **then**
13 $\quad\quad\quad (\text{Eff}_\alpha, c_\alpha, t_\alpha) := (\text{Eff}_R, c_R, T_R)$;
14 $\quad\quad\quad \alpha_{min} := \alpha$;
15 $\quad\quad$ **else**
16 $\quad\quad\quad (\text{Eff}_\alpha, c_\alpha, t_\alpha) := (\text{Eff}_L, c_L, T_L)$;
17 $\quad\quad\quad \alpha_{max} := \alpha$;
18 $\quad\quad$ **endif**
19 $\quad$ **else**
20 $\quad\quad \alpha_{max} := \alpha$;
21 $\quad$ **endif**
22 $\quad$ **if** $\text{Eff}_{best} < \text{Eff}_\alpha$ **then**
23 $\quad\quad (\text{Eff}_{best}, c_\lambda, T_\lambda) := (\text{Eff}_\alpha, c_\alpha, T_\alpha)$;
24 $\quad$ **endif**
25 **endw**

Given task set Γ, break-even time t_{be}, and utilisation α, then Algorithm 6.2.1 determines the period T_{solve} and duration c_{solve} for the low-power task yielding maximal efficiency, Eff. This is done by interval bisection over a range of possible durations $[c_{min}, c_{max}]$, which are also input to the function. Interval bisection stops as soon as the interval size drops below the error margin ε_c. A duration c is rejected when the feasibility test (Line 5) fails, then the lower half of the interval is taken. In the other case, the upper half is taken. This will maximise the duration c for a given utilisation α. The function will return a negative efficiency if no duration within the given range is found to be real-time feasible.

The idea behind this interval bisection is: assume a given utilisation α, then set the period to the fraction of duration by utilisation; and if a duration c_1 that is less than a duration c_2 is not real-time feasible, than c_2 cannot yield a feasible solution because the higher duration denotes a

stronger condition – it inhibits the processor from processing for a longer time than c_1.

The major part is done by Algorithm 6.2.2. It explores the design space of possible utilisations α, which the low-power task may introduce. This is done by interval bisection and gradient decision. The direction providing better efficiency is taken. If for a given α no feasible parameters were found, the direction to lower utilisation is taken.

Possible utilisations α are bounded above by Theorem 6.2.1 and bounded below by 0. Possible durations c are bounded above by the least available idle time within any interval, and bounded below by the break-even time.

The interval bisection is done by estimation of the efficiencies that are slightly left of the interval centre and slightly right of it. 'Slightly' means very close to the centre, i.e. as close as no extrema fall between interval centre and 'slightly' next to it: we speak of gradient based decision. The interval half with the better efficiency in it is taken next. The bisection ends when the interval size drops below the error margin ε_α. This margin also defines 'slightly'.

Complexity of the Solution: Since the intervals of the solution space are halved by each iteration, the algorithm will converge. Interval bisection has logarithmic complexity. The total complexity of the algorithm is

$$\begin{aligned} O(\text{Algorithm 6.2.2}) &= \log_2\left(\frac{\alpha_{\max}-\alpha_{\min}}{\varepsilon_\alpha}\right)\cdot 2\log_2\left(\frac{c_{\max}-t_{\text{be}}}{\varepsilon_c}\right)\cdot O(\text{feas. test}) \qquad (6.6)\\ &\leq \log_2\left(\frac{1}{\varepsilon_\alpha}\right)\cdot 2\log_2\left(\frac{\min\{d_\tau - c_\tau : \tau\in\Gamma\}}{\varepsilon_c}\right)\cdot O(\text{feas. test}). \end{aligned}$$

If each task is approximated after, say x steps, then it is $O(\text{feas. test}) = x\cdot\#\text{tasks}$, thus the algorithm is very fast.

Assumptions for the Solution: The algorithm is based on two statements. First, for each α there is a maximal duration c that induces a real-time feasible task, and a maximal efficiency.

Clearly, if there were two different solution pairs $(c_1,T_1),(c_2,T_2)$ with $\alpha=\frac{c_1}{T_1}=\frac{c_2}{T_2}$, $c_1>c_2$, then the former pair would yield higher efficiency according to Equation (6.5). That is, the first assumption holds.

Second, if an increase in utilisation α yields a decrease in efficiency, then utilisation is to be decreased; and if a decrease in utilisation yields a decrease in efficiency, then utilisation is to be increased. These decisions are based on the assumption, that the function depicting efficiency along a range of utilisations has one single maximum, i.e. every local is a global one.

Let us validate whether this second assumption is true with an example (from [LHW2002]).

Figure 6.2, page 87, depicts the maximal efficiency that can be obtained according to Algorithm 6.2.1, Function `estimate()`. The efficiency is diagrammed along a range of utilisations α for a break-even time of 3 ms, and for several approximation indices (test indices). The curve family shows how the maximal efficiency grows for growing test indices. Except for an approximation index of 1, where each event stream is approximated after the first element, all curves show several local extrema.

The question why the problem is not convex arises. The break-even time is a constant, and the efficiency, Equation (6.5), page 84, is convex because it has similar structure as the non-linear

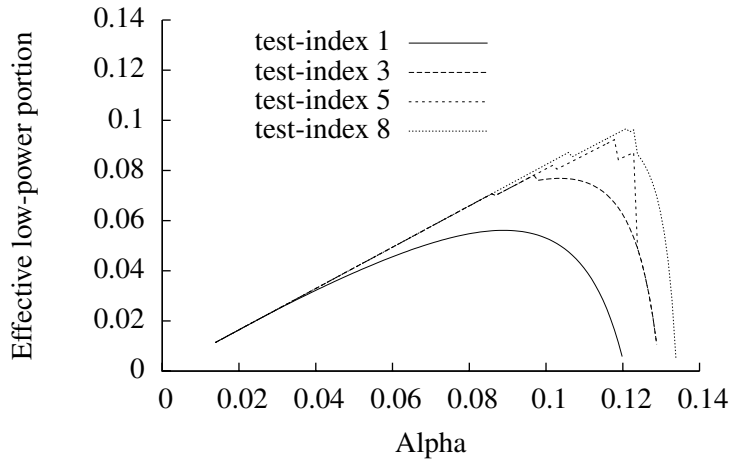


Figure 6.2.: Several Local Extrema of Utilisation by Efficiency Graphs in Case of Periodic Shutdown

Algorithm 6.2.3: Computation of Parameters for Periodic Shutdown for Set of Break-Even Times

Input: EDF scheduled Task set Γ, decreasing set of break-even times $t_1 > t_2 > \cdots > t_n$
Output: Set of solutions containing a pair (duration,period) for each break-even time

$c_{\text{pre}} := t_n$;
$T_{\text{pre}} := 1$;
for $t_{be} := t_1, \ldots, t_n$ **do**
    (c, T, Eff)=Result of Algorithm 6.2.2;
    $\text{Eff}_{\text{pre}} = \frac{c_{\text{pre}}}{T_{\text{pre}}} \cdot \left(1 - \frac{t_{\text{be}}}{c_{\text{pre}}}\right)$;
    if $\text{Eff} < \text{Eff}_{pre}$ **then**
        solution$(t_{\text{be}}) := (c_{\text{pre}}, T_{\text{pre}})$;
    else
        solution$(t_{\text{be}}) := (c, T)$;
        $(c_{\text{pre}}, T_{\text{pre}}) := (c, T)$;
    endif
endfor
$(c_{\text{pre}}, T_{\text{pre}})$:=solution(t_n);
for $t_{be} := t_{n-1}, \ldots, t_1$ **do**
    if $c_{pre} > t_{be}$ **then**
        solution$(t_{\text{be}}) := (c_{\text{pre}}, T_{\text{pre}})$;
    endif
    $(c_{\text{pre}}, T_{\text{pre}}) :=$ solution(t_{be});
endfor

objective for local slowdown. Further, in the previous chapter, the constraints given by the real-time feasibility test where shown to be linear, i.e. convex. Increasing the execution time was also shown to be linear. This leaves over the changes to the set of test points which are made whenever the number of approximation steps changes, or a deadline changes.

We found the second assumption false, but the next paragraph will show how to downsize this problem.

Determination of 'Period' and 'Duration' Along a Range of Break-Even Times: If periodic shutdown is to be computed for a set of break-even times, a set of solutions is obtained. The duration parameter of one solution may fit for several break-even times: a solution for a high break-even time always fits for all lower break-even times.

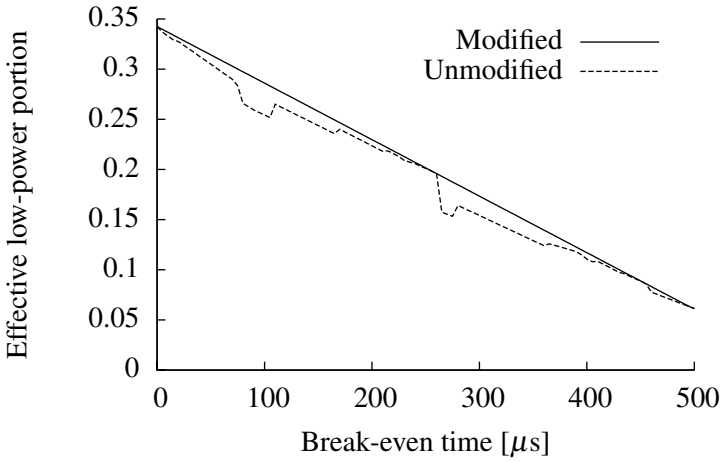


Figure 6.3.: Modified Versus Unmodified Algorithm for the Computation of Parameters for Periodic Shutdown

The solution to downsize the problem from the previous paragraph is depicted in Algorithm 6.2.3, page 87. The idea is to compute solutions along decreasing break-even times, to compare new solutions with the old, and finally to pick the better one. Since this may still leave over gaps, the solution set is also inspected in the direction of increasing break-even times; it is tested, if a solution also fits for a larger break-even time and if so, it is preferred. This will yield a smooth relation of break-even time to efficiency.

Let us validate the modification with the example task set from reference [TC1994].

Figure 6.3, depicts the obtained efficiency (according to Equation (6.5)) for periodic shutdown along a range of break-even times. Both the results of Algorithm 6.2.2, and of the proposed modification (Algorithm 6.2.3) are depicted. In both cases, each event stream has been approximated with 10 steps. The unmodified solution shows several spikes that have a lower efficiency than their neighbours, whereas the modified solution shows a smooth curve with monotonically descending efficiency along monotonically ascending break-even times.

6.3. Task-Dependent Shutdown of the Processor

Periodic shutdown will achieve a drawback on task sets that have a low utilisation and with some tasks having very short deadlines, because this combination leads to rare busy situations that have short idle times within. In this category fall task sets with a possible slack of $s < 1 - U$, an example is the task set of Table 8.2, page 132.

The short deadline of some task as restricting constraint is avoided if the low-power duration never intersects with the execution of the corresponding jobs. This is achieved if transition into low-power mode (low-power task λ) inherits the occurrence behaviour of the task with the short deadline, and the processor wakes up the latest before jobs of the special task must be executed.

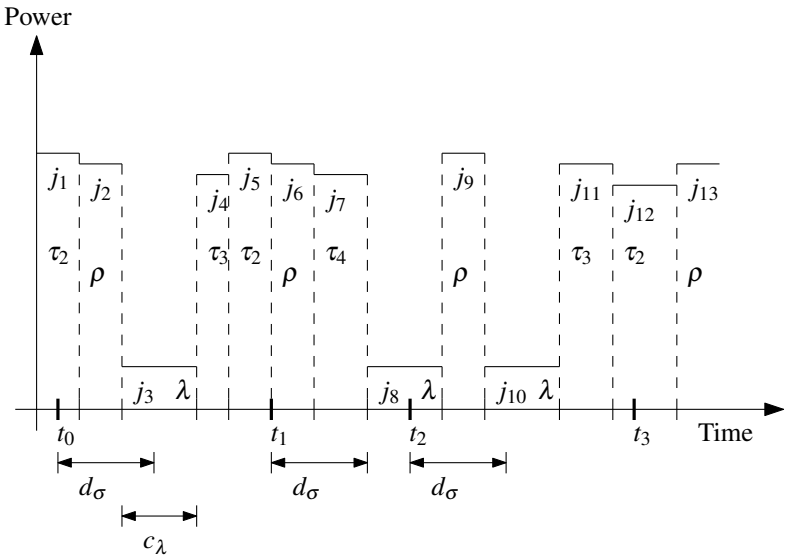


Figure 6.4.: Task-Dependent Shutdown, Every Instantiation of ρ

In Figure 6.4 thirteen jobs are depicted, each marked with the task it corresponds to. The low-power jobs are marked with "λ", have a duration of c_λ, and their latest start time inherits the occurrence behaviour of task ρ. Jobs for task ρ are created at times $t_0, \ldots, t_3$. Each low-power job is executed no later than d_σ time units after a job of ρ is created.

Algorithms for task-dependent shutdown have to determine the special task ρ out of the task set Γ, have to compute the latest relative start d_σ for the low-power mode, and the duration for the low-power mode c_λ.

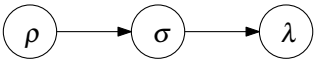
The intersection avoidance is a relaxation from the synchronicity assumption, which has been taken for the real-time feasibility test. A discussion on the new situation is required and is given in the following subsection. The next subsections will present an algorithm that determines parameters for task dependent processor shutdown and modifications of the algorithm.

6.3.1. The Case of Two Never Overlapping Consecutive Executed Tasks

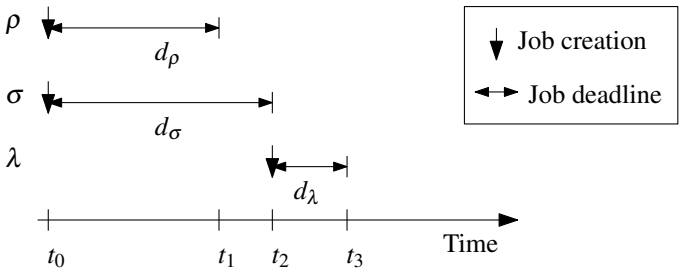
Let Γ be a set of tasks. Suppose there are two tasks ρ, λ in the task set: ρ triggers λ, and the job of λ has to finished before the next job of ρ has to to be finished; thus no jobs will overlap.

Suppose too, that a job of task λ, if running, has always highest priority.

For a better understanding of the situation, let us introduce a third task into the task graph, σ, which's purpose is to represent the trigger of λ. Any job of task σ will not consume any execution time. This dependency relation and the corresponding relation of the deadlines is depicted in Figure 6.5.



(a) Dependency Relation



(b) Deadline Relation

Figure 6.5.: Relations of Three Consecutive Tasks, It is $d_\lambda := c_\lambda$, $d_\sigma + d_\lambda \leq d_\rho + a_\rho^{(2)} - c_\rho$, and $d_\rho \leq d_\sigma$

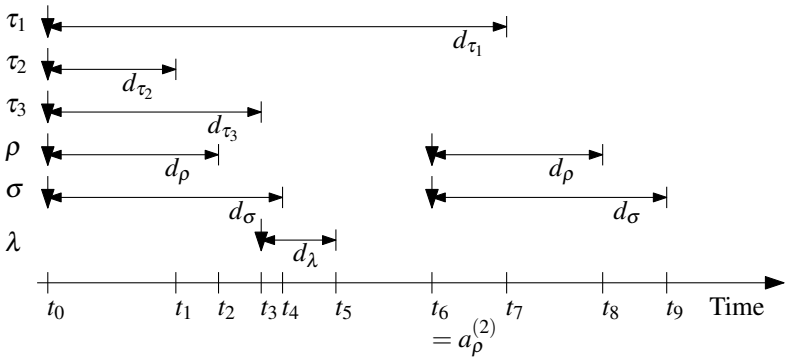
The deadline of task σ is set equal to the latest time to trigger low-power task λ, and to avoid intersected execution, this is later than the deadline of task ρ. Task λ is assigned highest priority by setting deadline equal to its execution time, $d_\lambda := c_\lambda$. This situation is depicted in Figure 6.5(b). Tasks ρ and σ are instantiated synchronously at time point t_0: the job of task ρ expires at time point t_1; the job of σ expires at t_2 and triggers λ; the job of λ is finished at t_3.

Further, the latest time to finish an instance of λ must be earlier or right before the latest start time of the next instance of ρ, which is equal to $d_\rho - c_\rho + a_\rho^{(2)}$, with $a_\rho^{(2)} \in \left(a_\tau^{(n)}\right)_{n\in\mathbb{N}}$.

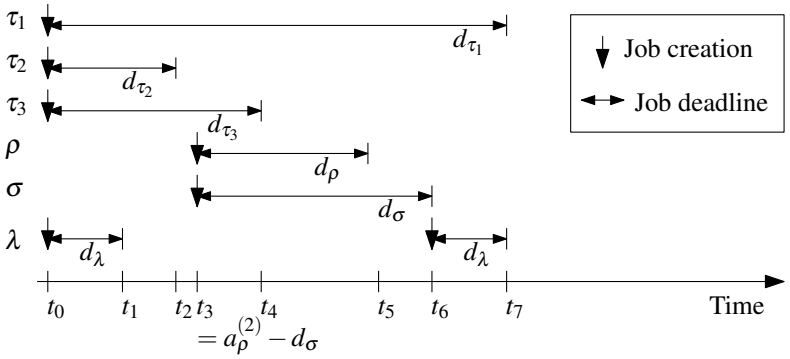
Two Worst-Cases: Since synchronicity is not given for tasks ρ and λ, two situations occur: one where all tasks except task λ are instantiated synchronously, and one where all tasks except ρ are instantiated synchronously. Each situation has the asynchronous task occurring as early as possible. Both situations are depicted in Figure 6.6, page 91.

In Figure 6.6, Case A, tasks $\tau_1, \tau_2, \tau_3, \rho, \sigma$ are instantiated at time point t_0. Task λ is instantiated at time point t_3, which is earlier than deadline d_σ expires and later than the finishing time of tasks τ_2, τ_3, ρ. Tasks τ_2, τ_3, ρ each have a higher priority than σ. At time point t_6 tasks ρ and σ are instantiated again, this is the earliest time: because with $t_6 = a_\rho^{(2)}$, time point t_6 equals the minimal time to pass between two instantiations of task ρ.

In Figure 6.6, Case B, tasks $\tau_1, \tau_2, \tau_3, \lambda$ are instantiated at time point t_0. Tasks ρ and σ are instantiated as early as possible, which is $a_\rho^{(2)} - d_\sigma$ time units after t_0: because task λ is



(a) Case A: $\tau_1, ..., \tau_3$ Triggered Synchronously to ρ and σ, It is $t_6 = a_\rho^{(2)}$



(b) Case B: $\tau_1, ..., \tau_3$ Triggered Synchronously to λ, It is $t_3 = a_\rho^{(2)} - d_\sigma$

Figure 6.6.: Never-Overlapping Execution: Two Worst-Cases

instantiated no later than d_σ time units after the last instantiation of task ρ, it needs to pass $a_\rho^{(2)} - d_\sigma$ time units before the next job of task ρ is created.

In Case B the next instance of task ρ is to be finished no later than

$$a_\rho^{(2)} - d_\sigma + d_\rho. \tag{6.7}$$

Recognise that tasks ρ and λ are asynchronous: then, for Case A, the demand is denoted by

$$\begin{aligned} \mathrm{D}_A(\Delta) = &\sum_{\tau \in \Gamma \setminus \{\lambda\}} m(\tau, d_\tau, \Delta) c_\tau \\ &+ m(\rho, d_\sigma + d_\lambda, \Delta) c_\lambda, \end{aligned} \tag{6.8}$$

whereas for Case B the demand is denoted by

$$\begin{aligned} \mathrm{D}_B(\Delta) = &\sum_{\tau \in \Gamma \setminus \{\rho, \lambda\}} m(\tau, d_\tau, \Delta) c_\tau \\ &+ m(\rho, c_\lambda, \Delta) c_\lambda \\ &+ m(\rho, d_\rho + a_\rho^{(2)} - d_\sigma, \Delta) c_\rho. \end{aligned} \tag{6.9}$$

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

Lemma 6.3.1. *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.*
(See [LS2009])

Proof.

$$\text{If } \Delta t = c_\lambda, \text{ then} \qquad \mathrm{D}_A(\Delta t) - \mathrm{D}_B(\Delta t) = -c_\lambda. \tag{6.10}$$

$$\text{If now } \Delta t = d_\rho = a_\rho^{(2)} \in \left(a_\rho^{(n)}\right)_{n\in\mathbb{N}}, \text{ then} \qquad \mathrm{D}_A(\Delta t) - \mathrm{D}_B(\Delta t) = c_\rho. \tag{6.11}$$

The former example may repeat for $\Delta t = c_\lambda + a_\rho^{(2)}$, $a_\rho^{(2)} \in \left(a_\rho^{(n)}\right)_{n\in\mathbb{N}}$, and the latter for $\Delta t = d_\rho + a_\rho^{(j)} - d_\sigma$, $j = 2,3,4,\ldots$. Thus, in general, the function D_A is not always greater than D_B, and vice versa. □

Since both demand bound functions start with all tasks being synchronous – except for tasks ρ and λ – there are no other demand bound functions denoting further worst-cases that are not already included here.

We conclude: if a set of preemptive tasks contains tasks that depend on each other – precedence constraints, a task graph – and a task with highest priority is involved, then two demand bound functions have to be identified; one function to test the situation when the root of the task graph is triggered synchronously with all other tasks, and the other to test the situation when the high priority task is triggered synchronously with the other tasks.

Finding a Feasible Latest Start Time for Two Consecutive Tasks: In Algorithm 6.3.1, page 93, both worst-case situations are tested. It has the purpose to find a latest start time for task λ, which has the execution time c and highest priority. The occurrence behaviour of task λ is coupled to a special task ρ and is given by $\left(a_\lambda^{(n)}\right)_{n\in\mathbb{N}}$.

Within the interval $[d_{\min}, d_{\max}]$, the algorithm searches by interval bisection for a feasible time d for task λ to instantiate latest possible after an instantiation of task ρ. Interval bisection stops if either the interval size drops below the error margin ε_d, or a feasible time d was found.

The decisions in lines 10 and 13 are based on two properties of D_A and D_B. Refer to Figure 6.5 and Equations (6.8),(6.9): 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 latest start time d leads to an infeasible system according to function D_A, then increasing the deadline will relax the problem and might yield a feasible system. On the other hand, if d leads to an infeasible system according to function D_B, then decreasing d will probably lead to a feasible system.

The algorithm returns the pair $(true, d)$, if a feasible value is found and $(false, -1)$ otherwise.

Note, that for task λ an *own* event stream $\left(a_\lambda^{(n)}\right)_{n\in\mathbb{N}}$ is required as input: as a consequence of the discussions on Cases A and B, the occurrence behaviour of the dependent task ρ and that of task λ must be coupled. But it is not necessary for both tasks to occur at the same rate: for example, it is allowed that task λ occurs with half the rate than task ρ. However, it is necessary that every job of task λ is created within a certain range of the creation time of an earlier job of task ρ, and this range must be bounded over the lifetime of the system.

Algorithm 6.3.1: Function `find_feasible`$\left(\Gamma, \rho, \left(a_\lambda^{(n)}\right)_{n\in\mathbb{N}}, c_\lambda, d_{\min}, d_{\max}\right)$ (see also in [LS2009])

Input: Task set Γ, special task ρ, event stream of low-power task $\left(a_\lambda^{(n)}\right)_{n\in\mathbb{N}}$, duration of low-power c_λ, bounds for latest start of task λ: $d_{\min}, d_{\max}$

Output: *true* and a deadline d, if and only if a feasible deadline d was found

1 **if** $d_{min} < d_{max}$ *and feasible_CASE_A* $\left(\Gamma, \rho, c_\lambda, d_{max}, \left(a_\lambda^{(n)}\right)_{n\in\mathbb{N}}\right)$
2 *and feasible_CASE_B* $\left(\Gamma, \rho, c_\lambda, d_{min}, \left(a_\lambda^{(n)}\right)_{n\in\mathbb{N}}\right)$
3 **then**
4 **while** $d_{max} - d_{min} > \varepsilon_d$ **do**
5 $d := d_{\min} - (d_{\max} - d_{\min})/2$;
6 **if** *feasible_CASE_A* $\left(\Gamma, \rho, c_\lambda, d, \left(a_\lambda^{(n)}\right)_{n\in\mathbb{N}}\right)$ **then**
7 **if** *feasible_CASE_B* $\left(\Gamma, \rho, c_\lambda, d, \left(a_\lambda^{(n)}\right)_{n\in\mathbb{N}}\right)$ **then**
`/* CASES A and B successful */`
8 **return** $(true, d)$
9 **else**
`/* CASE B failed, increase d */`
10 $d_{\max} := d$;
11 **endif**
12 **else**
`/* CASE A failed, decrease d */`
13 $d_{\min} := d$;
14 **endif**
15 **endw**
16 **else**
17 **return** $(false, -1)$
18 **endif**

6.3.2. Processor Shutdown in Between Two Jobs of the Same Task

To arrange the transition into low-power mode interleavingly to the jobs of a certain task of the task set is the central idea of task-dependent shutdown: the occurrence behaviour of the low-power mode is inherited from the occurrence behaviour of the task. According to the least available idle time in busy situations, cases A and B, the execution time c_λ is adjusted. Because the task to depend on determines the instantiation rate of the low-power interval, the choice of the task to depend on becomes an objective variable. Further, a feasible latest start time d_σ needs to be found and verified against cases A and B. The result of the optimisation will be a real-time feasible setup for processor shutdown.

Let the transition into low-power mode depend on a task, say ρ, then the (delayed) event function for task ρ computes the number of times the processor is shut down. Let c_λ be the

duration of the mode transition, then, according to Equation (6.1) the efficiency of the method is

$$\mathrm{Eff}(\Delta t) = \frac{m(\rho, d_\rho, \Delta t)}{\Delta t} \cdot (c_\lambda - t_{\mathrm{be}}), \tag{6.12}$$

or with $\mathrm{avg}(\rho)$ being the average number of jobs of task ρ per time unit

$$\mathrm{Eff} = \mathrm{avg}(\rho) \cdot (c_\lambda - t_{\mathrm{be}}). \tag{6.13}$$

For a task set Γ and a break-even time t_{be}, Algorithm 6.3.2 computes the parameters ρ, d_σ, and c_λ.

Algorithm 6.3.2: Determination of Parameters ρ,c_λ,d_σ for Shutdown in Between a Number of Task Instances (See also in [LS2009])

Input: Task set Γ, break-even time t_{be}
Output: Task ρ, to be followed by time c_λ in low-power mode, the latest d_σ time units past instantiation of ρ, Efficiency $\mathrm{Eff}_{\mathrm{best}}$

1 $\mathrm{Eff}_{\mathrm{best}} := 0$;
2 **for** $\tau \in \Gamma$ **do**
3 $\quad c_{\max} := \min\{\Delta t - d(\Delta t, \Gamma \setminus \{\tau\}) : \Delta t > 0\}$;
4 $\quad c_{\min} := t_{\mathrm{be}} + \mathrm{Eff}_{\mathrm{best}}/\mathrm{avg}(\tau)$;
5 $\quad a_\lambda^{(n)} := a_\tau^{(n)}, \quad n = 0, 1, 2, \ldots$;
6 $\quad$ **while** $c_{max} - c_{min} < \varepsilon_c$ **do**
7 $\quad\quad c = c_{\min} - (c_{\max} - c_{\min})/2$;
8 $\quad\quad d_{\min} := d_\tau$;
9 $\quad\quad d_{\max} := a_\tau^{(2)} + d_\tau - c_\tau - c$;
10 $\quad\quad (solution\_found, d_{\mathrm{solve}}) =$`find_feasible`$\left(\Gamma, \tau, \left(a_\lambda^{(n)}\right)_{n\in\mathbb{N}}, c, d_{\min}, d_{\max}\right)$;
11 $\quad\quad$ **if** *solution_found* **then**
12 $\quad\quad\quad \mathrm{Eff}_\tau := avg(\tau) \cdot (c - t_{\mathrm{be}})$;
13 $\quad\quad\quad$ **if** $\mathrm{Eff}_{best} < \mathrm{Eff}_\tau$ **then**
14 $\quad\quad\quad\quad (\rho, c_\lambda, d_\sigma) := (\tau, c, d_{\mathrm{solve}})$;
15 $\quad\quad\quad\quad \mathrm{Eff}_{\mathrm{best}} := \mathrm{Eff}_\tau$;
16 $\quad\quad\quad$ **endif**
17 $\quad\quad\quad c_{\min} := c$;
18 $\quad\quad$ **else**
19 $\quad\quad\quad c_{\max} := c$;
20 $\quad\quad$ **endif**
21 $\quad$ **endw**
22 **endfor**

For each task in the task set, the algorithm explores the design space by interval bisection over possible execution times c for the low-power task λ, and tries to determine a feasible deadline by interval bisection over a range of deadlines using function `find_feasible`, page 93. If

no deadline is found for c, then the execution time for low-power mode must be less than c for the actual parameter setting: c defines the new maximum $c_{\max}$. In the opposite case, c defines the new minimum $c_{\min}$. The search stops as soon as the interval size $c_{\max} - c_{\min}$ drops below the error margin ε_c.

The maximal duration to spend in low-power mode $c_{\max}$ is defined by the amount of available idle time in the busiest situation. The lower bound equals the break-even time. The processor may switch into low-power mode not before the special task has finished, thus its deadline defines the lower bound $d_{\min}$ for the latest start time of low-power task λ. Further, task λ has to be finished before the next job of the special task has to be finished: the latest start time $d_{\max}$ is the sum of $d_{\min}$ and the shortest time span between two jobs of the special task minus the sum of the execution times of the special task and task λ.

If a solution is found, then its efficiency is validated against previous solutions to take the best.

Complexity of the Solution: Since the intervals of the solution space are halved by each iteration, the algorithm will converge. Now, let ρ be a task of the task set Γ. The exploration of task dependent shutdown for task ρ has the complexity:

$$O(\text{Algorithm 6.3.2}, \rho) = \log_2\left(\frac{c_{\max}(\rho) - t_{\text{be}}}{\varepsilon_c}\right) \cdot \log_2\left(\frac{d_{\max}(\rho) - d_{\min}(\rho)}{\varepsilon_d}\right) \cdot const \cdot O(\text{feas. test}). \quad (6.14)$$

Let n be the number of tasks in task set Γ. Define $c_M := \max\{d_\tau - c_\tau : \tau \in \Gamma\}$, $c_m := \min\{c_\tau : \tau \in \Gamma\}$, and $a_M := \max\{a_2^\tau \in \Theta_\tau : \tau \in \Gamma\}$. Then, it is $c_{\max}(\rho) \leq c_M$, $d_{\min}(\rho) \geq c_m$, and $d_{\max}(\rho) \leq a_M$. Therefore, the complexity for the algorithm to explore task-dependent shutdown for all tasks in the task set is:

$$O(\text{Algorithm 6.3.2}) \leq n \cdot \log_2\left(\frac{c_M}{\varepsilon_c}\right) \cdot \log_2\left(\frac{a_M - c_m}{\varepsilon_d}\right) \cdot const \cdot O(\text{feas. test}). \quad (6.15)$$

Comparison to Slowdown of a Single Task: The insertion of a new task that has the same occurrence behaviour than a present task affects the processing demand in the same way as if the execution time of the present task is extended. Local slowdown applied to a single task increases the task's execution time as well, but local slowdown has the task's deadline as a limiting constraint, which the task dependent shutdown has not.

The following theorem states that it is either the case that task-dependent shutdown exploits more available idle time than global slowdown, or that there is another task with a shorter deadline limiting available idle time.

Theorem 6.3.2. *Let γ be a real-time feasible global slowdown factor. Let c be the worst-case execution time of task ρ, and let c_λ be the minimal allowed time to spent in low power mode in between any two jobs of ρ. Define $\delta := c \cdot \gamma - c$, then it is either $\delta > d_\tau$ for some task τ, or $\delta \leq c_\lambda$.*

Proof. Let $d := \min\{d_\tau : \tau \in \Gamma\}$: this is an upper bound on time units that are free of preemption. It follows $c_\lambda < d$.

Consider now a job of ρ is running and has been processing for c_ρ time units. If $\delta \leq d$, then the job is processed continuously until it finishes; every incoming job will have its deadline later than δ time units. Therefore c_λ at least equals δ in this case. □

We conclude that if Algorithm 6.3.2 is applied to a task with lowest deadline, the exploited idle time will be more than the idle time which is exploited if slowdown is applied to the same task. Additionally, processor shutdown experiences no interruptions due to highest priority. However, as we have seen in the last chapter, remaining or exploited idle time is not an indicator for power optimality: whether to use slowdown or shutdown depends on the processor's physical values.

6.3.3. Shutdown in Between Many Instantiations of the Same Task

Following the same argumentation as in the subsection before, it can also be thought of waiting for a number of instances, say k, then to apply shutdown, and then again wait for the same number of instances before the next shutdown. Since the number of shutdowns is dependent on the task's occurrence rate, this modification may reduce the number of switches and may increase the efficiency.

Figure 6.7, page 97, shows a sample timeline for the modified method. The time points $t_0, \ldots, t_3$ mark the instantiation times of task ρ. The jobs marked with "λ" depict the time when the processor is put into low-power mode, this happens every second instance of task ρ, thus the occurrence rate of the low-power interval is halved; its duration is increased (compared to Figure 6.4, page 89).

The latest start time d_σ is bounded by the minimal distance between two consecutive jobs of task ρ: this is identical to the previous subsection.

The occurrence behaviour of low-power task λ is coupled to task ρ, but now the occurrence rate is different. Say, a job of task λ is to be executed after k jobs of task ρ, then the occurrence behaviour for task λ is defined by

$$a_\lambda^{(n+1)} := a_\rho^{(n\cdot k+1)}, \quad n = 0,1,2,\ldots. \tag{6.16}$$

The efficiency of this modification is

$$\mathrm{Eff}_\tau := avg(\rho) \cdot k \cdot (c - t_{\mathrm{be}}). \tag{6.17}$$

Algorithm 6.3.3, page 98, shows the modification of Algorithm 6.3.2. There is an extra loop for each task (line 4), where the algorithm cycles through possible factorisations, $k = 1, \ldots$; the cycling stops when no further improvement can be achieved, and this happens when $c_{\min} > c_{\max}$ (line 6) because the lower bound $c_{\min}$ is defined by the efficiency of previous solutions (line 5).

The modified occurrence behaviour is changed to inherit every k-th element only (line 9).

The efficiency computation according to Equation (6.13) is replaced in line 16 by the modified variant, Equation (6.17).

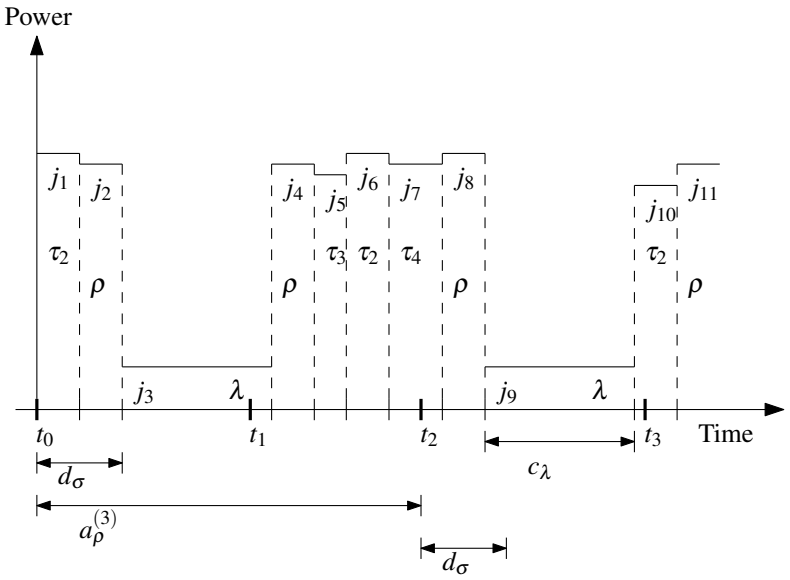


Figure 6.7.: Task-Dependent Shutdown, Every 2nd Instantiation

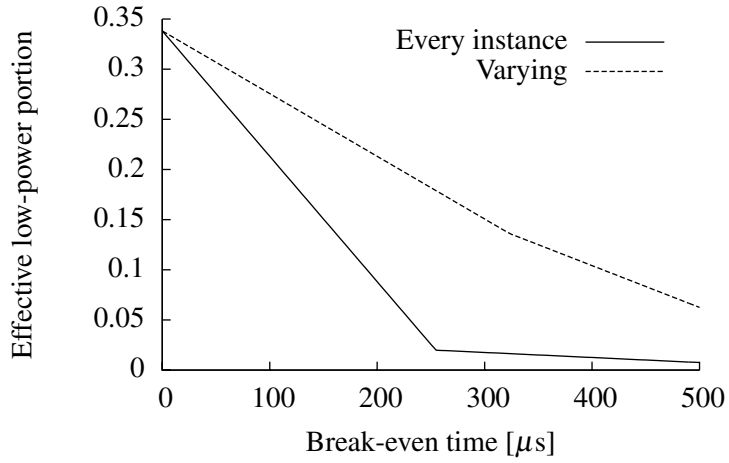


Figure 6.8.: Evaluating Task-Dependent Shutdown: Every Instance vs. Varying Number of Instances

Comparison to Shutdown in Between Every Two Instantiations: The presented modification will be useful for task sets if the occurrences of some tasks are clustered, as the optimal occurrence rate for processor shutdown may fall in between two clusters, and then the modification can compute a suitable occurrence rate.

The task set given in [TC1994] contains 17 periodic tasks, some with jitter. One task occurs with a rate of $800\,\mu s$, then there is a gap to the next shortest period of $20.000\,\mu s$ with seven tasks falling in the range up to $59.000\,\mu s$, followed by seven tasks with periods of $100.000\,\mu s$ and $200.000\,\mu s$, and two tasks having periods of $1.000.000\,\mu s$.

Algorithm 6.3.3: Determination of Parameters ρ,c_λ,d_σ,k_l for Shutdown in Between a Number of Task Instances

Input: EDF scheduled Task set Γ, break-even time t_{be}

Output: Task ρ, instances to wait k_l, to be followed by time c_λ in low-power mode, the latest d_σ time units past instantiation of ρ, Efficiency Eff_{best}

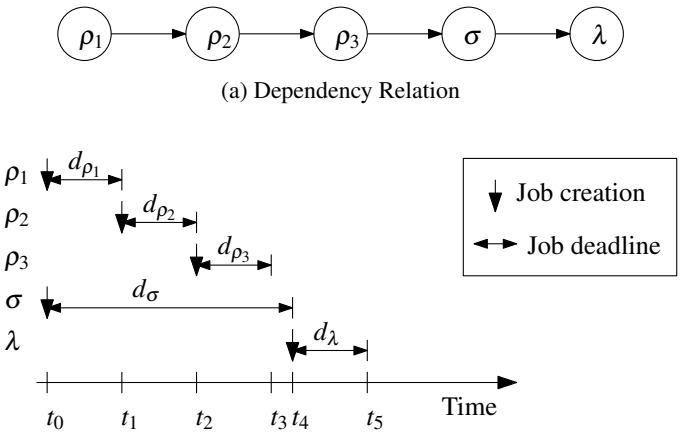
$\text{Eff}_{\text{best}} := 0$;
for $\tau \in \Gamma$ **do**
  $c_{\max} := \min\{\Delta t - d(\Delta t, \Gamma \setminus \{\tau\}) : \Delta t > 0\}$;
  for $k := 1,2,3,4,\ldots$ **do**
    $c_{\min} := t_{\text{be}} + k \cdot \text{Eff}_{\text{best}}/\text{avg}(\tau)$;
    if $c_{min} > c_{max}$ **then**
      exit for loop;
    endif
    $a_\lambda^{(n+1)} = a_\tau^{(n\cdot k+1)}, \quad n = 0,1,2,\ldots$;
    while $c_{max} - c_{min} < \varepsilon_c$ **do**
      $c = c_{\min} - (c_{\max} - c_{\min})/2$;
      $d_{\min} := d_\tau$;
      $d_{\max} := a_\tau^{(2)} + d_\tau - c_\tau - c$;
      $(solution\_found, d_{\text{solve}}) = \texttt{find\_feasible}\left(\Gamma, \tau, \left(a_\lambda^{(n)}\right)_{n\in\mathbb{N}}, c, d_{\min}, d_{\max}\right)$;
      if *solution_found* **then**
        $\text{Eff}_\tau := \frac{avg(\tau)}{k} \cdot (c - t_{\text{be}})$;
        if $\text{Eff}_{best} < \text{Eff}_\tau$ **then**
          $(\rho, c_\lambda, d_\sigma, k_l) := (\tau, c, d_{\text{solve}}, k)$;
          $\text{Eff}_{\text{best}} := \text{Eff}_\tau$;
        endif
        $c_{\min} := c$;
      else
        $c_{\max} := c$;
      endif
    endw
  endfor
endfor

Figure 6.8 shows how Algorithms 6.3.2, and 6.3.3 perform along a range of break-even times. The optimal occurrence rate for processor shutdown turned out to be around $2.000\,\mu s$. Although none of both methods can match this rate, the modification presented here can compensate occurrence rate for higher break-even times.

6.3.4. Shutdown After Execution of Tasks Sharing Occurrence Behaviour

The idea to avoid the intersection of job execution can also be applied to analyse task-dependent shutdown for cases of tasks that share occurrence behaviour, or tasks that are instantiated consecutively *and* share occurrence behaviour. These two possibilities are discussed in the following.

Consecutive Instantiation: Consider Figure 6.9, where tasks ρ_1, ρ_2, ρ_3 have precedence constraints and their jobs are to be executed consecutively. After the last job has finished, the processor may switch into low-power mode: that is no earlier than $d_{\min} := d_{\rho_1} + d_{\rho_2} + d_{\rho_3}$. Let $\left(a_\rho^{(n)}\right)_{n\in\mathbb{N}}$ be the event stream denoting the task graph's occurrence behaviour. The low-power task has to be finished before task ρ_1 occurs next. In analogy to Equation (6.7), the processor has to wake up from low-power mode no later than $d_{\max} := a_\rho^{(2)} + d_{\rho_1} - c_{\rho_1}$.



(a) Dependency Relation

(b) Deadline Relation

Figure 6.9.: Precedence Constraints, Consecutive Instantiation

Although not to be further discussed, but following the argumentation: the low-power task may be placed within the task graph which leads to a similar result.

Parallel Instantiation: Consider now Figure 6.10, page 100, where tasks ρ_1, ρ_2, ρ_3 occur synchronously. First, the low-power job is to be executed after the last task has eventually finished, which implies $d_{\min} := \max\{d_{\rho_1}, d_{\rho_2}, d_{\rho_3}\}$. Second, let $\left(a_\rho^{(n)}\right)_{n\in\mathbb{N}}$ denote the occurrence behaviour of the task graph, then the low-power task has to be finished no later than $d_{\max} := a_\rho^{(2)} + \min\{d_{\rho_1} - c_{\rho_1}, d_{\rho_2} - c_{\rho_2}, d_{\rho_3} - c_{\rho_3}, d_{\min} - c_{\rho_1} - c_{\rho_2} - c_{\rho_3}\}$, which is the latest time to begin execution of the next jobs of tasks ρ_1, ρ_2, ρ_3.

Consecutive and parallel instantiation do not modify the method of avoiding the intersection of execution. Solely by adjustment of the bounds $d_{\min}$ and $d_{\max}$ these two problems are solved.

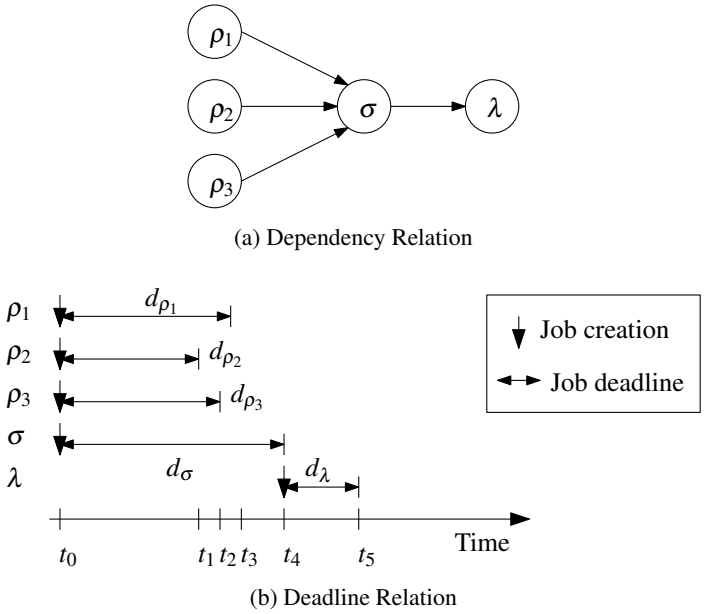


Figure 6.10.: Precedence Constraints, Parallel Instantiation

6.4. Summary

This chapter characterised energy minimisation by processor shutdown. Periodic shutdown and task-dependent shutdown were presented. Both methods optimise their parameters by invoking a real-time feasibility test, and integrate processor shutdown as a software task with execution time. The execution stands for the duration of mode switch and stay in low-power mode. Interruption of the low-power mode is prevented by setting deadline equal to execution time.

The optimisation algorithm for periodic shutdown has to find optimal period and duration, but although objective and real-time test are convex, the optimisation problem showed several local extrema because the set of test points changes between repeated invocation of the real-time test. The problem was downsized by computing solutions along a range of break-even times, because solutions show to be valid for neighbouring break-even times as well.

Task-dependent shutdown leads to a new theory for a relaxed real-time feasibility problem, where two tasks are not to be instantiated at the same time.

The algorithm finds optimal duration of low-power mode, a selection of a task to depend on, and a factorisation of the task's occurrence behaviour. The latter was shown necessary to compensate gaps in available occurrence rates.

Task-dependent shutdown has to be implemented in a task itself. This gives the advantage of online awareness: if a job at run-time finds that it will need less processing time than predicted for the worst-case, then it may increase the duration of the low-power task by an amount yet to be determined.

In all cases of processor shutdown, the tasks' execution times and frequencies are retained as well, but a low-power task λ is added to the task set. The low-power task comes with own

deadline, own power consumption, and own occurrence behaviour. This leads to new feasible configurations of the task set.

7. Modelling the Battery as Energy Source

From the previous Chapters 5, and 6, we obtained several real-time feasible configurations for the software. The tasks' power consumptions and execution times were modified by the slow-down methods; a new low-power task was introduced by both shutdown methods.

Such a configuration consists of the task set and the processor's power consumptions when idle or in low-power mode. Each task of the task set comes with own deadline, execution time, power consumption (though through frequency), and event stream.

These different configurations need to be judged in terms of the operating life they yield. A battery model's aim is to take these configurations as input and to provide an operating life, or a minimal operation time as output.

In the next section, an overview over factors affecting operating life is given. In the second section, the time granularity for a suitable battery model is discussed. The chapter closes with a presentation of a feasible model for the prediction of operating life with task configurations.

7.1. Factors Influencing the Operating Life of a Battery

Already in the Preliminaries, basic mechanical cell constructions were shown, as well as some cell reactions for common, commercial cells. Both are factors that depend on battery design and influence operating life, and both are discussed in the next subsection. In the second subsection, factors regarding the usage of a battery are discussed. A special subsection addresses pulsed discharge and its influence on the operating life.

A summarising subsection will present a selection of factors.

7.1.1. Battery Design Dependent Factors

Geometry, chemistry, and battery physics are fixed by battery design, which itself is a field of its own. All of these factors influence the amount of charge that can be drawn from a battery.

Geometry The volume of the battery's container determines the amount of material to put into. The surface area of the electrodes determines the speed of electron exchange from reacting material to current collector, and the number of reactions taking place in parallel.

The shapes of container, electrodes, current collectors, as well as arrangements of these mechanical components affect the solution gradients which are evolving, and affect size and shape of electric fields within the cell.

For precision, battery geometry may cause a developer of battery models to consider three spatial dimensions, [HL1999].

Nevertheless, shape and surface of electrodes and current collectors is knowledge of the battery manufacturer, and may only be obtained by dismantling the device.

Chemistry The used materials define a battery's chemistry. Anode and cathode material define the reduction-oxidation reaction that takes place. The reaction determines the cell voltage. The reaction rate limits electric current for charge and discharge.

The amount of anode and cathode material determines the cell's theoretical capacity; together with the battery's geometry the battery's theoretic volumetric and gravimetric energy densities are determined.

The electrolyte conducts and stores ions for the reduction-oxidation reaction. Ion conductivity may limit the cell's reaction rate in case the battery designer has not taken decisions to prevent this.

The battery chemistry is an essential factor affecting operating life.

Physics The used materials also define physics of the battery in terms of heat conductance. The cell's reaction may produce and consume thermal energy. Since reaction rate as well as the electrolyte's ion conductance are affected by the surrounding temperature, heat conductance becomes important.

Depending on material used, the cell can be considered iso-thermal: thus a model that characterises battery temperature does not appear as a distribution through the cell's components but as a single scalar value.

Maximum Ratings Also the battery's maximum ratings are design dependent: these define the maximal current for charge and discharge, and the temperature range for storage as well as usage.

The current for charge and discharge is limited above by the cell's reaction rate in combination with the cell's electrode surface.

The temperature ranges are limited below by the cell components to provide a useful reaction rate, and are limited above by the probability of cell components to degenerate.

Maximum ratings are common specifications that are given in data sheets for batteries provided by manufacturers.

7.1.2. Usage Dependent Factors

How a battery is handled also influences its performance. "Improper treatment may cause damage to battery or user" – this sentence serves as a warning sign on almost every commercial battery. The battery is treated properly if it is operated within the maximum ratings. Storage of the battery, ambient temperatures, charge and discharge currents are specified. Especially, charge and discharge currents affect the battery's performance for the next cycle.

Here, battery charge is left out because the model that is developed covers solely battery discharge: the battery is considered properly and fully charged before discharge takes place.

Cycling If a fully charged nickel-based cell is discharged by a shallow current and then recharged, or is recharged before discharge reached cut-off potential, then the cell will have a gradual reduction in power and capacity for the next discharge cycle. Its terminal voltage will be significantly lower than usual. This effect is called "memory effect" or "voltage depression", refer to

[LR2002], pp. 27.23, **28.18**, 29.19. The effect is reversible by one discharge down to cut-off potential followed by a complete recharge.

Another effect regarding cycling is the capacity fade induced by an increasing number of charge/discharge cycles. The energy a battery can deliver decreases by each time it is discharged and charged again. The values can be obtained from the manufacturer's data sheets, and the decrease can be in the order of one forth of the rated capacity of a new battery, e.g. refer to [SAN2003]. Research is going on to reduce and analyse this, see for example [SHW2005], and [ZW2008]. The effect also varies with the battery chemistry: for example, portable sealed nickel-cadmium batteries retain 80% of their initial rated capacity after 1000 cycles (See [LR2002], page 28.17.).

Storage When stored in a discharged state for a long time (order of months), lead-acid batteries may suffer from irreversible polarisation of the electrodes (sulfation), see [LR2002], p. 23.2). This will damage the battery, but it is limited to discharged storage.

Battery self-discharge, also known as capacity or charge retention, is a process that unloads a battery without a connection to any circuit. The discharged amount depends on ambient temperature, duration, and the battery chemistry. Whereas for primary batteries self-discharge is almost negligible (order of years, (See [LR2002], page 6.10.)), it comes to be an issue for secondary batteries (order of month, ibid). Commercially available valve-regulated lead-acid batteries may loose 9% of the initial charge after three months at 25°C, 18% after 6 months, and 36% after 12 months, see [Pan2005]. Although these data are linear, in the general case they are not, refer to [LR2002], pages 24.15 (lead-acid), 28.16 (nickel-cadmium), 30.25 (nickel-metal hydride), 35.51 ($LiCoO_2$).

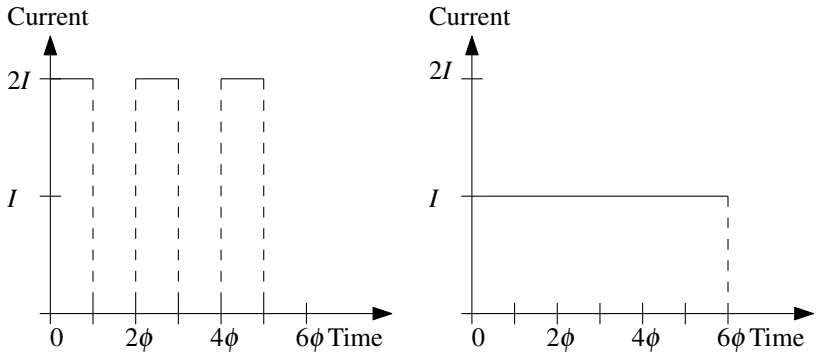
Ambient Temperature Temperature acts on the speed of the reduction-oxidation reaction. The higher the temperature, the likelier chemicals react within the cell: low temperature causes low reaction speed for the redox reaction, that is the current that can be drawn is limited; and high temperature increases the probability for reaction, which increases the current that can be drawn, but also increases self-discharge, refer to [GW2000], and [LR2002], page 3.10.

Self-discharge depends exponentially on the temperature, both for primary, [LR2002] p. 7.20, and secondary batteries, ibid p. 22.19. Internal resistance (in terms of the speed of the redox reaction) depends rather quadratically on the temperature, but the dependence changes with the discharge current, refer to [LR2002], page 29.13. Together, the change in internal resistance and the change in self-discharge form a capacity's dependence of temperature in the shape of an inverted parabola, refer to [LR2002], pages 7.18, 7.19 for primary batteries, and page 22.18 for secondary batteries.

Discharge Current According to the Preliminaries and citing the Peukert-formula (Equation (2.16), page 43), the discharge rate influences the capacity that is obtained from the battery. If the discharge current is doubled, then the time before cut-off potential is reached is halved; this relation is worse for a Peukert-coefficient greater than 1.

Nevertheless, the formula is given for constant discharge currents. Two non-constant discharge currents which have the same mean value may lead to different capacities.

Further, in reference [FDN1994a], the authors report a relaxation phenomenon for lithium-based electrochemical cells. This work is interpreted by several authors cited in the chapter *State of the Art*: they propose for lithium-based batteries to have a relaxation effect, that is discharge currents with periods of zero discharge would lead to an increased capacity compared to constant discharge with the same mean value. Compare Figure 7.1, the three pulses of height $2I$ have in sum the same area as the block of height I. The stated assumption is that the profile from the left draws energy out of the battery more efficiently than the profile from the right.



(a) Three pulses for 6ϕ time units (b) Constant over 6ϕ time units

Figure 7.1.: Pulsed Discharge Versus Constant Discharge

This assumption has to be validated by measurements. With the battery measurement device described in [Wei2008], a SONY li-ion cell[1] was discharged by several profiles, pulsed and constant, each of different amplitude. The cell parameters are:

a) Cylindrical: diameter of 18 mm and a length of 65.0 mm.

b) Nominal cell voltage of 3.7 V.

c) Rated capacity of 1300 mAh at the one hour discharge rate (I=1.3 A).

d) Weight of 49 g.

e) Recommended cut-off voltage of 2.5 V.

Table 7.1, page 107, shows the results of the validation (rounded values). A *type* of "$\phi = 10$ ms" is a discharge profile similar to Figure 7.1(a): pulsed discharge, the load is hold for ϕ time units, and then followed by a pause (zero load) for the same number of time units. The current I is the measured value of the current which the device has drawn from the battery. It denotes t the time that discharge lasted, and C the resulting capacity.

In between the discharges, the battery was charged using the standard charge method for this type of battery: (1) constant-current constant-voltage with $I = 1300$ mA, $V = 4.2$ V; (2) charge

[1] Exact chemistry: LiC_6 - $LiMn_2O_4$, same discharge reaction as was illustrated in the Preliminaries, Subsection 2.6.1.

terminated when the current dropped below 65 mA; and (3) the battery was neither charged nor discharged for 30 minutes to let the temperature stabilise.

Measurements were taken on 2009-06-10, and repeated a few days later on 2009-06-13, when constant discharges 975 mA and 650 mA were added. In both cases, the ambient temperature was about 17°C.

Table 7.1.: Pulsed Versus Constant Discharge, Measured with a *SONY 18650 Li-Manganese Cell, 3.7 V, 1300 mAh, 2.5 V Cut-Off*

No.	Type	I [A]	t [s]	C [As]
1	const.	1.298	3662	4752
2	const.	1.297	3658	4746
3	$\phi = 10\,\text{ms}$	1.298	7376	4788
4	const.	2.595	1811	4700
5	$\phi = 10\,\text{ms}$	2.595	3655	4743
6	const.	1.946	2419	4708
7	$\phi = 10\,\text{ms}$	1.947	4883	4754
8	const.	0.865	5496	4752
9	$\phi = 10\,\text{ms}$	0.865	11061	4786

(a) Start Date: 2009-06-10, End Date: 06-11

No.	Type	I [A]	C [As]	t [s]
1	const.	1.297	3675	4768
2	const.	1.297	3670	4761
3	$\phi = 10\,\text{ms}$	1.298	7395	4800
4	const.	2.595	1813	4705
5	$\phi = 10\,\text{ms}$	2.595	3658	4747
6	const.	1.946	2421	4712
7	$\phi = 10\,\text{ms}$	1.947	4890	4761
8	const.	0.865	5506	4761
9	$\phi = 10\,\text{ms}$	0.865	11094	4800
10	const.	0.972	4891	4756
11	const.	0.649	7368	4779

(b) Start Date: 2009-06-13, End Date: 06-15

The measurement data for the first date clearly show, that there is a small offset of about 40 As, or 11 mAh, between constant discharge to pulsed discharge of the same current. The data also show, that the elongation of discharge time by pulsed discharge equals double the discharge time of constant discharge. Concerning the assumption that pulsed discharge would perform better: a pulsed discharge of 2.6 A yields less or equal the capacity as the constant discharge of 1.3 A.

The measurement data of the second date show a reduced offset between constant and pulsed discharge. The data point to the same direction as before: the assumption proves to be false.

Additionally, the added discharge rates 975 mA, and 650 mA confirm the opposite of the assumption for the corresponding pulsed discharges with 1950 mA and 1300 mA.

The obtained results clearly contradict the assumption pulsed discharge performs better than constant discharge.

Table 7.2.: Fitting the Peukert-Coefficient for the SONY Lithium-Manganese Cell

No.	I [A]	t [s]	$I^{1.011} \cdot t$	$I^{1.012} \cdot t$	$I^{1.013} \cdot t$
1 a), 2 a), 6 a), 1 b), 2 b), 6 b)	1.298	3666	4772	4773	4775
4 a), 4 b)	2.595	1812	4752	4756	4761
8 a), 8 b)	0.865	5501	4751	4750	4749
10 b)	0.972	4891	4767	4767	4767
11 b)	0.649	7368	4759	4757	4755
Mean value			4760	4761	4761
Standard deviation			9.43	9.35	9.91

Computing the Peukert-Coefficient Concerning the Peukert-formula, we can compute the Peukert-coefficient out of the data given in Tables 7.1 a) and b). Let us take the average of the measured times for each current and compute the formula $I^{pc} \cdot t$ for each pair of current and time. This yields a solution vector for each coefficient pc. All entries in a vector should be the same: we minimise the standard deviation by adjusting the coefficient pc, then we obtain a coefficient which results in minimal error, this will be the Peukert-coefficient for the cell. Table 7.2 shows the solution vectors for coefficients $pc = 1.011$, $pc = 1.012$, and $pc = 1.013$, and the table shows the standard deviation within each vector. A coefficient of $pc = 1.012$ yields the best result.

The battery then has the normalised capacity $C_{\text{norm}} = 4761$ As.

Different Load at Cut-Off Potential Data of Table 7.1 show small offsets if we compare the capacity that is obtained by a constant discharge to the capacity that pulsed discharge of same amplitude yields. These offsets may be caused by the measurement device because it measures cell voltage with connected load. Since the load differs, and because the cut-off potential does not, the states-of-charge of the battery when discharge ends are not the same.

It is now to see, that constant discharge followed by constant discharge with half the amplitude leads to the same capacity as pulsed discharge with a share of 50%.

Table 7.3, page 109, shows the data of the repeated measurements (rounded values). The currents were of amplitude $I = 2.6$ A, $I = 1.3$ A, and $I = 0.65$ A, and the ambient temperature was about 17.1°C. Before each measurement, the battery was charged with the same settings as for previous experiments. For each measurement, the amount of capacity C and discharge time t are shown for each block and in total (bold face). The sequence for each measurement is to perform first block a) and then immediately block b): discharge to cut-off potential with the first profile, and without charge, discharge to cut-off with the second profile.

First, the data shows: a current of 1.3 A that follows a constant discharge of 2.6 A increases the totally obtained capacity by 62 As; for the pair (0.65 A, 1.3 A), this is 40 As; and for the pair

Table 7.3.: Evaluating the Difference between Pulsed and Constant Discharge, Each Followed by Constant Discharge with Half the Amplitude, *SONY 18650 Li-Ion Cell, 3.7 V, 1300 mAh, 2.5 V Cut-Off*, Start Date of Measurements: 2009-06-20, End Date: 06-21

No.	Block	Type	I [A]	t [s]	C [As]
1	a)	const.	1.298	3640	4723
	b)	const.	0.649	58	37
			Total	**3697**	**4760**
2	a)	const.	2.595	1803	4678
	b)	const.	1.298	48	62
			Total	**1850**	**4739**
3	a)	$\phi = 10\,\mathrm{ms}$	2.596	3636	4719
	b)	const.	1.298	4	5
			Total	**3640**	**4724**
4	a)	$\phi = 50\,\mu\mathrm{s}$	2.596	3630	4713
	b)	const.	1.298	4	5
			Total	**3634**	**4718**
5	a)	const.	1.298	3625	4705
	b)	const.	0.649	61	40
			Total	**3687**	**4744**
6	a)	$\phi = 10\,\mathrm{ms}$	1.298	7318	4750
	b)	const.	0.649	3	2
			Total	**7322**	**4753**
7	a)	$\phi = 50\,\mu\mathrm{s}$	1.298	7317	4749
	b)	const.	0.649	3	2
			Total	**7321**	**4751**
8	a)	const.	0.649	7317	4746
	b)	const.	0.324	54	17
			Total	**7370**	**4763**
9	a)	$\phi = 10\,\mathrm{ms}$	0.649	14719	4777
	b)	const.	0.324	4	1
			Total	**14722**	**4779**
10	a)	$\phi = 50\,\mu\mathrm{s}$	0.65	14714	4780
	b)	const.	0.324	4	1
			Total	**14718**	**4781**

(0.325 A, 0.65 A), this is 17 As. Second: a constant current of 1.3 A that follows a pulsed discharge of 2.6 A increases the totally obtained capacity by 5 As; for the pair (0.65 A, 1.3 A), this is 2 As; and for the pair (0.325 A, 0.65 A), this is only 1 As. Third, considering measurements 1 and 2, constant discharge yielded more capacity than pulsed discharge; considering measurements 5 and 8, it is the other way around. In summary, the measurement data shows diminishing

differences between constant discharge and pulsed discharge of same amplitude.

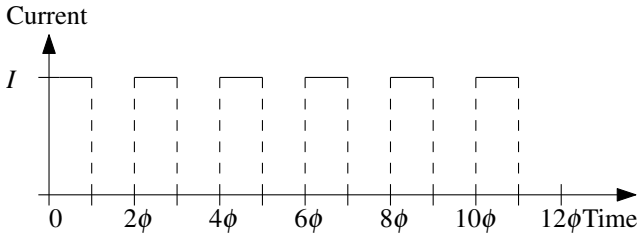
The data shows that unifying the termination condition for constant and pulsed discharge compensates for the differences denoted by earlier measurements.

Note that once more, it is falsified for pulses in the range of 10 ms to 50 μs that pulsed discharge performs better in terms of capacity than constant discharge of half the amplitude.

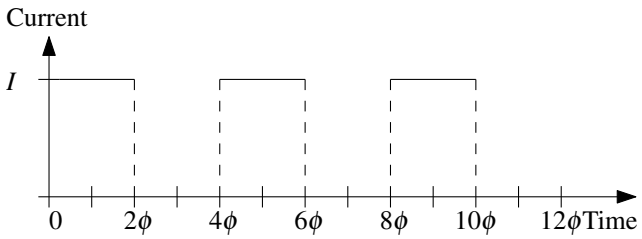
7.1.3. Influence of Pulse Duration Concerning Pulsed Discharge

In the previous subsection, the assumption pulsed discharge with doubled amplitude performs better in terms of capacity than constant discharge was falsified for pulses shorter than 10 ms. But keeping the amplitude constant, a variation of pulse durations may yield different capacities.

In Figure 7.2, two current profiles are shown, the left has half the number of pulses than the right, but has a doubled width. It is the objective of this subsection to validate if variation of pulse durations makes any difference for the resulting capacity.



(a) Six Pulses per 12t Time Units



(b) Three Pulses per 12t Time Units

Figure 7.2.: Variation of Pulse Width

Indicated by the data of Table 7.3, with negligible differences between pulsed discharge of periods 10 ms and 50 μs, the variation of the pulse durations may not lead to improved discharge performance. To validate this assumption, more detailed measurements were taken, with the same battery as in the previous subsection, the same conditions concerning charge method and cut-off voltage, and with an ambient temperature of about 16.4°C.

Table 7.4, page 111, shows the measurement data (rounded). The pulse durations were chosen among 10 ms, 1 ms, 500 μs, and 100 μs. Several pulse durations were repeated for the same current to test for influences of previous discharges. Before each discharge, the battery was charged using the standard charge as mentioned above. This was also the case for the first measurement, but here the cell had about 95% of its charge – thus was not empty – before the measurement, therefore the data in the first row may not be representative.

Table 7.4.: Equidistant Pulses and the Capacity Obtained, for a 3.7 V *SONY 18650 Li-Ion Cell* with 1300 mAh Rated Capacity (1 Hour Rate), $V_{cut\text{-}off} = 2.5V$, Start date of Measurements: 2009-06-06, End date: 06-08

No.	ϕ [ms]	I [A]	t [s]	C [As]
1	10	2.596	3700	4802
2	1	2.596	3674	4768
3	0.1	2.596	3669	4762
4	10	2.595	3669	4761
5	0.5	2.597	3664	4757
6	0.1	2.596	3660	4750
7	10	1.298	7377	4788
8	1	1.298	7378	4789
9	0.1	1.298	7378	4789
10	10	1.298	7381	4791
11	0.5	1.298	7380	4791
12	0.1	1.298	7377	4788
13	10	1.947	4897	4768
14	1	1.947	4893	4764
15	0.1	1.948	4890	4762
16	10	1.947	4893	4763
17	0.5	1.948	4890	4762
18	0.1	1.946	4888	4756

The table depicts three groups of capacities, partitioned by the corresponding discharge current. If we exclude the first row, each of the groups encircles a close spectrum of capacities and discharge times, with the first group intersecting with the last in terms of capacity. Again, if we exclude the first row, all capacity data is within a range of 41 As, with the first group having a range of 18 As, the second a range of 3 As, and the third having a range of 12 As.

The data shows no effect for variations of pulse durations: although a 10 ms pulse is 100 times longer than the 0.1 ms pulse, the error is within 18 As, and if we exclude measurement no. 1, this error could also be explained by a variation of the ambient temperature through the complete measurement. Note, that one charge takes about 1 hour 45 minutes, and a pulsed discharge of 2.6 A (share=50%) takes about an hour, that makes 5·1:45 h+4·1 h=12:45 h, or half a day.

Unfortunately, the measurement device only measures the cell temperature on the battery surface – the outside. Since every discharge is prepended by 30 minutes where the battery rests, the battery is likely to have cooled down after charge to ambient temperature. Therefore, the ambient temperature may be deduced from the temperature which the battery has at the start of a measurement.

Table 7.5, page 112 depicts temperature data measured at the battery's surface. Minimal, average, and maximal temperature are given, as well as the standard deviation. Although the data can again be divided into three blocks, according to the current drawn, within the blocks, variations are very small and do not point into any direction. A more detailed look at the mea-

Table 7.5.: Battery Surface Temperatures for Measurements of Table 7.4, in °C

No.	1	2	3	4	5	6	7	8	9
min	16.37	16.00	16.56	16.62	16.56	16.37	16.31	16.31	16.31
avg	18.14	18.19	18.22	18.33	18.19	18.00	16.87	16.89	16.95
max	20.25	20.31	20.37	20.43	20.31	20.12	17.93	18.00	18.12
std	0.67	0.66	0.65	0.65	0.63	0.63	0.28	0.3	0.3

(a) Measurements 1 - 9

No.	10	11	12	13	14	15	16	17	18
min	16.50	16.56	16.37	16.43	16.43	16.37	16.50	16.56	16.62
avg	17.14	17.13	17.02	17.6	17.5	17.52	17.68	17.68	17.73
max	18.25	18.25	18.12	19.31	19.12	19.25	19.18	19.31	19.37
std	0.3	0.29	0.3	0.46	0.45	0.48	0.46	0.46	0.46

(b) Measurements 10 - 18

surement data as well as recordings of ambient temperature are needed to either reject or accept a correlation between temperature and the variations in capacity.

The measurements were repeated from 2009-06-15 through 2009-06-20. Additionally, measurements were made for the currents 975 mA and 650 mA. The results (rounded) are shown in Table 7.6, page 113. The temperatures at each measurement start were in average 17°C. Now, before the battery was charged for the first measurement, the battery was empty, thus the difference in capacity between the first two measurements is reduced to 3 As – the difference was 34 As. The data confirm the result of previous measurements.

Thus, the conclusion is that variation of pulse durations does not show any difference for the capacity that is obtained.

7.1.4. Selection of Influences

From the list of design dependent factors, only those may come into consideration, which can be obtained without dismantling a battery. On the one hand, detailed geometry data concerning electrode surface can not serve as model parameter, and the same for reaction rates. On the other hand, battery chemistry and maximum ratings are published by the manufacturer, and this information is to be incorporated. Maximum ratings must be obeyed to avoid serious harm.

From all the mentioned influences concerning battery usage, only a few will come into consideration for a battery model because they are either insignificant or unimportant for a model that is to cover only one discharge.

First, it is unnecessary to include the memory effect into a model, but a system's designer should be aware of it and is advised to take care that shallow discharges for nickel-based cells are avoided.

Second, sulfation of lead-acid batteries becomes insignificant because it concerns only discharged batteries stored for months without recharge.

Table 7.6.: Repetition of Measurements of Table 7.4, Start Date of Measurements: 2009-06-15, End Date: 06-20, Temperature Around 17°C

No.	ϕ [ms]	I [A]	t [s]	C [As]
1	10	2.595	3665	4756
2	1	2.596	3662	4753
3	0.1	2.596	3658	4748
4	10	2.596	3656	4745
5	0.5	2.596	3651	4739
6	0.1	2.596	3646	4732
7	10	1.298	7353	4773
8	1	1.298	7353	4773
9	0.1	1.298	7353	4773
10	10	1.298	7354	4773
11	0.5	1.298	7353	4773
12	0.1	1.298	7345	4767
13	10	1.947	4869	4741
14	1	1.947	4866	4738
15	0.1	1.947	4865	4737
16	10	1.947	4868	4739
17	0.5	1.947	4867	4739
18	0.1	1.947	4866	4738
19	10	0.973	9808	4772
20	1	0.973	9811	4773
21	0.1	0.973	9811	4774
22	10	0.973	9814	4775
23	0.5	0.973	9814	4775
24	0.1	0.973	9812	4774
25	10	0.649	14756	4789
26	1	0.649	14754	4789
27	0.1	0.649	14753	4791
28	10	0.649	14753	4789
29	0.5	0.649	14746	4787
30	0.1	0.649	14743	4787

Third, self-discharge is neglected, though it mainly concerns secondary batteries: if a battery is supposed to provide energy over months without daily or weekly recharge, it is to consider, if not a primary battery would fit even better, and then the effect becomes negligible. That is, if either the battery is recharged at least weekly or it is a primary one, then there is no need to model this effect.

Fourth, ambient temperature may be important. Several data sheets show significant differences in operating life for various temperatures, and the capacity and time variations of the presented measurements in Tables 7.1, 7.4, and 7.6 may be caused by different temperatures.

On the one hand, as evaluation of temperatures shows (Table 7.5), a detailed view on the relation of time to cell temperature and the relation of time to ambient temperature are needed to incorporate changes in battery capacity. On the other hand, data sheets show capacity variations for temperature changes in the order of 10°C: so that in absence of more detailed information, temperature should be considered at least in these terms, and then be assumed constant over discharge, at a certain error.

Although it is unimportant when a discharge current is drawn, its share on the discharge profile and its amplitude influence operating life significantly.

7.2. Battery Modelling

As hitherto multiply depicted, especially in the chapter *State of the Art*, battery models exist over a wide range of granularities regarding time scale, temperatures, geometries, and other parameters.

A battery for an embedded system has to provide energy over the ranges of months, weeks, days, or hours, depending on the application. On the one hand, if we consider a subterranean sensor node for the surveillance of buildings that is going to operate in terms of months without recharge, these systems will be driven by primary batteries because of low self-discharge; on the other hand, we can consider a sensor node that operates above surface and is going to be recharged every day. Mobile phones, cameras, sensors for medical equipment, or audio players are likely to be recharged after several days of operation. Therefore, assuming proper recharge in between, the battery model developed in this section is designed for one discharge cycle in the order of days or hours.

In the first subsection, it is discussed whether to model discharge time continuously, or if an abstraction will fit as well. The second subsection presents the conclusions that can be drawn out of a description of the energy demand in terms of event streams. The third subsection presents discharge profiles and the final battery model.

7.2.1. Voltage, Current, and Time Dependence of a Battery Model

An example discharge curve is depicted in Figure 7.3. It shows the relation of voltage to time for the previously mentioned lithium battery at the one hour discharge rate. The curve is monotonically decreasing: it is fast decreasing for the first few seconds, then with the same or slightly changing slope for the next 3000 seconds, and finally dropping sharply for the last 500 seconds.

Such a discharge curve might motivate research for a battery model considering a time dependent voltage, or a time dependent internal resistance. Here, time is used in the sense of discharge history.

Let us consider now the battery being part of a series circuit which consists also of a voltage regulator that drives further consumers of electric energy. The voltage regulator transforms any battery voltage (above cut-off potential) to the designated output voltage, and it keeps variations of the battery voltage away from the circuit it drives: thus the current drawn by the circuit will not change due to variations of the battery voltage. Consider the regulator dissipation free (high

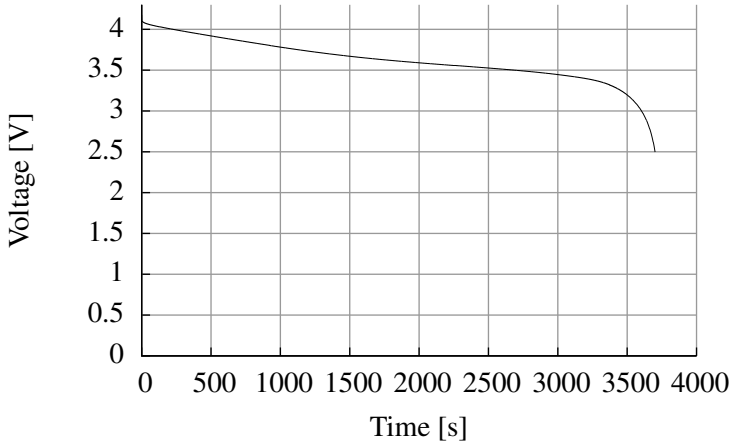


Figure 7.3.: Discharge Curve, Voltage by Time, for a SONY US 18650 LiMn Battery, 1 Hour Discharge, $V_{\text{cut-off}} = 2.5$ V

efficiency), then, because the current in a series circuit through each of the components is the same, the current drawn from the battery will not change as well.

That is, instead of considering time dependent voltage, a battery model based on the current is needed.

We have seen in the paragraph *discharge current* of Subsection 7.1.2 that lithium based cells do not provide better performance for pulsed discharge than for the same constant discharge rate. And we have seen in Subsection 7.1.3 that operating life is independent for variations of pulse durations: small pulses can be merged to wider pulses without loss of precision. Both results indicate an independence of discharge history within one discharge cycle.

Therefore, in the following, a battery model that is based on discharge history is rejected, and the advantage is given a battery model that is based only on the amplitudes of the different currents drawn by the final load, together with the share of each current on the discharge.

7.2.2. Methodology of Event Streams

The result of the previous subsection was to neglect discharge history. By only considering lengths of intervals, independent of the place of any interval in the time scale, history is also neglected by the methodology of event streams. As we have seen in the Preliminaries, Example 2.1, page 22, this independence ranges so far that demands computed for short intervals need not to take place within parts of the same size that are within longer intervals.

Worst-Case Deduction The idea of event streams is to check worst-cases. The worst-case for battery discharge is when the processor demands most electric power and the battery is the least capable of delivering it: if this case fails, the processor eventually shuts down, loses its data, and leaves an uncontrolled system with further consequences.

Let P_{Bat} be the battery's time dependent power, and let P_{Proc} be the processor's time dependent power consumption. Then for a valid system, it must hold

$$\min_{t>0} P_{\text{Bat}}(t) \geq \max_{t>0} P_{\text{Proc}}(t). \tag{7.1}$$

Analogously to the real-time feasibility test, we have to check available energy against demanded energy for every possible interval length. We can reduce the complexity if we disregard when a demand occurs and instead take the maximum over all intervals of equal length. The least available energy from the battery for any interval of length Δt is

$$E_{\text{Bat}}(\Delta t) := \inf_{b-a=\Delta t} \int_a^b P_{\text{Bat}}(t)dt. \tag{7.2}$$

The most demanded energy from the processor for any interval of length Δt is

$$E_{\text{Proc}}(\Delta t) := \sup_{b-a=\Delta t} \int_a^b P_{\text{Proc}}(t)dt. \tag{7.3}$$

Consequently, the check results in proving

$$E_{\text{Bat}}(\Delta t) \geq E_{\text{Proc}}(\Delta t) \quad \text{for all } \Delta t > 0. \tag{7.4}$$

Now, let $\Delta t > 0$ be arbitrary, and observe In-Equation (7.1):

$$\begin{aligned} E_{\text{Bat}}(\Delta t) &= \inf_{b-a=\Delta t} \int_a^b P_{\text{Bat}}(\theta)d\theta && (7.5)\\ &\geq \inf_{b-a=\Delta t} \int_a^b \min_{t>0} \left(P_{\text{Bat}}(t)\right) d\theta && (7.6)\\ &= \Delta t \cdot \min_{t>0} P_{\text{Bat}}(t) && (7.7) \end{aligned}$$

Applying the In-Equation yields:

$$\begin{aligned} E_{\text{Bat}}(\Delta t) &\geq \Delta t \cdot \max_{t>0} P_{\text{Proc}}(t) && (7.8)\\ &= \sup_{b-a=\Delta t} \int_a^b \max_{t>0} \left(P_{\text{Proc}}(t)\right) d\theta && (7.9)\\ &\geq \sup_{b-a=\Delta t} \int_a^b P_{\text{Proc}}(\theta)d\theta && (7.10)\\ &= E_{\text{Proc}}(\Delta t). && (7.11) \end{aligned}$$

That is, whenever In-Equation (7.1) holds, it proves the test to be true, and if it does not hold, the test is obsolete.

The evaluation works without any knowledge of a battery's operating life. The operating life was left out. Therefore, another approach to estimate the operating life is needed.

Energy Demands Although a worst-case deduction for operating life is not given, energy demands for interval sizes can still be useful.

Electric fuses are distinguished by nominal voltage, nominal current, and reaction speed. Below nominal voltage, the nominal current is passed through. The reaction speed defines the duration which an over-current needs to last before the fuse breaks off the circuit.

Similarly to this, some battery manufacturers give maximum ratings in their data sheets. For example, a cylindrical (size 17 mm by 50 mm) nickel-metal hydride battery with rated capacity 2450 mAh, 5 hour rate, and a maximal continuous discharge current of 8.1 A is described in manual [VAR2003], and the manual specifies a current of 13.5 A may last for 2 seconds before the battery is damaged.

The energy bound function which is based on event streams provides exactly the needed information about the discharging circuit. For each duration, the function returns the amount of energy needed within. Thus, the compliance to limits regarding short-term, mid-term, or long-term load can be proven.

Event Stream Approximation We know from the Preliminaries that approximation of event streams is done by a slope s from a certain index k on. We know from Chapter 5, that the sum of all slopes must not exceed 1 for a task set to be real-time feasible. The remainder, 1 minus the sum of all slopes, denotes the relative amount of time, the processor stays idle. The presented methods for processor shutdown introduced a special low-power task, which is characterised by an event stream as well.

With task set, idle task, and low-power task, every point in time is characterised by a power consumption of the processor. That is, we have a discrete set containing all possible power consumptions (over-estimations). Thus, if we assume a voltage regulator between battery and processor, we know the spectrum of possible amplitudes for the electric current that is drawn from the battery.

With the slopes given, we can compute the relative idle time, and therefore, we know the share of each element of the amplitude spectrum.

Approximation of event streams leaves the amplitude spectrum unchanged, as it does not change the worst-case execution time, but it changes the share: the approximation overestimates the occurrence rate of a task, that is the share of the task on the processing time. Now, 'amplitude' refers to the magnitude of the current, and 'share' refers to the relative time this current is drawn. We obtain adjustable precision for the set of shares, with low precision, if all event streams are approximated at low indices, and increasing precision with increasing approximation indices.

7.2.3. Discharge Profiles and Final Battery Model

As we have seen, share and amplitude of the currents that discharge the battery are important. Ambient temperature is also important but a capacity model that is also based on temperature is beyond this thesis, instead, the temperature is assumed to be constant, and the normalised capacity for a certain ambient temperature is assumed to be given by $C_{\text{norm}}(T)$, with T the requested temperature.

According to the Peukert-formula, a battery performs differently for different discharge currents. Let a non-constant discharge consist of the constant discharge currents $I_1,\ldots,I_n$, let $C_1,\ldots,C_n$ be the corresponding capacities, and let $\tilde{t}_1,\cdot,t_n$ be the discharge times, where a capacity of C_i is obtained when a fresh battery is completeley discharged with the current I_i, within a time of $\tilde{t}_i$ time units; in short: $C_i = I_i \cdot \tilde{t}_i$, $i = 1,\ldots,n$. Then the authors of reference [RF1900] suggested to use the relation

$$\frac{I_1 \cdot t_1}{C_1} + \cdots + \frac{I_n \cdot t_n}{C_n} = 1 \tag{7.12}$$

for estimating the battery's operating life, where t_i denotes the time that a discharge current of I_i is drawn. The equation becomes

$$\frac{I_1 \cdot t_1}{I_1 \tilde{t}_1} + \cdots + \frac{I_n \cdot t_n}{I_n \tilde{t}_n} = 1 \tag{7.13}$$

$$\Rightarrow \qquad \frac{t_1}{\tilde{t}_1} + \ldots \frac{t_n}{\tilde{t}_n} = 1. \tag{7.14}$$

For generality, the authors did not insist on a specific relation of current to capacity or discharge time, although they suggested to use the Peukert-relation[2]. Multiplying Equation (7.14) with C_{norm}, and replacing C_{norm} for each term belonging to index i with $I^{\text{pc}} \cdot \tilde{t}_i$ yields:

$$I_1^{\text{pc}} \cdot t_1 + \cdots + I_n^{\text{pc}} \cdot t_n = C_{\text{norm}}, \tag{7.15}$$

with pc the Peukert-coefficient, and C_{norm} the normalised capacity, both depending on battery chemistry.

Validation of the Rossander and Forsberg Idea

We know from earlier measurements with the SONY lithium-manganese cell: if a battery is discharged with current I_1 to cut-off voltage, then it can further be discharged with current $I_2 < I_1$ for some amount of time. According to the Equation (7.14) this behaviour equals to discharge 100% of the capacity with current I_1 and then still be able to discharge some percentage with current I_2: the battery would have a capacity of over 100%.

Let us study this effect, instead of using Equation (7.14), let us verify its equivalent, Equation (7.15). If we discharge specific amounts of normalised capacity, then we should be able predict the remaining capacity. The prediction error is then the difference of predicted capacity to real capacity at discharge termination.

The battery ULT 18650 FP is a lithium secondary cell of the same size as the previously mentioned SONY cell, its chemistry is not lithium manganese, but consists of lithium, iron, and phosphate.

In Table 8.12, page 159, Chapter 8, we find the Peukert-coefficient to be pc $= 1.13$, the normalised capacity to be $C_{\text{norm}} = 3090$ with a standard deviation of 7 As.

If we discharge specific amounts of the normalised capacity, we are able to estimate the resulting operating life, or the discharged capacity. Table 7.7 shows for some discharge currents

[2] They suggest a Peukert-*like* solution taken from an earlier book named "Hülfsbuch der Elektrotechniker" by the authors Grawinkel and Strecker, the relation was given for a special kind of battery. Peukert was just the first to find the formula to be correct for a range of batteries.

and some amounts of normalised capacity the time that each current must be drawn, according to the Peukert-formula $I^{1.13} \cdot t = C_{\text{norm}}$ – dashes mark unused combinations. For example to discharge 1200 As of the normalised capacity, we need to discharge for 390,607 ms with a current of 2.7 A, note, this discharges a real capacity of $2.7 \cdot 390,607\,\text{As} = 1055\,\text{As}$.

Table 7.7.: Computation of Discharge Times to the Discharge of Specific Amounts of Normalised Battery Capacity, Currents 0.675 A, 1.35 A, 2.7 A

	c_1 =300 As	c_2 =600 As	c_3 =900 As	c_4 =1200 As
I_1 =0.675 A	467,744 ms	-	1,403,231 ms	-
I_2 =1.350 A	-	427,439 ms	641,158 ms	854,878 ms
I_3 =2.700 A	-	-	292,955 ms	390,607 ms

According to the table, there are several combinations of currents and time to discharge an amount of 2700 As normalised capacity. Regardless of the combination used, for the ULT 18650 FP battery, an amount of 390 As normalised capacity remains for further discharge. Let us discharge the rest with a current of 0.5 A. According to Rossander and Forsberg, we should obtain a rest capacity of

$$\frac{390}{0.5^{1.13}}\,\text{s} \cdot 0.5\,\text{A} = 426.8\,\text{As}$$

or a remaining discharge time of 854 s, for all possible combinations!

Table 7.8 shows the measurement results for several combinations of currents and desired amount of normalised capacity, all combinations yield the same sum of 2700 As normalised capacity. For explanation, a combination "$I_2 : c_3$, $I_3 : c_3$, $I_1 : c_3$" denotes: first, discharge with a current of I_2 a normalised capacity of c_3, corresponding to the symbols of Table 7.7; second, discharge with a current of I_3 a normalised capacity c_3; and third, discharge with a current of I_1 a normalised capacity of c_3. This yields a normalised capacity of $c_3 + c_3 + c_3 = 2700$ As, denoted in column C_{norm}. The column C_{real} denotes the real capacity that was discharged by the currents I_2, I_3, I_1. The next two to last columns denote the measured values for the experiments. The column C_{rest} denotes the capacity that has been discharged with a current of 0.5 A until the cut-off voltage was reached, and the column C_{term} denotes the total capacity that has been discharged when discharge was terminated.

First, we observe that although different discharge sequences have been used, the capacity at termination is always almost the same. Second, we observe that the predicted rest capacity that can be discharged was underestimated by at least 80%: the predicted value was 426.8 As and we measure a value between 742 As to 824 As. Third, we observe that sequences consisting of the same combinations of currents and capacities yield the same rest capacity, compare measurements 1 and 4, 2 and 5.

We can stop at this point and confirm that the idea of Rossander and Forsberg holds, though it is a significant underestimation. But let us proceed the investigation instead.

Now, let us vary the amount of normalised capacity. Table 7.9 provides other discharge times, currents, and normalised capacities than the previous.

A discharged normalised capacity of 1800 As yields a rest normalised capacity of 1290 As, which makes $1290/0.5^{1.13}\,\text{s} = 2824\,\text{s}$ or a real capacity of $2824\,\text{s} \cdot 0.5\,\text{A} = 1412\,\text{As}$. A discharged

Table 7.8.: Discharging the Same Amount of Normalised Capacity by Varying Pre-Discharge Current and Corresponding Normalised Capacity per Current, Day of measurements: 2009-07-02, Temperature: 21.5°C

No.	Sequence	C_{norm} [As]	C_{real} [As]	C_{rest} [As]	C_{term} [As]
1	$I_2 : c_3, I_3 : c_3, I_1 : c_3$	2700	2607	757	3355
2	$I_2 : c_4, I_3 : c_4, I_1 : c_1$	2700	2518	824	3343
3	$I_2 : c_2, I_2 : c_4, I_3 : c_3$	2700	2572	774	3348
4	$I_3 : c_3, I_1 : c_3, I_2 : c_3$	2700	2607	742	3341
5	$I_2 : c_3, I_2 : c_1, I_3 : c_3$	2700	2518	823	3343

Table 7.9.: Computation of Discharge Times to Discharge Specific Amounts of Normalised Battery Capacity, Currents 1 A, 2 A, 3 A

	c_2 =600 As	c_3 =900 As	c_4 =1200 As
I_4 =1.000 A	600,000 ms	900,000 ms	1,200,000 ms
I_5 =2.000 A	274,149 ms	411,224 ms	548,229 ms
I_6 =3.000 A	173,382 ms	260,073 ms	346,764 ms

normalised capacity of 2400 As yields a rest normalised capacity of 890 As, which makes $890/0.5^{1.13}\,\text{s} = 1948\,\text{s}$ or a real capacity of $1948\,\text{s} \cdot 0.5\,\text{A} = 974\,\text{As}$. Table 7.8 shows the results for measurements where 1800 As and 2400 As normalised capacity were discharged before the discharge with 0.5 A down to cut-off potential is started. Measurement 3 was repeated in measurement 4.

Table 7.10.: Discharging Specific Amounts of Normalised Capacity with Varying Distributions of Current and Normalised Capacity per Current, Day of Measurements: 2009-07-03, Temperature: 22.5°C

No.	Sequence	C_{norm} [As]	C_{real} [As]	C_{rest} [As]	C_{term} [As]
1	$I_5 : c_3, I_6 : c_3$	1800	1600	1748	3348
2	$I_6 : c_3, I_5 : c_3$	1800	1600	1755	3355
3	$I_5 : c_4, I_6 : c_4$	2400	2133	1210	3343
4	$I_5 : c_4, I_6 : c_4$	2400	2133	1213	3347

Again, we observe that the capacity at discharge termination is the same for all measurements. We observe that computed rest capacity for a discharged normalised capacity of 1800 As is underestimated by 200 As, and the other, 2400 As, is overestimated by 200 As.

Similar measurements have been repeated, the results are shown in Table 7.11. The previous measurements made are confirmed. Especially observe measurements 3 and 4, where a normalised capacity of 3000 As is to be discharged, the estimated rest then is 90 As normalised

capacity, which is 98 As real by 0.5 A discharge, but discharge by 0.5 A yields 624 As real discharged rest capacity!

Table 7.11.: Discharging the Same Amount of Normalised Capacity with Varying Distributions of Current and Normalised Capacity per Current, Repeated and Extended Measurements, Days of Measurements: 2009-07-03 and 07-04, Temperature: 21.5°C

No.	Sequence	C_{norm} [As]	C_{real} [As]	C_{rest} [As]	C_{term} [As]
1	$I_4 : c_3, I_5 : c_3, I_6 : c_3$	2700	2497	858	3355
2	$I_6 : c_3, I_4 : c_3, I_5 : c_3$	2700	2497	858	3356
3	$I_5 : c_4, I_6 : c_4, I_4 : c_2$	3000	2731	624	3355
4	$I_4 : c_2, I_5 : c_4, I_6 : c_4$	3000	2731	629	3360
5	$I_2 : c_3, I_3 : c_3, I_1 : c_3$	2700	2598	762	3361
6	$I_2 : c_4, I_3 : c_4, I_1 : c_1$	2700	2519	830	3349
7	$I_2 : c_2, I_3 : c_4, I_1 : c_3$	2700	2574	788	3361
8	$I_3 : c_3, I_1 : c_3, I_2 : c_3$	2700	2598	756	3354
9	$I_2 : c_4, I_1 : c_1, I_3 : c_4$	2700	2519	833	3353

The conclusion of this subsubsection is that the approach of Rossander and Forsberg disqualifies for predicting the operating life.

Battery Model with Discharge-End Detection

At least for pulse durations below 10 ms, the measurements made falsify the assumption that pulsed discharge performs better than constant discharge. They reject the assumption of a difference between pulse widths. And they reject the assumption that the order in which different currents are drawn does matter in terms of discharged capacity.

A careful look at the measurement results finds the order in which different currents are drawn *does matter in a specific point*. Although the order of the currents was varied, in the previous sub-subsection, the termination condition was always the same,: discharge with 0.5 A until the battery voltage drops below 2.5 V at the terminals.

If we apply the Peukert-formula on the normalised capacity for the ULT 18650 FP cell, we obtain a real capacity for a current of 0.5 A:

$$\frac{3090}{0.5^{1.13}}\,\text{s} \cdot 0.5\,\text{A} = 6763\,\text{s} \cdot 0.5\,A = 3381\,\text{As}.$$

This is close to the capacity that all measurements showed.

The capacity that a discharge current of 0.5 A yields was measured, and the results are depicted in Table 7.12. The measurements confirm the estimated real capacity.

For the measurements made, we find the following true:

Proposed law 7.2.1. *The remaining discharge time and the capacity that is discharged in total are determined by the capacity that is associated with the current drawn when discharge terminates minus the capacity that has already been discharged.*

Table 7.12.: Discharge of the ULT 18650 FP $LiFeO_4$ Secondary Battery, Day of Measurement: 2009-07-04, Temperature: 22°C

No.	Current [A]	Time [s]	Capacity [As]
1	1.35	2286	3080
2	0.5	6807	3393
3	0.5	6806	3393

Associated capacity is to be understood as the real capacity that is obtained by discharge with the corresponding constant current.

In symbolic notation, the law proposes

$$\int_0^t I(s)ds + I_{\text{term}} \cdot t = C_{I_{\text{term}}}, \tag{7.16}$$

or using the Peukert-formula for $C_{I_{\text{term}}}$

$$\int_0^t I(s)ds + I_{\text{term}} \cdot t = \frac{C_{\text{norm}}}{I_t^{\text{pc}}} \cdot I_t, \tag{7.17}$$

where t denotes the point in time when discharge with current I_{term} starts, and $I(s)$ denotes the time dependent current (discharged before time point t), and $C_{I_{\text{term}}}$ denotes the capacity that is associated with the current I_{term}.

With a Peukert-coefficient of almost 1, the law is not questioned by measurements made for the SONY lithium-manganese cell.

Let us verify the proposed law by further measurements with the ULT 18650 FP battery. Table 7.13 shows measurement results, where the battery was discharged prior to the termination current. Column "Pre-discharge" denotes the discharge which has taken place before discharging to cut-off potential by a current denoted in the column I_{term}. The column C_{rest} depicts the discharged capacity until cut-off potential has been reached. The column C_{total} depicts the total discharged capacity. If we use the Peukert-formula, we expect total capacities for currents 1.35 A and 3 A to be 2972 As and 2679 As. All measurements confirm the law, further, measurements 1, 2, and 3 falsify the assumption that a lower current prior to termination may provide better battery performance.

If we recognise temperature variation, then we can enrich the law by using $C_{\text{norm}}(T)$, where T denotes the temperature while discharge takes place.

Discharge profiles

The construction of a time dependent load profile for the processor in terms of processing demand requires to generate a set of profiles out of the information that an event stream provides: an event stream is an abstraction which aggregates a set of schedules, and this aggregation needs to be dissolved. A time dependent load profile in terms of electric current drawn from the battery would require the same computations, and is yet not precise because the processor's power

Table 7.13.: Pre-Discharging and Discharging Battery ULT 18650 FP, Day of Measurement: 2009-07-05, Temperature: 21°C

No.	Pre-Discharge	I_{term} [A]	C_{rest} [As]	C_{total} [As]	Error
1	3 A for 933,332 ms = 2795 As	1.350	385	3180	6.7%
2	2 A for 1400,000 ms = 2795 As	1.350	292	3087	3.6%
3	0.675 A for 4148,149 ms = 2794 As	1.350	318	3113	4.5%
4	2 A for 1250,000 ms = 2495 As	3.000	238	2733	2.0%
5	1 A for 2500,000 ms = 2494 As	3.000	294	2788	4.1%
6	0.675 A for 3703,703 ms = 2495 As	3.000	361	2855	6.6%

consumption may also vary with data and the sequence of commands a task's implementation consists of.

According to measurements made, time dependent load profiles are not needed for an estimation of a battery's operating life. On the one hand, the idea of Rossander and Forsberg clusters all currents by their amplitude, and on the other hand, the proposed law adds up the currents that have been discharged and uses the actual current drawn while assuming it is constant for predicting the remaining operating life.

In both cases, a discharge profile containing the currents' amplitudes and their shares is sufficient. The combination of power values and event streams, see especially in Chapter 5, provides this information. Each task was assigned a power value denoting the average power consumption when processing jobs of this task, this power value denotes the current's amplitude. And the event stream of each task was approximated by a slope to determine the long-term demand of the task: this slope denotes the share of the current.

In summary, we have a current I_τ for each task, and a share s_τ, a processor dependent current is associated when the processor is idle as well (by P_{idle}, processor idle power), and its share is $1 - \sum_{\tau \in \Gamma} s_\tau$. This information determines a discharge profile.

7.3. Summary

In this chapter, we have seen many parameters influencing a battery's operating life, and if we assume proper charge between cycles, then for modelling one discharge cycle only a few are necessary for a battery model.

Detailed parameterisation of the battery's construction may provide useful but cannot be obtained, if not from the manufacturer, and this is unlikely because it may disclose important confidential information. Dismantling a device is not an option either.

Further information on the shape of the discharge current, like discharge history within the discharge cycle, or pulsed discharge for lithium-based secondary batteries, does not lead to a difference in terms of operating life.

Several measurement series, to be continued in the Evaluation Chapter, allow for falsifying several assumptions regarding the estimation of operating life. Amplitude and share of the discharge currents, temperature, and chemistry remain as most important qualities. A simple model

that incorporates these qualities was presented.

On a future embedded system, the proposed model may be used in combination with an energy counting device ('Coulomb counting', Chapter 3). The device is supposed to count down remaining energy within the battery with respect to a specific discharge current, which the embedded system must be able to draw, at all times. This will ensure a reliable operation without any failures.

Other energy sources than batteries might require other models, and new discussions and tests may be necessary.

8. Experimental Setup and Evaluation

Evaluation of algorithms and validation by measurements increases confidence in theoretical work. Using examples does not allow for generalising results, but general assumptions can be validated, and if an assumption does not hold for an example, then it is not generally valid. Experiments help us to estimate the significance of a property or influence factor.

This chapter presents an evaluation of the algorithms and models described in this thesis on certain example task sets and batteries.

Note, the decision, whether processor slowdown is better than processor shutdown purely depends on the designated hardware for the embedded system. Therefore, experiments in this direction would only serve as a further proof of concept and are left out; this extends to the final check on operating life with the proposed battery model. The three categories processor slowdown, processor shutdown, and battery modelling are evaluated separately.

The first section of this chapter is a description of the implementation of the solutions presented in Chapters 5, and 6. The implementation was already used to generate the figures in both chapters. Here, the implementation is used for a further study of the algorithms on certain selected task sets.

The second section is a description of the measurement conditions for measurement data presented in Chapter 7. Additionally, the presented measurements of Chapter 7 are extended to nickel-metal hydride batteries. The evaluation of the battery model is included in this section.

8.1. Evaluation of Presented Concept

The conceptual work flow described in Chapter 4 has been fully implemented. Together with the evaluation of the algorithms, this implementation is described in the following.

Task sets for evaluating the algorithms distinguish in their structure: deadlines equal to periods, one or more deadlines being only a fraction of corresponding periods, deadlines longer than periods, and one task set contains tasks with jitter. Further, the used real-time feasibility test allows for weighing up precision against computational complexity; with the help of the task set examples, the relation of increasing approximation steps to achieved precision is studied.

As local and global processor slowdown have the same frequency/power relation as input, both are compared regarding power peaks in the task set.

Algorithms for processor shutdown, periodic, and task-dependent, are compared to the online algorithm LC-EDF [LRK2003], as well as optimal shutdown without and with procrastination.

The first subsection is a description of the implementation's structure and some of its abilities. Task sets for evaluating the algorithms are presented in the second subsection. The implementation allows for evaluation of slowdown and shutdown separately: the first is presented in the next to last subsection, and the second in the last subsection.

8.1.1. Implementation

Flexibility regarding a task's timing behaviour is one of the goals stated in the Introduction. This is continued in the software used for evaluating the algorithms.

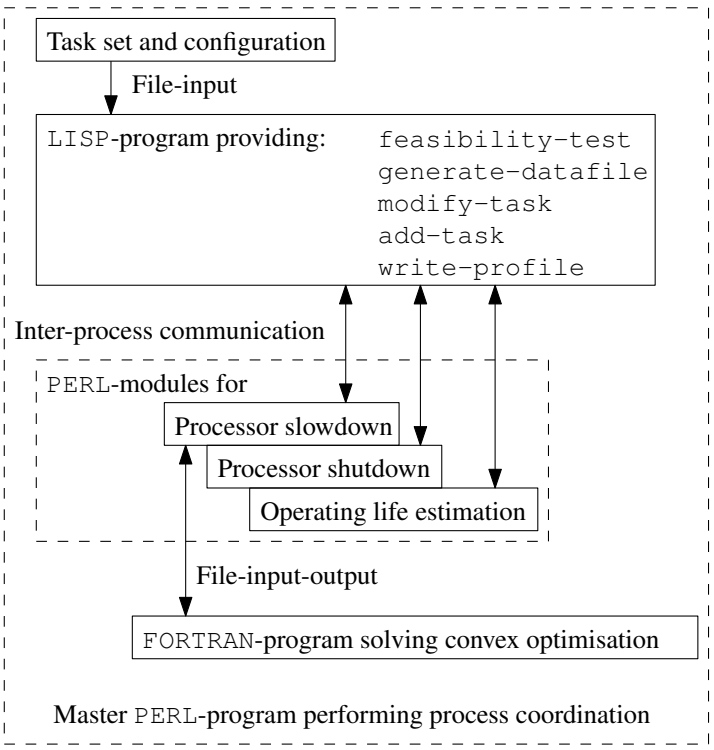


Figure 8.1.: Process Structure for Optimisation of Operating Life

Figure 8.1 depicts the process structure, which implements the conceptual work-flow from Figure 4.4, page 63. From a file, task set and configuration are read by the `LISP`-program. The program is interactive and provides the functions

- `feasibility-test`, which returns 'yes' or 'no' depending whether the input denotes a real-time feasible system,
- `generate-datafile`, which writes constraints according to Equation (5.26), page 78, in matrix form to file for further processing by convex optimisation solvers,
- `modify-task`, which allows for changing all task properties except *taskname*, a task's identifier,
- `add-task`, which allows for adding a new task to the task system, and
- `write-profile`, which writes the discharge profile of the task set to file for further processing.

The master PERL-program coordinates the subprocesses for slowdown optimisation, shutdown optimisation, and the prediction of the operating life. Each of the routines are implemented as PERL-modules. Inter-process communication allows for each module to interact with an instance of the LISP-program; there are multiple instances at once – one for each module – optimisation is done in parallel.

Local and global processor slowdown are each implemented in an own PERL-module. Each module calls a FORTRAN-program to solve the corresponding optimisation problem.

Periodic shutdown is implemented as PERL-module. The module is an implementation of the modified solution as of Algorithm 6.2.3, page 87.

Task-dependent shutdown is implemented in two PERL-modules: one represents shutdown in between two task instances, Algorithm 6.3.2, page 94; and the other represents the modified version, which enables shutdown in between many task instances, Algorithm 6.3.3, page 98.

The prediction of operating life is implemented according to the battery model given in Chapter 7 in Equation (7.16), page 122.

The division into modules allows for an easy extension with other optimisation functions, as well as modifications of existing routines. The interface for each module consists of the communication with an instance of the LISP-program and hardware specific inputs, for example a set of break-even times, power consumptions of the processor for specific settings, set of Peukert-coefficients etc.

Task Set Structure The task set definition is implemented in LISP, a functional programming language with lists as basic data structure. Consider a task set as a list of task definitions, and each definition of a task as a list of fields and field-values, then this allows for an easy extension:

The language LISP allows to dynamically add fields. Anywhere in the program, you can add a field by calling the statement **setf** (**getf** task :**field-name**)<*field-value*> to the data structure *task*; fields are removed by setting their content, *field-value*, to nil. Thus, a non-nil value indicates the presence of a field (like non-null pointer in several other programming languages).

Further, since event streams are sequences of numbers to be processed sequentially, an underlying list structure is advantageous.

Listing 8.1: Task Data Structure, Mandatory Fields

```
 1 (:taskname "3"
 2  :wcet 420.000000
 3  :deadline 15,000.000000
 4  :es (
 5       :sequence (0 31,110.0 71,110.0)
 6       :progression "slope"
 7       :approximation (
 8                       (0.000032 ((1 1)))
 9                       (0.000025 ((2 1))))
10      )
11 )
```

The implementation of the task definition is presented in Listing 8.1. List elements belonging to the same list are separated by whitespace; each list is enclosed in parentheses; a list can be

element of another list. The field :**taskname** holds an identifier for the defined task: this value is not to be changed. The fields :**wcet** and :**deadline** hold worst-case execution time and relative deadline: these values can be modified, e.g. when probing for feasible durations for a low-power task.

The field :**es** describes the task's occurrence behaviour. The nested field :**sequence** holds a finite set of elements $a_1, a_2, a_3, \ldots, a_n$ denoting the event stream. The field :**progression** is either set to "es" or "slope": the former denotes to continue the set of elements by appending elements, that is $a_{n+1} = a_n + a_2$, $a_{n+2} = a_n + a_3$, ...; the latter, a value of "slope", indicates for the program to compute requested stream elements by approximation – the definition of which is given in the field :**approximation**.

The field :**approximation** consists of a list of slopes $s_1, \ldots, s_p$, each slope is paired with a list of pairs containing associated start indices k and modules m (LISP-code):

$$\begin{array}{l} (\,(s_1 \quad ((k_1\ m_1)\ \ldots\ (k_j\ m_j))) \\ \quad\vdots \\ \quad (s_p \quad ((k_1\ m_1)\ \ldots\ (k_r\ m_r)))\,). \end{array}$$

A list element of (0.0001 ((2 1))) denotes a slope of $s = 0.0001$ with start index $k = 2$ and module $m = 1$: this allows to construct every stream element a_n with $n > k = 2$ by $a_n = a_2 + s \cdot (n-k)$. Whereas a list element of (0.0001 ((2 2))) would define the same slope with same start index, but $m = 2$ allows only for a construction of stream elements with even ($nmodm = 0$) index. As indicated, several index definitions can be combined: a list (0.0001 ((2 4) (2 5))) would extend the set of allowed indices to the union of $A = \{2,4,8,12,16,\ldots\}$ and $B = \{2,7,12,17,\ldots\}$. The approximation implementation provides a very flexible event stream definition.

Additional fields for the various stages of the optimisation procedure are introduced in the next paragraphs.

Real-Time Feasibility Test The function `feasibility-test` requires one more field, which discloses information about the approximation to apply. The added field :**test-index** denotes the number of approximation steps or test index; its value gives the number of stream elements used for performing the real-time feasibility test, and gives the index at which approximation is to take place. This value is the implementation of the variable k_τ, Equation (2.10), page 28, of the linear approximation of the event stream of task τ.

For example, to approximate after the third element, set :**test-index** 3. Now, the LISP-program will compute the first three elements of the event stream according to the other fields :**sequence**, :**progression**:, and :**approximation**, and the program will determine the minimal slope allowed for index 3 using the approximation definition.

Energy Optimisation with Processor Slowdown The function `generate-datafile` requires one more field disclosing information about the power a task instance causes the processor to consume at maximal processing speed. The added field :**power-factor** denotes the power consumption per time unit, or equivalent: the reciprocal of the associated (ohmic) resistance.

This field is used to direct the optimisation of per-task slowdown factors, and it is used to determine energy consumption per task instance.

The function also evaluates the field :**test-index**. First, for the feasibility test, the field value determines the slope used for the task in the constraint matrix; second, the maximum over all test indices yields the number of lines for the constraint matrix. The constraint matrix is constructed according to Equation (5.26), page 78.

The PERL-module for optimisation with global slowdown calls on this data a FORTRAN-program, which is an implementation of the global slowdown problem depicted in (5.14), page 75. After optimisation, the result is read from the solver, then the result is incorporated by calling the function change-task for every task in the task set: this introduces and sets the new field :**scale** to γ, that is, it sets for every task the global slowdown factor indicated by γ. A final call to the LISP-program will write the data into a specific file. This gives the opportunity to save the optimisation result, and to call the whole optimisation suite on the now modified task set once more at a later point in time, for example to test if further improvements by some not yet found energy optimisation can be made.

The PERL-module for local slowdown processes the data in the same way as before with the difference that not a scalar is obtained, but a solution vector. The used FORTRAN-program for local slowdown is an implementation of Equation (5.17), page 76. Analogously, the field-value :**scale** is set to an individual value for each task.

To compare the linear with the non-linear objective for local slowdown, a PERL-module incorporating the linear objective was written. It differs from the former only in using another FORTRAN-solution, which is an implementation of Equation (5.19), page 76.

The FORTRAN-programs were all written with the help of the ALGENCAN [MB2009] optimisation suite for non-linear, convex optimisations.

Periodic Processor Shutdown The PERL-module implementing the algorithm for periodic shutdown, Algorithm 6.2.3, page 87, adds a new task to the task set by invoking the function add-task. Further, it uses repeatedly the functions feasibility-test to check the solution, and change-task to modify the solution.

The final solution is written to file, as before. The added task in the final solution gets an additional field, :**break-even-time**, to save the break-even time, which the solution was computed for.

For comparison, also the unmodified variant of periodic processor shutdown was implemented, according to Algorithm 6.2.2, page 85.

Task-Dependent Shutdown of the Processor A PERL-module was written to implement both solutions for task-dependent shutdown: Algorithm 6.3.2, 94, the solution for computing task-dependent shutdown in between two instances; and Algorithm 6.3.3, 98, the solution to shutdown in between many instances. The procedure is the same as in the previous paragraph: via function add-task the new low-power task is introduced, and later on, it is modified using change-task. The final solution is written to file and the new task contains the field :**break-even-time**.

Nevertheless, task-dependent shutdown requires to solve two different real-time feasibility

tests. The two different tests contain the same tasks with the same event streams: the only difference is that each contains a task which is to be started later than the others. This behaviour was modelled by deadline adjustment for the real-time feasibility test (compare Equations (6.8), and (6.9), page 91). The adjustments are made temporarily for each case: first, with the function `change-task`, the deadline for the task that is to be started differently is modified; second, the corresponding real-time feasibility-test is performed; and last, the deadline is set back to its original value, and the other case can be tested using the same steps.

For shutdown in between two instances, the added field :**`inherit-es`** "taskname" causes the low-power task's occurrence behaviour to be inherited from another task given by "taskname". This will link the fields :**`es`** of both tasks together.

For shutdown in between many instances, the same field is added, but the low-power task will receive a modified copy of the :**`es`** field. Because not every stream element is used, a further added modificator field :**`every-es-index`** <n> will select only every *n*-th element for the low-power task.

Online Algorithms for Processor Shutdown For comparison with existing algorithms for processor shutdown, and to compute ideal shutdown of the processor with and without procrastination, a stand-alone `LISP`-program has been implemented.

Online algorithms cannot operate on an abstract schedule description like event streams. *Online* requires a schedule with absolute deadline, i.e. a timeline with time points denoting job creation times.

Fortunately, an event stream can be interpreted as a schedule in terms of processing demand: it serves as a worst-case schedule. Superposition of the event streams of all tasks in the task set yields a timeline with job creation times for the task set. The timeline starts at an initial time point '0', where all tasks become instantiated (synchronicity assumption). The schedule is to end at a certain point in time, for example the hyper-period in case of a periodic task set, or some multiple, in case of jitter.

The schedule has the advantage that it provides the least possible idle time in total, but the disadvantage that the idle-intervals of this schedule might not be representative for a set of 'real' schedules. Thus, a variation is needed.

For every task, the additional field :**`offset`**<*value*> will be used to denote schedule variations. A non-zero value has the effect, that the synchronicity assumption for this task is not taken but the task is said to be instantiated *value* time units later, and after that assumed to occur according to its event stream: the processing demand which is induced by the event stream is shifted *value* time units to the right.

By generating many task set definitions – only distinguishing in the distribution of offsets – online algorithms are evaluated, and the set of results is analysed for each online method.

Discharge Profiles and Battery Model The function `write-profile` evaluates the fields :**`scale`** and :**`break-even-time`**. A transformation that computes out of *scale* and *break-even time* the processor's power values serves as input. With this transformation, the function determines a discharge profile suitable for the battery model of discharge-end detection, Equation (7.16), page 122.

For each task, first, the value of *scale* is used to determine the fraction of the maximal processor frequency to run each job of the task with. The value *power-factor* gives the power, which a job of the task causes the processor to consume while it is processed. Scale and power-factor together with the mentioned transformation yield the current that the processor draws while it processes a job of the task.

Second, an evaluation of the fields :**test-index** and :**approximation** determines the task's slope, which gives the share of the current on the discharge profile. This step is performed for each task, including the low-power task. Any remaining *share* represents the processor's idle time, and is associated a current as well.

A PERL-module implementing a battery model can now compute resulting operating life. PERL-modules – battery models – may distinguish by Peukert-coefficients.

8.1.2. Task Sets for Evaluation

The optimisation algorithms are to be evaluated with several task sets. Regarding the possibility to model almost arbitrary occurrence behaviour with the event stream model, only a large set of tasks could cover this range – deadline variations not included. Even a large set of examples cannot replace a prove, but a restriction to some special cases will suffice for a study. Special cases are: strict periodic task sets with periods in the same order of magnitude; with periods in different orders of magnitude; with deadlines less than periods, greater, or both; and task sets with the same variations of periods and deadlines but which include jitter (and clock skews are yet not included).

The examples presented in the following cover certain special cases. The first three examples depict a strict periodic task set, with deadlines equal to periods in the first case, one short deadline in the second case, and two short deadlines in the third case; the fourth example contains periodic and sporadic tasks with several short deadlines; and the fifth example comes with arbitrary deadlines and jitter. The first and the two latter examples represent embedded systems designed for the *real world*.

Palm-Pilot Examples In [LHW2002], data for a Palm-pilot are given. The data consist of seven periodic tasks with periods ranging from 20 to 150 ms and deadlines equal to their periods. The hyper-period is 600 ms. The processor utilisation is 86.2%. Additionally, the reference provides information on average power consumption.

All values are shown in Table 8.1. The tasks are enumerated: it is c a task's worst-case execution time, T a task's period, d a task's deadline, and P the average power consumption a task instance causes.

For processor shutdown, since the Palm-pilot task set is strict periodic, i.e. all deadlines are equal to periods, the LC-EDF method ([LRK2003]), as well as the method of [NQ2004] are applicable here.

A deadline reduction neither changes the utilisation of a task set nor the number of tasks to occur, but it limits the available spare time for power saving with processor shutdown: the real-time constraints in the notion of In-Equation (2.5) become stronger.

To study the effect of one short deadline, the Palm-pilot example is modified. The deadline for one task is reduced to a still real-time feasible value. The modification is given in Table 8.2.

Table 8.1.: Task Set of the Palm-Pilot

Task	c [ms]	T [ms]	d [ms]	P [mW]
1	5	100	100	90
2	7	40	40	60
3	10	100	100	150
4	6	30	30	140
5	6	50	50	125
6	3	20	20	125
7	10	150	150	40

If we consider procrastination based shutdown: Equation (3.1), page 51, yields for this task set a negative delay; it is $\sum_{i=1}^{7} \frac{c_i}{d_i} = 1.31$. Shutdown methods based on this equation are not applicable.

Table 8.2.: Modification 1 of the Palm-Pilot

Task	c [ms]	T [ms]	d [ms]	P [mW]
6	3	20	5	125

To study the effect of two short deadlines, the previous made modification is extended to another task. The changes stated in Table 8.3 still yield a real-time feasible task set. As for the first modification of the Palm-pilot example, let us compute Equation (3.1): it is $\sum_{i=1}^{7} \frac{c_i}{d_i} = 1.71$ – again we had to replace periods by deadlines.

Table 8.3.: Modification 2 of the Palm-Pilot

Task	c [ms]	T [ms]	d [ms]	P [mW]
3	10	100	20	150
6	3	20	5	125

Satellite System Example This task set was taken from [BW1995]. It represents control and data acquisition software for an experimental television satellite. It consists of ten periodic tasks with periods ranging from 10 ms to 1000 ms, and four sporadic tasks with minimal distances between two instances ranging from 0.96 ms to 187 ms. Except for tasks 2, 8, 13, and 14, all task deadlines are less than periods; for tasks 4 and 5 deadlines are less than a tenth of their corresponding periods. The processor utilisation is 87.2%. The hyper-period (including sporadic tasks) of this example is 1,122,000 ms. All tasks of the system are to be started synchronously, or are assigned a delayed start time (offset), except for sporadic tasks. The reference does not provide any information on power consumption. The data are shown in table 8.4, where sporadic tasks are marked with “s” in the offset column.

Let us analyse the applicability of procrastination based shutdown, which makes use of Equation (3.1), page 51. If we include all tasks, periodic and sporadic, we obtain $\sum_{i=1}^{14} \frac{c_i}{d_i} = 1.48$. If

Table 8.4.: Task Set of the Olympus Attitude and Orbital Control System

Task	c [ms]	T [ms]	d[ms]	Offset [ms]
1	0.28	50	9	0
2	1.76	10	10	0
3	2.13	200	14	50
4	1.43	200	17	150
5	1.43	200	17	0
6	1.43	100	24	0
7	8.21	100	50	50
8	52.84	200	200	50
9	5.16	1,000	400	200
10	6.91	1,000	900	200
11	0.18	0.96	0.63	s
12	3.19	62.5	30	s
13	4.08	100	100	s
14	2.50	187	187	s

we exclude sporadic tasks, we obtain $\sum_{i=1}^{10} \frac{c_i}{d_i} = 1.04$. In either case, shutdown methods based on this equation are not applicable.

Aircraft Controller Example The last task set was taken from [TC1994]. It represents software for an aircraft controller. It consists of 17 tasks of which 9 have jitter. Periods are in the range of 800 μs to 1,000,000 μs. Deadlines are arbitrary: some less than periods, some equal to periods, and some greater than periods. Processor utilisation is 65.2%. The hyper-period of this example is 59,000,000 μs. The authors do not provide power information. Table 8.5, page 134, depicts the data.

Here, if a periodic task has jitter, this denotes that the task may occur half the jitter earlier than the period, and half the jitter later than the period.

Let us analyse the applicability of procrastination based shutdown that makes use of Equation (3.1), page 51. First, the task set has deadlines greater than periods, which would formally disable these shutdown methods, second, it is $\sum_{i=1}^{17} \frac{c_i}{\min\{T_i,d_i\}} = 1.18$ leading to a negative delay, thus methods based on this equation are not applicable, third, this task set contains jitter and therefore the method of [NQ2004] is hardly applicable, because it needs to recompute online its – offline computed – optimisation data whenever a task occurs with jitter.

None of the hard real-time feasible solutions presented in Section 3.2.2 dealing with processor shutdown for hard real-time systems can be applied: the solutions presented in this thesis are the only that can provide processor shutdown for this task set.

Table 8.5.: Task Set of the Aircraft Controller

Task	c [μs]	T [μs]	Jitter [μs]	d [μs]
1	150	800	0	800
2	2,277	200,000	0	5,000
3	420	40,000	8,890	15,000
4	552	20,000	10,685	20,000
5	496	20,000	9,885	20,000
6	1,423	25,000	0	12,000
7	3,096	50,000	0	50,000
8	7,880	59,000	0	59,000
9	1,996	50,000	15,786	100,000
10	3,220	100,000	34,358	100,000
11	3,220	100,000	55,558	100,000
12	520	200,000	0	100,000
13	1,120	200,000	107,210	200,000
14	954	1,000,000	141,521	200,000
15	1,124	200,000	0	200,000
16	3,345	200,000	0	200,000
17	1,990	1,000,000	0	1,000,000

8.1.3. Evaluation of Processor Slowdown

The main contribution to the state-of-the-art presented in this thesis regarding processor slowdown is the reduced computational complexity to calculate slowdown factors, and the ability to optimise the energy with task individual slowdown factors for tasks having (almost) arbitrary occurrence behaviour including arbitrary but fixed deadline assignments. Therefore, the purpose of this subsection is not a comparison to related work, but first to study the performance and accuracy trade-off introduced by the event stream model, and second to study the difference between global and local slowdown.

All slowdown methods yield factors to reduce the frequency for running the corresponding task instances such that the energy demand of the task set is reduced. Frequency reduction comes at the cost of increased worst-case execution times and thus less remaining idle time. Following this argumentation, the exploited idle time may be used as indicator for the performance of different slowdown methods. Although in Chapter 5 it was shown that remaining idle time is no indicator for power optimality, it is an indicator for errors introduced by approximation. Because early approximation over-estimates processing demand, approximation with few approximation steps will yield less optimal slowdown factors than an approximation with higher test indices.

Remaining idle time is computed by dividing the processing demand without approximation for a very long interval (at least of length equal to hyper-period) by the interval's length. Processing this step for each unoptimised and each optimised task set along a range of test indices will show the difference that approximation has caused. These tests are made in the first subsubsection.

The second sub-subsection shows a comparison of global and local slowdown in terms of achieved reduction of power consumption.

Evaluation of Error Occurred Through Approximation

Errors through approximation limit the available slack which leads to a decrease of the performance of methods for saving energy. Therefore, this comparison is done in terms of exploited idle time. Figures 8.2-8.6, pages 135-137, show the different results graphically, and Table 8.6 depicts results for the aircraft controller numerically.

Each figure depicts the relation of remaining idle time and test index for global as well as local slowdown. On the vertical axis, the relative amount of remaining idle time is depicted, and on the horizontal axis, the corresponding number of steps before linearisation takes place, the test index, is depicted. For comparison, a constant line shows the relative amount of idle time available: this is the idle time an unoptimised task set will produce. Boxes show the idle time that is left over by global slowdown, and the upper triangles do the same for local slowdown.

Figure 8.2 shows the results for the Palm-pilot example. Global and local slowdown show a remaining idle time of zero, regardless of the test index none is affected by any drawback due to approximation. The reason is that the Palm-pilot task set is strict periodic. If deadlines are equal to periods we obtain a feasible task set if

$$\sum_{\tau\in\Gamma} \frac{c_\tau}{T_\tau} =: \alpha \leq 1.$$

Thus, $\gamma := \frac{1}{\alpha}$ yields a feasible and the best slowdown factor for this kind of task sets.

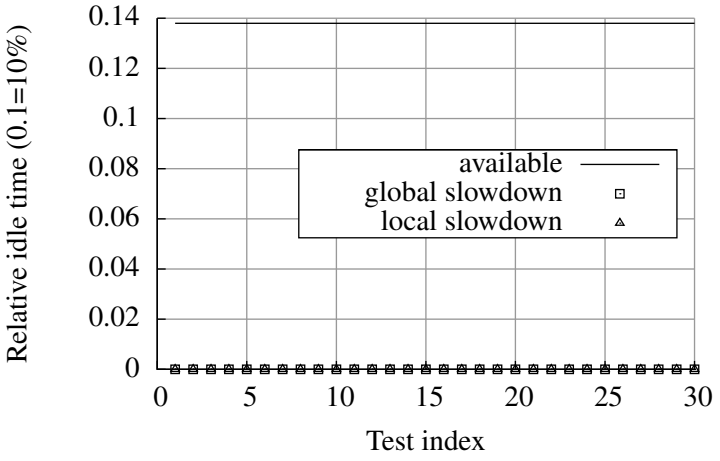


Figure 8.2.: Palm-Pilot Example, Test Index Versus Remaining Idle Time

Figure 8.3 shows the results for the first modification of the Palm-pilot example, one deadline was reduced. Approximation from the first element on (test index = 1) is significantly different from zero, but quite small compared to the available idle time. Approximation for test indices ranging from seven to 30 also leads to a non-zero remaining idle time, and does not show any

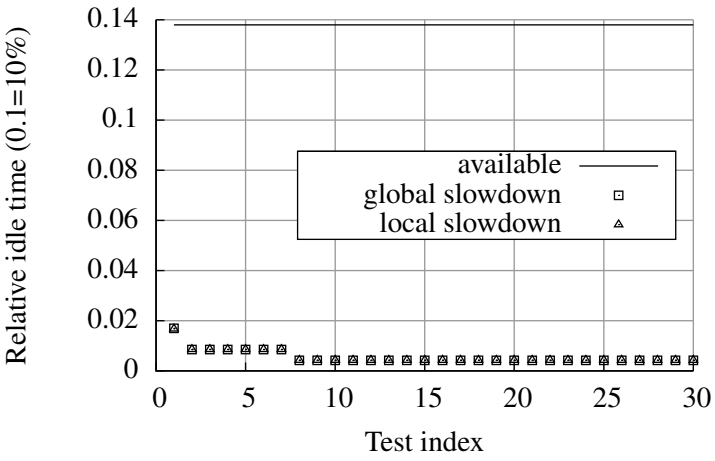


Figure 8.3.: Palm-Pilot Modification 1, Test Index Versus Remaining Idle Time

improvement for increasing approximation indices. Thus, the deadline reduction shows optimisation procedures to leave over a certain amount of available idle time, and it shows effects of over-approximation for test indices less than seven.

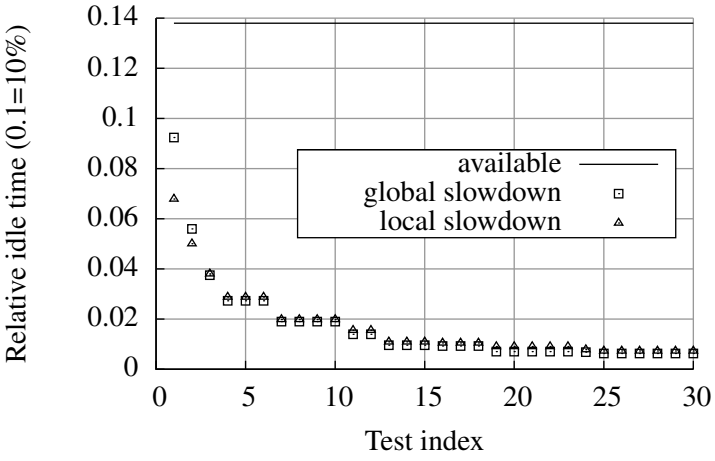


Figure 8.4.: Palm-pilot Modification 2, Test Index Versus Remaining Idle Time

Figure 8.4 shows the results for the second modification of the Palm-pilot example, where two deadlines were reduced. Significant deviations between global and local slowdown can be observed, and the figure shows, that approximation should start from an index not less than four because otherwise significant effects due to over-approximation show up. Again, the modification shows neither global nor local slowdown to reduce remaining idle time to zero, and the modification causes over-approximation effects up to a test index of 12, i.e. the values do not settle at a constant value before test index 13, and there are even slight steps again between indices 18 and 19, indices 23 and 24 (local slowdown), indices 24 and 25 (global slowdown).

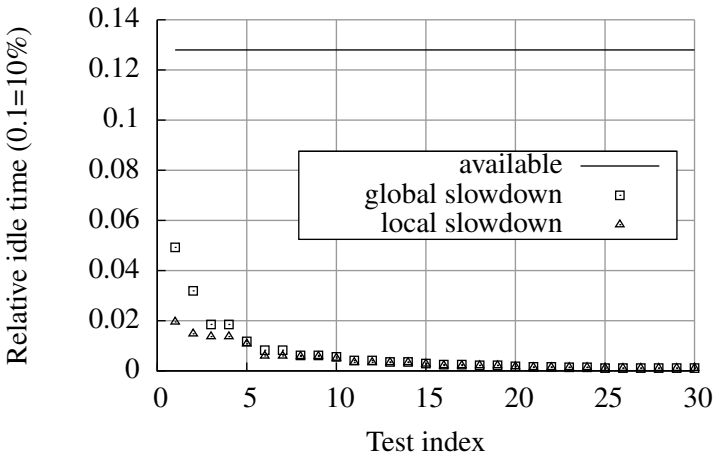


Figure 8.5.: Satellite Example, Test Index Versus Remaining Idle Time

The Olympus satellite example provides several deadlines that are shorter than the corresponding periods. Figure 8.5 shows the results for this example. The example shows over-approximation effects for test indices below eight; the values are close to zero for indices greater than eight.

The aircraft example contains deadlines lower and greater than periods, and jitter. Figure 8.6 depicts the results. Here, the approximation causes less errors than for the previous examples: values are close to zero for test indices 5 and above, and increasing the test index higher than 11 results in little improvement.

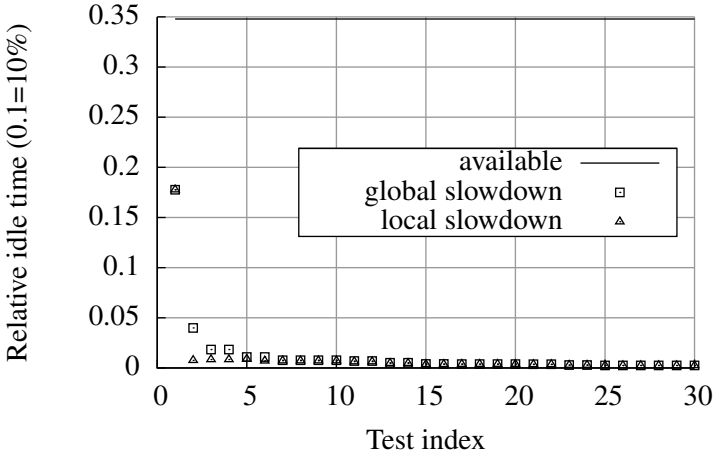


Figure 8.6.: Aircraft Controller, Test Index Versus Remaining Idle Time

It must be said that the scale of the vertical axis for the last example ranges from 0 to 0.35, twice the range than in the previous figures, and thus the graphic might suggest little error, although it is significant. Therefore, for the method of local slowdown, Table 8.6 shows the

(rounded) improvements for some test indices numerically. The second column depicts the exploited idle time by the slowdown method, and the third column depicts the ratio of exploited idle time by available idle time. The available idle time without any optimisations was 34.79 %. We observe: already with three test points per task, about 97% of the available idle time was exploited; and with 17 test points per task, about 99% was exploited. Compared with 17 test points per task, the version with 90 test points shows an improvement that is only less than 1 %. For the version with three test points, 42 different interval length to test (constraints for optimisation problem) were generated. For the version with 90 test points, these were 1370 constraints: this is a factor of 32 in complexity at 2.3% performance improvement.

Table 8.6.: Aircraft Controller, Performance Improvements of Local Slowdown Method

Test Index	Exploited idle time	Exploited by Available
1	17.02 %	48.91 %
3	33.96 %	97.61 %
7	34.07 %	97.92 %
17	34.42 %	98.94 %
50	34.65 %	99.59 %
90	34.72 %	99.78 %

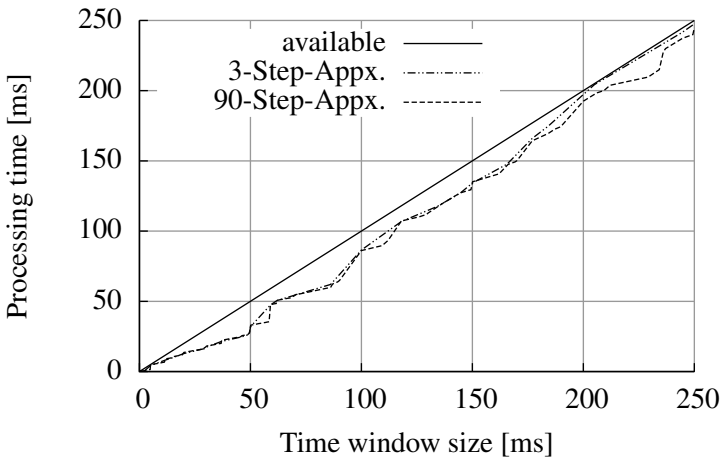


Figure 8.7.: Aircraft Controller, Demand Bound Function for Test Indices 3 and 90

Again for the aircraft task set, along time spans that range from 0 to 200 ms, Figure 8.7 shows the demand bound functions computed for approximation with 3 steps and approximation with 90 steps per task. Both curves show significant overlap up to a time span of 150 ms, for longer time spans, the version with three test points is closer to the curve depicting the available processing time, and at about 230 ms, the version with 90 test points closes up. As the Table 8.6 already indicated, there is only a small difference in between approximating all task demands with 3 test points per task or approximating with 90 test points per task. The reason for this can

only be that the long-term utilisation constraint turned out to be more restrictive than the demand bound constraints. This relation saves here 1300 constraints, and if we compare the version with 90 test points to the traditional hyper-period based approach, we save 78,000 constraints at no loss.

Although not an indicator for power optimality, remaining idle time was used to inspect errors due to approximation. All task sets showed low errors for low test indices (indices below 30), note that the hyper-period based approach contains a number of test points that is orders of magnitude greater, and, as shown, with no or little improvement.

Performance in Terms of Energy Minimisation

Chapter 5, Section 5.4, already included a comparison of two objectives for processor slowdown using a slowdown factor for each task. This was done with the help of the aircraft example task set of Table 8.5. Now, let us compare global and local slowdown along the Palm-pilot example and its second modification.

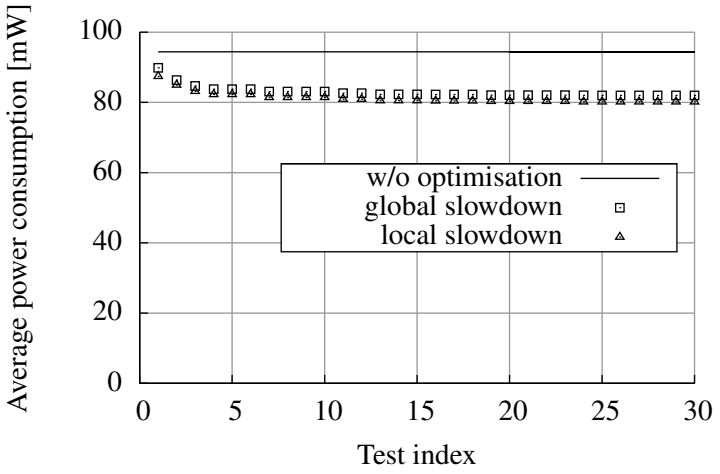


Figure 8.8.: Test Index Versus Average Power Consumption, Palm-Pilot Modification 2

The second modification of the Palm-pilot example showed significant errors due to approximation along remaining idle time. Figure 8.8 shows the achieved average power consumption for each approximation index. The constant line at 94.4 mW shows the average power consumption without any optimisation; the boxes show the achieved results of global slowdown; and the upper triangles show the results of local slowdown. As can be seen in Figure 8.4, page 136, the remaining idle time settles at a value above zero for a test index greater than 15, and here, already for test indices greater than five, the average power consumption does not improve, neither with global nor local slowdown: again, remaining idle time does not indicate power optimality.

The objective of global slowdown is to reduce the average power consumption by setting the processor frequency to a least possible value that is constant for all tasks; in contrast, local slowdown has the possibility to reduce the processor frequency for these tasks, which cause the most power consumption. The Palm-pilot example provides power consumption values in the

same order of magnitude, and as seen in Figure 8.8, even modification of deadlines does not lead to differences between local and global slowdown.

Let us now give one task after the other increased priority by setting its associated power consumption to 100 mW, while all other tasks have their power consumption set to 1 mW.

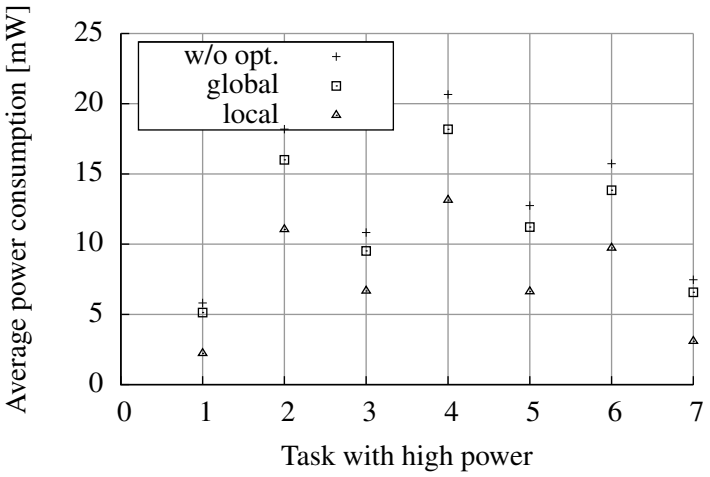


Figure 8.9.: High Power Task by Average Power Consumption, Palm-Pilot Modification 2

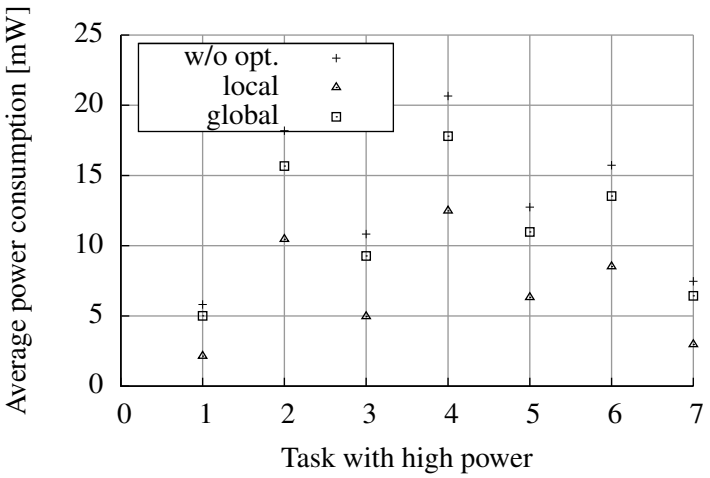


Figure 8.10.: High Power Tasks by Average Power Consumption, Original Palm-Pilot

Figure 8.9, shows the results for the second modification of the Palm-pilot task set. The results were computed using a test index of 10: the vertical axis shows the average power consumption, and the horizontal axis depicts the task which was given increased power consumption. Varying with the processing demand (fraction of worst-case execution time by period) that each task induces, the task set's power consumption varies. First, as expected, the optimisation with global slowdown shows little ability to compensate for the peaks in power consumption, second, there is

a significant gap between the result for local slowdown and the unoptimised result: optimisation with local slowdown reduces power consumption noticeably.

Let us see, if the deviations come from the power setting alone, or if they are affected by the modification with the short deadline. Figure 8.10 shows the result with the same modification of power consumption, but with the original deadlines of the Palm-pilot example. The Figure shows a slight difference for tasks 2 and 4, and moderate differences for tasks 3 and 6, the latter being the tasks, which were assigned short deadlines in the previous case. The effects that showed up regarding remaining idle time and approximation error are not perceptible regarding average power consumption.

It was shown, as was expected, local slowdown provides better compensation for peaks in power consumption than global slowdown. Peaks were tested that were orders of magnitude higher than the task set average.

8.1.4. Evaluation of Methods for Processor Shutdown

The main contribution to the state-of-the-art presented in this thesis regarding processor shutdown is the extension of application of low-power modes to task sets with (almost) arbitrary occurrence behaviour including arbitrary but fixed deadline assignments, and an offline optimisation resulting solely in a real-time clock as extra circuit that is required besides the processor. All works related require an extra circuit for monitoring the job set, or a circuit for computing re-optimisations.

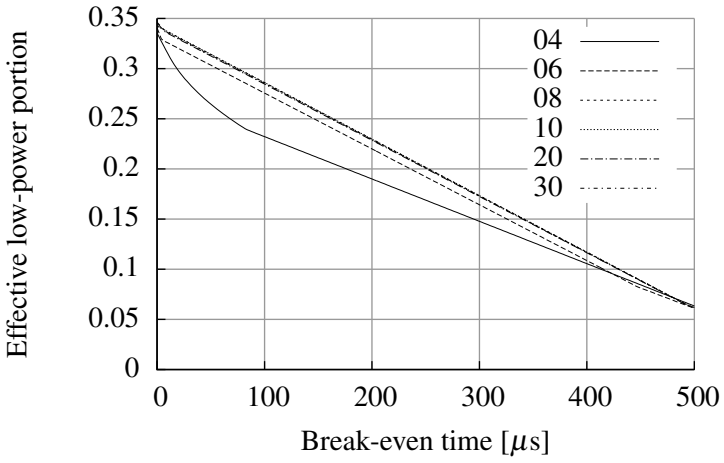
First, the error introduced by using an approximating real-time feasibility test are studied. The second sub-subsection first provides a comparison of the shutdown methods presented in this thesis to those of related work, in terms of efficiency, and second, the introduced task set examples are used to illustrate the performance in terms of energy efficiency of the offline optimisations that are presented here compared to ideal online optimisation.

Evaluation of the Influence of Different Approximation Steps

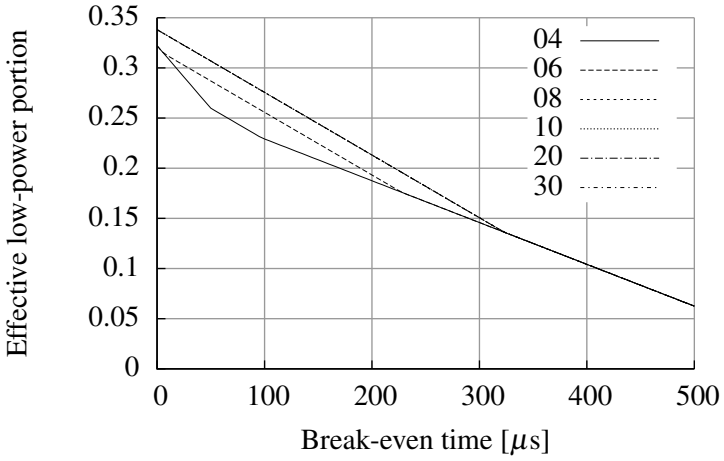
Optimisation algorithms for periodic shutdown and task-dependent shutdown apply an approximating real-time feasibility test with an adjustable number of approximation steps (test indices) for each task.

Figure 8.11, page 142, shows how varying test indices affect the efficiency of periodic and task-dependent shutdown. The test indices used are each 4, 6, 8, 10, 20, and 30; each is depicted by a single curve. It is observed, that task demand approximations show the same efficiency curves for either shutdown method. Approximations were made which compute the first 8 steps or more according to the event stream and which use an approximating slope for higher steps

Although, this is a single example, and for other examples the efficiency curves may not converge for the same test indices, the use of an approximation at test index 10 for each task may suffice for periodic and task-dependent shutdown.



(a) Periodic Shutdown



(b) Task-Dependent Shutdown

Figure 8.11.: Efficiency of Shutdown Methods Applied on the Aircraft Controller Task Set Using a Set of Various Test Indices

Performance of Shutdown Methods and Comparison to Related Work

The authors of [SCI2001] deny their solution to be applied to hard real-time systems ("blocking times"). The authors of [CG2005] present an extension of the former by voltage scaling (still with "blocking times"). And the authors of [LJ2000] made their algorithm prefer voltage scaling instead of processor shutdown. Therefore, all these works are not considered for a comparison.

The work presented in [LRK2003] (LC-EDF) provides a real-time feasible solution for saving energy using processor shutdown, and so does the work presented in [NQ2004]. The authors of [JPG2004] extend the work [LRK2003] by applying voltage scaling. Thus, for a comparison in terms of efficiency concerning energy saving for hard real-time systems, the algorithms introduced in [LRK2003] and [NQ2004] qualify.

Further, ideal shutdown is used for comparison as well: it is always applicable, though only of theoretical use. Two versions apply, one which shutdowns the processor whenever it is idle and the break-even time fits into the idle interval, and the other which procrastinates job executions to merge scattered idle intervals. The former is denoted in the figures by "Opt. w/o p." (optimal without procrastination), and the latter is named "Opt. w/ p." (optimal with procrastination). Both terms, 'optimal' and 'ideal', refer to situations where a completely beforehand computed schedule is used for optimisation: the creation times of all jobs are known to the ideal shutdown algorithms, and therefore each ideal shutdown method wakes up the processor in time.

Every online algorithm requires a schedule, and every online algorithm performs different for different schedules. Therefore, with the help of the information that event streams provide, schedules were generated: the schedules for each task set differ in the times of each task's first instantiation, the second instantiation and following instantiations are placed according to the corresponding event stream (varying offset plus constant event stream).

The resulting variations each online method produces are depicted by 90% bands, i.e. 5% of the computed efficiencies lie above the depicted band, 5% lie below, and the thickness of the band depicts the range of occurring variations.

Figures will show the performance of each method along a range of break-even times. The low-power modes of a single processor are found as points on the graph because each mode is associated one break-even time: if we compute an optimisation for each of many break-even times, then we compute an optimisation for each of many low-power modes.

For all figures, for the method presented in [NQ2004] no efficiency values were computed. This decision was made because the method was shown in [NQ2004] to perform better than LC-EDF: the efficiency of the method will always be between LC-EDF and the ideal shutdown with procrastination.

Palm-Pilot Task Set The first task set to be used for evaluation is the Palm-pilot task set because it is strict periodic, and of the related work, both chosen algorithms are applicable. Figure 8.12, page 144, shows the results for break-even times in the range from 0 ms to 10 ms, the task set has a utilisation of 86.2% leaving over 13.8% of the total time for application of processor shutdown. A test index of 10 was used for the real-time feasibility test to compute the results for periodic and task-dependent shutdown.

Ideal shutdown without procrastination ("Opt. w/o p.") and the LC-EDF method show noticeable variations: the bands intersect for break-even times greater than 7 ms. The curves for

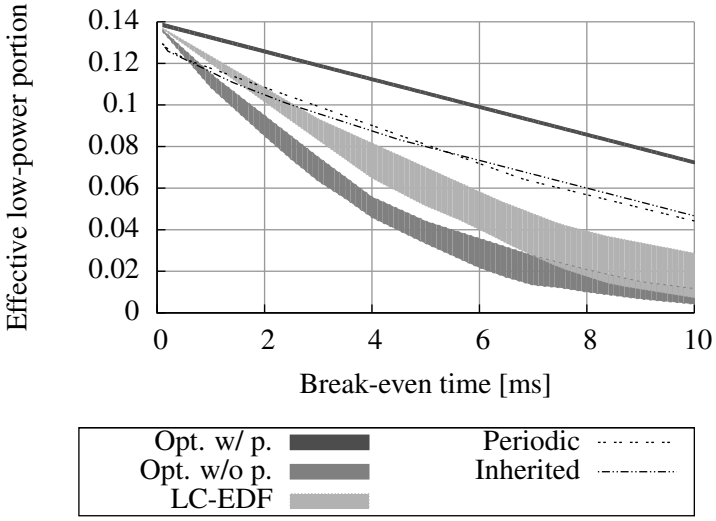


Figure 8.12.: Efficiency by Break-Even Time Diagram for Several Shutdown Methods, Task Set: Palm-Pilot in Original Form

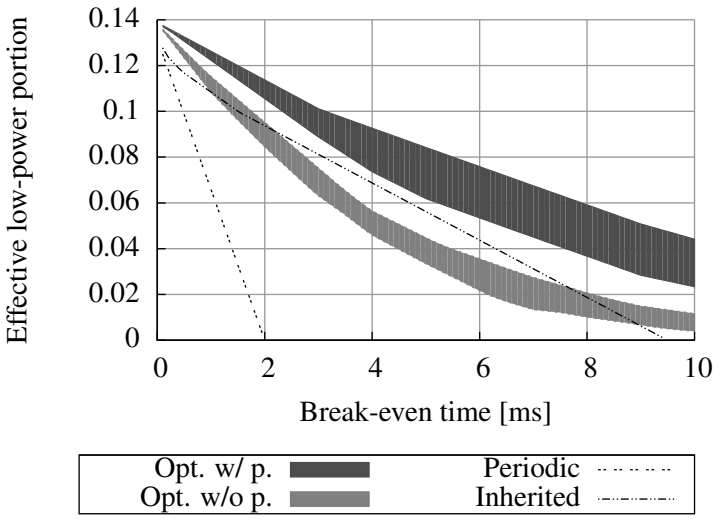
periodic shutdown ("Periodic") and task-dependent shutdown ("Inherited") lie close together; for break-even times less than 0.2 ms, both curves perform worse than the online methods; but for increasing break-even times, both curves close up; they intersect the LC-EDF curve at break-even time 2 ms; and for break-even times greater than 3 ms, the algorithms show better performance than the LC-EDF method.

Modifications of the Palm-Pilot Task Set Two modifications of the Palm-pilot task set were introduced. The modifications do not change the utilisation of the task set, but the introduced deadline modifications decrease the degrees of freedom for optimisations. The LC-EDF method is no longer applicable: the changes will not allow a positive delay because Equation (3.1), page 51, yields a value beyond 100%, see also Subsection 8.1.2.

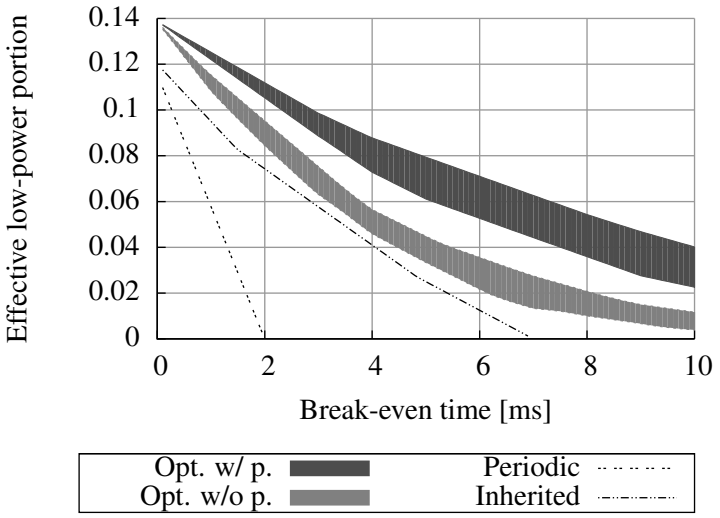
Figure 8.13, page 145, depicts the results for ideal shutdown, periodic shutdown, and task-dependent ("Inherited") shutdown. Ideal shutdown without procrastination was in either case not affected: it does not make use of deadlines because it does not procrastinate. Ideal shutdown with procrastination was affected in both cases: the shorter deadlines restrict the freedom to schedule jobs later than they are created.

Periodic shutdown allows no compensation for short deadlines, and as the figures show, it provides the worst performance of all methods. At least, it is unaffected by the deadline change introduced by the second task set modification.

Task-dependent shutdown inherits the occurrence behaviour of a certain task and avoids overlapping execution of low-power mode and jobs of the chosen task. The method can compensate for one short deadline: it shows better performance than ideal shutdown without procrastination for break-even times in the range of 2 ms up to 7.8 ms, and for the second modification, its optimisation results are still close to those obtained from ideal shutdown without procrastination.



(a) Modification 1



(b) Modification 2

Figure 8.13.: Efficiency by Break-Even Time Diagram for Periodic, Task-Dependent, and Ideal Shutdown, Experiments with Modified Palm-Pilot Task Sets

The Olympus Satellite Task Set An example with several task deadlines shorter than periods is the Olympus satellite attitude and orbital control system. It was introduced as a task set with the first task instantiations to be synchronous, except for tasks which were assigned an offset, and except for sporadic tasks. The task set specification with the synchronous job start at the time of system activation restricts the amount of schedule variations, now only the first instantiations of the sporadic tasks are varied.

Even without considering the sporadic tasks, the LC-EDF method is not applicable because

the short deadlines do not allow for a positive delay (see Subsection 8.1.2).

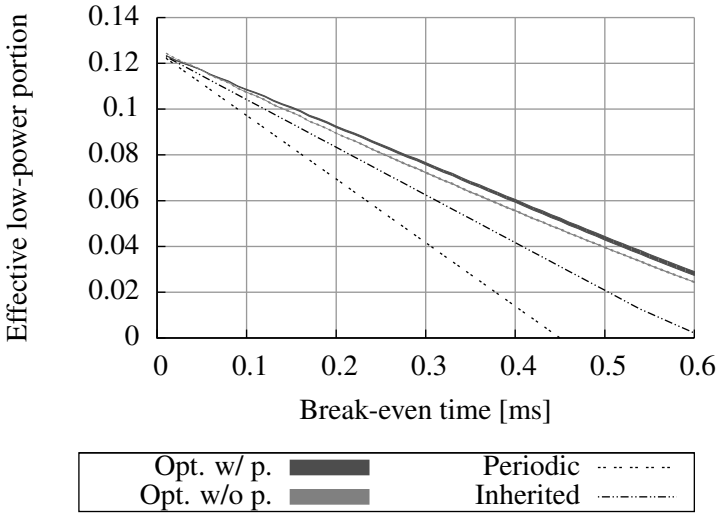


Figure 8.14.: Efficiency by Break-Even Time Diagram for Periodic, Task-Dependent, and Ideal Shutdown Methods applied on the Olympus Satellite Control Task Set, Used Test Index: 20

Figure 8.14, shows the results for ideal shutdown, periodic, and task-dependent shutdown. The latter two apply the real-time test with a test index of 20 for all tasks. As expected, both bands representing the results of ideal shutdown denote small variations: the bands are merely thicker lines in the figure. Procrastination does not show much improvement: both bands are close.

Periodic shutdown shows the same behaviour as for the Palm-pilot modifications: the short deadlines restrict its performance to be worse than the other methods. Task-dependent shutdown shows better performance, but cannot close up to ideal shutdown.

This task set is an example for the inter-task dependencies that were mentioned in Subsection 6.3.4: tasks have a specified instantiation delay compared to other tasks, and share occurrence behaviour. This task set motivates for an improved real-time feasibility test which incorporates and further develops the theory to handle never overlapping execution, which was presented in Subsection 6.3.1.

The Aircraft Controller Task Set Deadlines greater than task periods, deadlines shorter than task periods, and periods affected by jitter characterise the aircraft controller example. Even task periods distinguish in several orders of magnitude, showing a range from 800 μs to 1,000,000 μs.

This task set does not allow for the application of the LC-EDF method: the method would compute a negative delay. This task set does not allow for the application of the method presented in [NQ2004] either, because of deadlines greater than task periods. Only the shutdown methods presented in this thesis are applicable here.

Figure 8.15, shows the results. Periodic as well as task-dependent shutdown use the real-

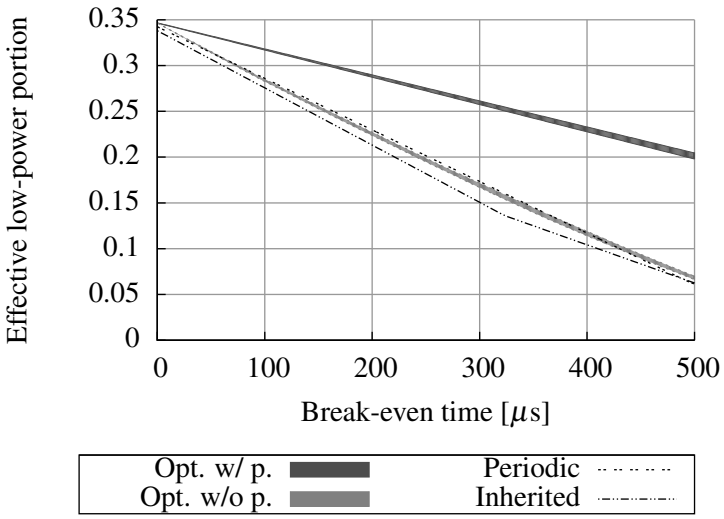


Figure 8.15.: Efficiency by Break-Even Time Diagram for Periodic, Task-Dependent, and Ideal Shutdown Methods applied on the Aircraft Controller Task Set

time test with a test index of 10 for all tasks. Both ideal shutdown methods do not show much variation: as in the example before, both bands are merely thicker lines. Both, periodic and task-dependent shutdown, result in similar efficiencies as ideal shutdown without procrastination, with periodic shutdown performing slightly better.

The reason task-dependent shutdown is likely to have a drawback was given in Subsection 6.3.3, by Figure 6.8, page 97: since task-dependent shutdown inherits the occurrence behaviour, it can only inherit periods that are present or non-fractional factorisations of the periods. The result is a restriction to a certain occurrence behaviour which limits the efficiency of the shutdown method.

8.2. Battery Modelling

Further battery measurements and the evaluation of the proposed model for predicting the operating life need to be performed: both are presented in this section.

First, the battery measurement device that was already used for the experiments shown in Chapter 7 is characterised. Second, battery measurements with other cell chemistries than lithium-manganese are reported. Finally one cell is chosen, and the proposed battery model is evaluated.

8.2.1. Measurement Device

“Do not trust measurements you have not done by yourself!” is a common saying. It is a motivation to document every information along a measurement, and especially to precisely protocol measurement conditions.

Battery discharge with pulses will suffer from error if pulses have long raise and fall times. Unsharp or 'round' pulses cause different power than a perfect rectangular shape, according to the power law $P = I^2R$: a thousand 'round' pulses will cause different power consumption compared to one pulse with a length of factor 1000. In consequence, pulsed discharge becomes difficult to compare with constant discharge, and at an error which is hardly estimated.

For the emulation of embedded systems, Thomas Weißmüller constructed a battery measurement device which is able to perform precisely the shapes of given current profiles, see [Wei2008]. This device is used to perform the measurements reported in this thesis.

Hardware The device is able to charge and discharge batteries with a nominal cell voltage ranging from 1 V to 14 V. It can charge and discharge a connected battery. Charge and discharge currents may be up to 4 A. The device contains a digital-to-analogue converter for setting the control input, which then is set and ensured by an analogue control loop, as digital control was shown to be too slow. Converter and control loop allow for setting pulses with durations of $1\,\mu$s: these short pulses can be set for a limited amount of time because a buffer will under-run, pulses of duration $50\,\mu$s do not suffer from this limitation.

Two analogue-to-digital converters are used to measure current and voltage of the battery, each with a sample rate of 1 million samples per second, and a bitwidth of 16. Data are stored on a memory card, data transfer to the card is at a speed of 2 MB per second, thus sample rates of 250 kHz are possible. Sample rates other than 1 million samples per second are performed by first adding samples, then dividing by the number of added values, and finally storing the average: every sample rate origins from 1 million measurements per second.

A temperature sensor placed at the battery's surface is read every two seconds, and temperature data are also stored on the memory card.

Software The device can be programmed with a measurement profile, which is placed on the memory card. Charge and discharge are programmable. The device supports for charging with constant current and/or constant voltage, and provides programmable termination conditions of the forms: current drops below given limit; temperature increase per minute is above given limit; or, voltage decrease per minute is above given limit ($-\Delta$V-method). This covers all recommended charge methods for commercially available batteries.

For discharge, Nils Schröder (see [Sch2009]) implemented a software solution that allows for programming different current profiles. Each profile represents one measurement, and the battery is charged according to a programmed charge method before each measurement.

A profile consists of a number of blocks, each block is repeated until its termination rule matches. Rules can be of the form: repeat the block a given number of times; repeat the block until battery temperature increases beyond a given limit; and repeat the block until battery voltage drops below a given limit.

Each block consists of a run-length encoded current profile, for example '1 A for 1 s then 0 A for 2 s then 500mA for 10 ms'. If a block is repeated, the device starts again a discharge with the first given current, then the second and so on.

The software stores measured voltage, measured current, and measured temperature per block each in an own file on the memory card. The software creates for each measurement and each

block an own folder, including the charge which has taken place before the measurement was started. Further, the software stores meta data for each measurement such as sample rate and date of measurement.

Raise and Fall Times for Pulsed Profiles An important advantage of the measurement device is its ability to discharge with sharp pulses. It is able to draw a discharge current in rectangular shape with only small error. With the next figures, using an ENERGIZER nickel-metal hydride battery of rated capacity 2.45 mAh and size R06 (AA), this ability is illustrated and quantified.

In Figure 8.16 four pulses are shown. The durations are 1 ms, 500 μs, 100 μs, and 50 μs with an amplitude of 3.675 A as control input. One graph shows the battery voltage, and the other shows the discharged current; both depict the actual (measured) values. The values were sampled with a rate of 250 kHz, i.e. one sample point every 4 μs. The figure shows an interval of length 75 ms, which was cut out of a longer series of pulses, which lasted about 10 seconds.

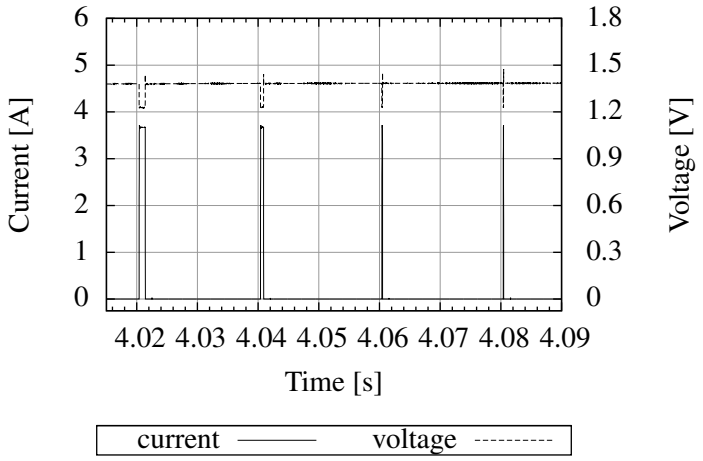


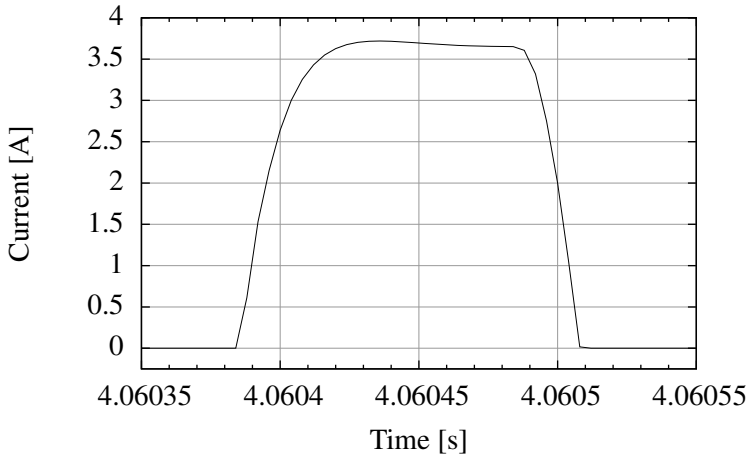
Figure 8.16.: Raise and Fall Times for Pulsed Discharge, Current $I = 3.675$ A, Pulse Durations 1 ms, 500 μs, 100 μs, 50 μs

Each of the pulses reaches the value given as control input. Before and after each pulse, the device switches between currents 3.675 A and 0 A without noticeable over- or undershoots, and in a time which's duration is shorter than can be observed in this scale.

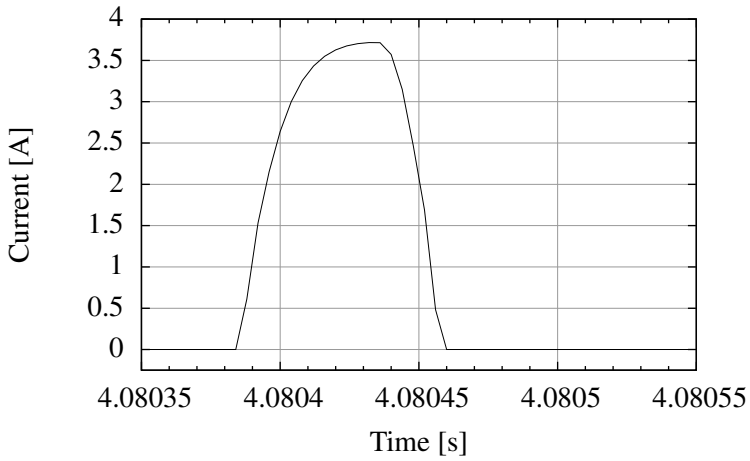
Note that the battery voltage drops only for the duration of the pulse and recovers immediately after the load reaches 0 A.

The last two pulses of Figure 8.16, which have durations 100 μs and 50 μs, are shown in another scale in Figure 8.17, page 150, both in a time window of length 200 μs.

The shapes of both raising edges match, the same for the falling edges: the raise times are equal, and so are the fall times. For the raising edge, the device needs about 15 μs to reach a current of 2.5 A, and further 10 μs to reach 3.5 A. For the falling edge, the device needs about 20 μs in total to reach 0 A. Both signal edges do not overshoot or undershoot control input: this emphasises the quality of the measurement device.



(a) Duration 100 μs and Amplitude 3.675 A



(b) Duration 50 μs and Amplitude 3.675 A

Figure 8.17.: Raise and Fall Times for Pulsed Discharge Current, a Closer Look

As noted earlier, the device qualifies for the emulation of embedded systems. At least if a DECT cordless phone or a GSM mobile phone are considered. The former produces sending pulses with a duration of about 460 μs, and the latter 690 μs, see also [Wei2008].

In the experiments performed to evaluate the assumptions concerning battery models, pulsed currents with pulse durations from 10 ms down to 50 μs are drawn. Assured by the figures, the device also qualifies for these experiments.

Low-Band Filtered Discharge-End Detection The measurement device measures the battery voltage always under load. Battery voltage is measured with the same sample rate as the current. To avoid early abortions of discharge because of measurement peaks, the device averages over the past 500 ms and compares this value to the value given in the block termination condition.

For a given profile that contains pulses with load and no load, the battery voltage for the periods of no load will be higher than the periods with load, increasing the average, which will cause the discharge to terminate later than desired. Shrinking the time window that is used to average the values will never completely compensate for the effect: simply because the step from no load to load requires a non-zero time to raise.

For a valid comparison of different discharge profiles, the termination condition must be the same over all measurements, i.e. same discharge current at same voltage. Ideally, a measurement device would provide a voltage measurement at a certain load, or to be neutral, voltage measurement at no load; and ideally, the device will not interrupt the specified discharge profile to perform the voltage measurement. Both ideas contradict, and the designer of the battery discharge device that was used for the measurements presented in this thesis has opted for measuring battery voltage under load.

8.2.2. Measurements of Batteries

Chapter 7 contained several measurements performed with a SONY lithium-manganese cell. The assumption, pulsed discharge performs better than constant discharge could be falsified, and it was shown that variation of pulse durations has no effect either. Clearly, all these measurements are only valid for the specific tested samples. It is to see if the results hold for other samples. This subsection presents measurements for three cells that are of different manufacturer, size, and chemistry.

To allow a direct comparison, the measurements are of the same kind as for the SONY lithium cell, presented in Table 7.3, page 109: constant discharge versus pulsed discharge with a pulse width of 10 ms versus pulsed discharge with a pulse width of $50\,\mu$s. Afterwords, Peukert-coefficients are computed for each cell.

Tests with an ENERGIZER NH15 Nickel-Metal Hydride Cell

The cell of type NH15 from manufacturer ENERGIZER is a standard nickel-metal hydride cell for usage in gadgets like portable digital cameras, or music players. The cell parameters are:

a) Size R06 (AA), i.e. a cylindrical cell of length and diameter of 50.5 mm x 14.5 mm.

b) Nominal cell voltage of 1.2 V.

c) Rated capacity of 2.450 mAh (8820 As) at the 5-hour discharge rate ($I = 490\,$mA).

d) Weight of 30 g.

e) Recommended cut-off voltage of 1.0 V.

The recommended charge method is: (1) charge with a constant current of 1.225 A; (2) stop charging when either the cell temperature increase exceeds 0.75°C per minute, or the cell voltage increase exceeds 10 mV per minute; (3) proceed with a topping charge lasting one hour at a current of 245 mA.

The tested cell was taken fresh out of the box, and discharged several times before the measurements were started.

Table 8.7.: Validating the Influence of Pulsed Discharge on the ENERGIZER NH15 Cell, Temperature Around 19°C, Start Date of Measurement: 2009-06-22, End Date: 06-24

No.	Block	Type	I [A]	t [s]	C [As]
1	a)	const.	2.446	3042	7440
	b)	const.	1.222	574	702
			Total	**3616**	**8142**
2	a)	const.	3.668	1732	6353
	b)	const.	1.834	838	1537
			Total	**2570**	**7890**
3	a)	$\phi = 10\,\text{ms}$	3.671	4290	7875
	b)	const.	1.834	8	15
			Total	**4298**	**7890**
4	a)	$\phi = 50\,\mu\text{s}$	3.686	4239	7812
	b)	const.	1.834	13	23
			Total	**4251**	**7835**
5	a)	const.	2.446	3011	7363
	b)	const.	1.222	594	726
			Total	**3604**	**8089**
6	a)	$\phi = 10\,\text{ms}$	2.447	6616	8094
	b)	const.	1.221	7	9
			Total	**6623**	**8103**
7	a)	$\phi = 50\,\mu\text{s}$	2.458	6602	8115
	b)	const.	1.221	8	9
			Total	**6610**	**8124**
8	a)	const.	1.834	4244	7784
	b)	const.	0.917	479	439
			Total	**4722**	**8223**
9	a)	$\phi = 10\,\text{ms}$	1.835	8974	8234
	b)	const.	0.916	8	7
			Total	**8981**	**8241**
10	a)	$\phi = 50\,\mu\text{s}$	1.848	8960	8276
	b)	const.	0.916	7	6
			Total	**8967**	**8284**

Before each measurement, the cell was charged according to the recommended charge method. All measurements were performed consecutively. The duration of the series was about 3 days. The ambient temperature was around 19°C.

The measurement series consists of: constant discharge at about 2.45 A, constant discharge at about 3.675 A, pulsed discharge at the same rate with a pulse width of $\phi = 10\,\text{ms}$, and a second pulse discharge with a pulse width of $\phi = 50\,\mu\text{s}$; this was repeated for currents $I = 2.45$ A, and $I = 1.838$ A.

Table 8.7, page 152, shows the measurement results. As in Chapter 7, it is I the current drawn (actual value, not control input), t the time discharge endured, and C the amount of electric energy discharged from the battery. For each measurement, the total of both blocks is depicted in bold letters.

First, measurements 3 and 4, 6 and 7, and 9 and 10 show the same discharged capacity: we cannot observe significant deviation if pulse width are varied, although the width was varied by a factor of 200.

Second, regarding measurements 3, 4, and 8 a), the data show no hints that pulsed discharge leads to more discharged capacity than constant discharge of half the amplitude. The relative error is 1-7784 As/7890 As= 1.3%, and 1-7784 As/7835 As = 0.7%.

Third, direct comparison of the total values for measurements 1, 5, 6, and 7 confirm the proposed law of Chapter 7. The same is true for measurements 2, 3, 4, and 8 a) (max. error of 1.3%), as well as 8, 9, and 10.

If we consider the a-blocks of measurements 1,2,5, and 8, averaging 1 a) and 5 a), we can compute the Peukert-coefficient of the cell. Let us use the original unrounded values and fit the data to the model, then we obtain $pc = 1.30$. The rounded results are depicted in Table 8.8; the last line denotes the standard deviation for each coefficient 1.29, 1.30, and 1.31.

Table 8.8.: Fitting the Peukert-Coefficient for the ENERGIZER Cell

No.	I [A]	t [s]	$I^{1.29} \cdot t$	$I^{1.30} \cdot t$	$I^{1.31} \cdot t$
1 a), 5 a)	2.446	3026	9594	9680	9767
2 a)	3.668	1732	9261	9382	9505
8 a)	1.834	4244	9280	9337	9394
Mean value			9378	9466	9555
Standard deviation			186.3	186.6	191.6

Note the high standard deviation, compared to coefficient computations for the SONY cell and the ULT cell.

Tests with two SANYO HR4U Nickel-Metal Hydride Cells

The cell of type HR4U from manufacturer SANYO is, as the ENERGIZER cell before, a standard nickel-metal hydride cell for usage in digital gadgets like portable digital cameras, music players, or cordless phones. The cell parameters are:

a) Size R03 (AAA), i.e. a cylindrical cell of length and diameter of 44.5 mm x 10.5 mm.

b) Nominal cell voltage of 1.2 V.

c) Rated capacity of 1.000 mAh (3600 As) at the 5-hour discharge rate ($I = 200\,\text{mA}$).

d) Weight of 13 g.

e) Recommended cut-off voltage of 1.0 V.

Table 8.9.: Validating the Influence of Pulsed Discharge on the SANYO HR4U Cell, Temperature Around 20°C, Start Date of Measurements: 2009-06-25, End Date: 06-26

No.	Block	Type	I [A]	t [s]	C [As]
1	a)	const.	0.997	2991	2981
	b)	const.	0.498	534	266
			Total	**3525**	**3248**
2	a)	const.	1.996	995	1987
	b)	const.	0.998	1018	1016
			Total	**2013**	**3003**
3	a)	$\phi = 10\,\mathrm{ms}$	1.997	3025	3021
	b)	const.	0.997	6	6
			Total	**3031**	**3027**
4	a)	$\phi = 50\,\mu\mathrm{s}$	2.010	3008	3022
	b)	const.	0.997	7	6
			Total	**3014**	**3029**
5	a)	const.	0.997	2986	2979
	b)	const.	0.499	563	281
			Total	**3549**	**3260**
6	a)	$\phi = 10\,\mathrm{ms}$	0.998	6599	3294
	b)	const.	0.498	4	3
			Total	**6603**	**3296**
7	a)	$\phi = 50\,\mu\mathrm{s}$	1.000	6594	3296
	b)	const.	0.498	4	2
			Total	**6598**	**3298**
8	a)	const.	0.498	6612	3296
	b)	const.	0.249	395	98
			Total	**7007**	**3394**
9	a)	$\phi = 10\,\mathrm{ms}$	0.499	13728	3426
	b)	const.	0.249	4	1
			Total	**13733**	**3427**
10	a)	$\phi = 50\,\mu\mathrm{s}$	0.500	13771	3444
	b)	const.	0.249	4	1
			Total	**13733**	**3445**

The used charge method was: (1) charge with a constant current of 1 A; (2) stop charging when either the cell temperature increase exceeds 1°C per minute, or the cell voltage increase exceeds 10 mV per minute; (3) finally, proceed with a topping charge lasting one hour at a current of 100 mA.

The tested cell was taken fresh out of the box, and discharged several times before the measurements were started.

Before each measurement, the cell was charged according to the recommended charge method. All measurements were performed consecutively, that is the cell did not rest after charge or discharge. The ambient temperature was around 20°C.

The measurement series consists of: constant discharge at about 1 A, constant discharge at about 2 A, followed by pulsed discharge at the same rate with a pulse width of $\phi = 10\,\text{ms}$, and a second pulse discharge with a pulse width of $\phi = 50\,\mu\text{s}$; this was repeated for currents $I = 1$ A, and $I = 0.5$ A.

Table 8.9, page 154, shows the measurement results. The data show the same behaviour as for the ENERGIZER cell.

First, measurements 3 and 4 show no variation in discharged capacity, although the width of the pulses was varied with a factor of 200. The same is true for measurements 6 and 7, and 9 and 10.

Second, pulsed discharge does not lead to noticeably more discharged capacity than constant discharge of half the amplitude. Regarding measurements 1a) and 3, we receive an error of 1-2981 As/3027 As = 1.5%, measurements 1a) and 4 yield an error of 1.6%, measurements 8 a), 6, and 7 yield no error.

Third, measurements 1 a), 2, 3, and 4 confirm the proposed law of Chapter 7, the same is true for measurements 1, 5, 6, and 7, as well as 8, 9, and 10.

Validation of the Proposed Law with Another SANYO HR4U Cell The charge method used for the measurements of Table 8.9 stresses the battery. Battery capacity starts degrading in further measurements, as can be seen in Appendix C, Table C.1 and C.1.

The manufacturer recommends: (1) charge for about 20 minutes with a low current (50 mA) to let battery terminal voltage raise above 1.15 V; (2) then charge with a constant current of 1 A; (3) and stop charging when either the cell temperature increase exceeds 1°Cper minute, or the cell voltage increase exceeds 10 mV per minute; (4) finally let the battery rest for about one hour.

Table 8.10, page 156, shows measurements made with the second cell sample of type HR4U. The measurements were taken consecutively, the cell recharged with the recommended charge method and a rest period of 30 minutes after charge.[1]

Three times (Measurements 2-6, 12-16, and 27-31), distributed over the measurement series, the capacity was measured for constant currents of I_{term}=2 A, 1.8 A, 1.2 A, 1 A, 0.5 A. And for the same termination currents, pre-discharges were made, discharging a defined real capacity (Measurements 7-11: $I = 2\,\text{A}$, 17-21: $I = 1\,\text{A}$, and 22-26: $I = 1.2\,\text{A}$).

Measurements 32 to 37 consist of pre-discharges down to cut-off potential followed by a termination current until cut-off potential is reached again.

The expected value for the capacity C_{exp} is the mean value of the three measurements for each constant current. The rest capacity C_{rest} is the capacity discharged by the termination current I_{term} after pre-discharge.

If we divide total capacity C_{total} by expected capacity C_{exp}, we observe a maximal error of 4.1%: this confirms the proposed-law.

[1] Thirty minutes are shorter than recommended, but the cell did not start to deteriorate.

Table 8.10.: Discharging a Battery of Type SANYO HR4U to 1 V Cut-Off Potential, with and without Pre-Discharge, Days of Measurement: 2009-07-14 and 2009-07-18, Temperature: 20°C

No.	Pre-Discharge	I_{term} [A]	C_{rest} [As]	C_{total} [As]	C_{exp} [As]	Error
1	none	0.9 A	-	3078		-
2	none	1.0 A	-	3033		-
3	none	2.0 A	-	2292		-
4	none	1.8 A	-	2363		-
5	none	1.2 A	-	2809		-
6	none	0.5 A	-	3132		-
7	2 A*600 s= 1200 As	2 A	1104	2301	2245	2.5%
8	2 A*600 s= 1200 As	1.8 A	1134	2331	2334	0.1%
9	2 A*600 s= 1200 As	1.2 A	1582	2780	2790	0.4%
10	2 A*600 s= 1200 As	1.0 A	1735	2933	2982	1.6%
11	2 A*600 s= 1200 As	0.5 A	1930	3128	3132	0.1%
12	none	1.0 A	-	2970		-
13	none	2.0 A	-	2232		-
14	none	1.8 A	-	2333		-
15	none	1.2 A	-	2785		-
16	none	0.5 A	-	3134		-
17	1 A*1200 s= 1200 As	2 A	1049	2246	2245	0%
18	1 A*1200 s= 1200 As	1.8 A	1107	2304	2334	1.3%
19	1 A*1200 s= 1200 As	1.2 A	1569	2766	2790	0.9%
20	1 A*1200 s= 1200 As	1.0 A	1719	2916	2982	2.2%
21	1 A*1200 s= 1200 As	0.5 A	1929	3126	3132	0.2%
22	1,2 A*1000 s= 1200 As	2 A	1059	2257	2245	0.5%
23	1,2 A*1000 s= 1200 As	1.8 A	1117	2315	2334	0.8%
24	1,2 A*1000 s= 1200 As	1.2 A	1579	2777	2790	0.5%
25	1,2 A*1000 s= 1200 As	1.0 A	1710	2908	2982	2.5%
26	1,2 A*1000 s= 1200 As	0.5 A	1941	3139	3132	0.2%
27	none	1.0 A	-	2942		-
28	none	2.0 A	-	2211		-
29	none	1.8 A	-	2306		-
30	none	1.2 A	-	2775		-
31	none	0.5 A	-	3131		-
32	2 A to $V_{cut\text{-}off}$= 2176 As	1.0 A	743	2918	2982	0.2%
33	2 A to $V_{cut\text{-}off}$= 2193 As	1.2 A	608	2801	2790	2.1%
34	2 A to $V_{cut\text{-}off}$= 2179 As	1.8 A	166	2345	2334	0.5%
35	1.8 A to $V_{cut\text{-}off}$= 2272 As	1.0 A	597	2869	2982	3.8%
36	1.8 A to $V_{cut\text{-}off}$= 2322 As	1.2 A	429	2751	2790	1.4%
37	1.2 A to $V_{cut\text{-}off}$= 2756 As	1.0 A	103	2861	2982	4.1%

Note, if defined pre-discharge is followed by discharge to cut-off potential with the same current, even then, we observe an error of 2.5% for $I = 2\,\mathrm{A}$ (Measurement 7), 0.9% for $I = 1.2\,\mathrm{A}$ (Measurement 20), and 2.5% for $I = 1\,\mathrm{A}$ (Measurement 25).

Computation with the capacity means for currents $I = 0.5\,\mathrm{A}, 1\,\mathrm{A}, 1.2\,\mathrm{A}, 1.8\,\mathrm{A}, 2\,\mathrm{A}$ results in a Peukert-coefficient of pc $= 1.209$ and a normalised capacity of $C_{\mathrm{norm}} = 2811\,\mathrm{As}$, with a standard deviation of 139 As. The standard deviation is again very high compared to values of 10 As for the SONY US 18650 lithium manganese cell and 7 As for the ULT 18650 FP lithium-iron-phosphate cell.

Tests with a ULT 18650 FP $LiFePO_4$ Secondary Battery

The cell of type ULT 18650 FP consists besides lithium also of iron and phosphate. It is to be used for example in laptop computers, or remote controlled vehicles, e.g. model aircraft. The cell parameters are:

a) Cylindrical, has a length and diameter of 65.0 mm x 18 mm.

b) Nominal cell voltage of 3.2 V.

c) Rated capacity of 1.35 mAh (4860 As) at the 5-hour discharge rate ($I = 270\,\mathrm{mA}$).

d) Weight of 82 g.

e) Recommended cut-off voltage of 2.5 V.

The recommended charge method is: (1) charge with a constant current of 1.25 A (25/27 of rated capacity), not to exceed a voltage of 3.65 V; (2) and stop charging when the current drops below 0.67 A (1/20 of rated capacity).

The cell was never in use before the first measurement, therefore, the first measurement is to be neglected.

Before each measurement, the cell was charged according to the recommended charge method. All measurements were performed consecutively. The duration of the series was about 2 days. The ambient temperature was around 20.5°C.

The measurement series consists of: three times a constant discharge at about 1.35 A because the cell was never in use; and thereafter constant discharge at about 2.7 A; followed by pulsed discharge at the same rate with a pulse width of $\phi = 10\,\mathrm{ms}$; and followed by a second pulse discharge with a pulse width of $\phi = 50\,\mu\mathrm{s}$; the last three steps were repeated for currents $I =$ 1.35 A, and $I = 0.675$ A.

Table 8.11 shows the measurement results. Also this data confirm the observations made hitherto. Capacity remains constant, regardless of the duration of the pulses before constant discharge. We already saw the proposed law regarding discharge-end detection and remaining capacity confirmed by several measurements in Chapter 7.

If we consider measurements 2,3, and the a-blocks of measurements 4,7, and 10 and if we average the times for current $I = 1.347\,\mathrm{A}$, then we can compute the Peukert-coefficient of the cell. Let us use the original unrounded values and fit the data to the model, then we obtain $pc = 1.13$. The rounded results are depicted in Table 8.12; the last line denotes the standard deviation for each tested coefficient 1.12, 1.13, 1.14.

Table 8.11.: Validating the Influence of Pulsed Discharge on a ULT $LiFePO_4$ Cell, Temperature of 20.5°C, Start Date: 2009-06-30, End Date 07-01

No.	Block	Type	I [A]	t [s]	C [As]
1		const.	1.347	2296	3093
2		const.	1.347	2232	3007
3		const.	1.347	2209	2976
4	a)	const.	2.695	1007	2714
	b)	const.	1.347	292	393
			Total	**1299**	**3107**
5	a)	$\phi = 10\,\mathrm{ms}$	2.696	2221	2294
	b)	const.	1.347	8	11
			Total	**2229**	**3005**
6	a)	$\phi = 50\,\mu\mathrm{s}$	2.697	2206	2975
	b)	const.	1.347	9	13
			Total	**2216**	**2988**
7	a)	const.	1.347	2188	2947
	b)	const.	0.674	422	284
			Total	**2610**	**3231**
8	a)	$\phi = 10\,\mathrm{ms}$	1.348	4822	3249
	b)	const.	0.674	8	5
			Total	**4830**	**3255**
9	a)	$\phi = 50\,\mu\mathrm{s}$	1.348	4821	3249
	b)	const.	0.674	8	6
			Total	**4830**	**3255**
10	a)	const.	0.674	4810	3240
	b)	const.	0.337	445	150
			Total	**5254**	**3390**
11	a)	$\phi = 10\,\mathrm{ms}$	0.674	10257	3458
	b)	const.	0.337	7	2
			Total	**10264**	**3460**
12	a)	$\phi = 50\,\mu\mathrm{s}$	0.675	10274	3467
	b)	const.	0.337	6	2
			Total	**10280**	**3469**

Table 8.12.: Fitting the Peukert-Coefficient for the ULT 18650 FP Cell

No.	I [A]	t [s]	$I^{1.12} \cdot t$	$I^{1.13} \cdot t$	$I^{1.14} \cdot t$
2, 3, 7 a)	1.347	2209	3085	3094	3103
4 a)	2.695	1007	3057	3087	3118
10 a)	0.674	4810	3092	3080	3068
Mean value			3083	3090	3100
Standard deviation			18.64	7.04	25.80

8.3. Conclusion

Methods for saving energy and assumptions regarding the performance of batteries have been evaluated. Setups were described that show the flexibility of used software and hardware implementations.

A flexible implementation was created to evaluate the energy optimisations with processor slowdown and processor shutdown. The task sets that were used range from strict periodic, and periodic with deadlines less than periods, to periodic task sets with jitter and deadlines greater than periods. Three of five task sets were taken from the literature. Two task sets are modifications to illustrate the effect of short deadlines on the efficiency of the optimisation methods.

The device for battery measurements was introduced as a flexible solution for verifying assumptions regarding battery capacity. It guarantees sharp pulses, is able to measure and record voltage and current with high sample rates, and protocols the battery's surface temperature. The battery samples represent cells that are commonly used in mobile embedded devices.

Experimentation is quite time intensive: computation of the results for ideal shutdown or LC-EDF shutdown took about a few days per task set[2], and battery measurements were made over a time span of about two months[3].

8.3.1. Methods for Saving Energy

For processor slowdown, errors caused due to the approximated feasibility test were inspected and found to be low. The error was found to be significant for approximation with only one approximation step. The differences of achieved results for approximation with 10 steps or more were found to be negligible.

The expected difference between global and local processor slowdown was observed: if the task set contains peaks in power consumption, local slowdown shows significant improvement of power dissipation compared global slowdown.

For processor shutdown, also errors caused due to the approximated feasibility test were inspected and found to be high for approximations with a small number of steps. And the errors were found to be negligible for approximations with 10 steps or more.

Task-dependent and periodic shutdown were compared to LC-EDF shutdown method, the

[2]Hardware used: PC with Intel Dual Core CPU, 2GB RAM

[3]Including only those month used for the presented results, excluding time for testing.

latter to perform worse for high break-even times. Both, task-dependent and periodic shutdown, were shown to achieve efficiencies close to those by online methods – although both are offline methods and cannot take advantage of knowledge that is only known at run-time.

Short deadlines were shown to reduce the efficiency of the shutdown methods presented in this thesis. A possible improvement is to be made by incorporating the theory of non-overlapping job execution which is depicted in Subsection 6.3.1.

8.3.2. Battery Model

Experiments with battery samples falsified the assumption that pulsed discharge performs better than constant discharge at half amplitude: experiments show no improvement for pulses in the range of 10 ms to 50 μs, which are standard cases for today embedded systems like GSM or DECT telephones. The assumption was found to be false for the tested lithium-manganese cell sample (this was found in Chapter 7), the lithium-iron-phosphate sample, and the nickel-metal hydride cell samples.

Experiments with battery samples falsified the assumption that pulse width does matter in terms of operating life, this was tested for 10 ms to 50 us pulses. Mind that a frame width of 10 ms is used by the DECT standard for time-division-multiple-access (TDMA) communication, with a slot width of 460 us for sending and receiving slots, and mind that the a communication slot for the GSM standard has a pulse width of 690 us, further note that reference [Wei2008] also falsified the assumption using a size AAA nickel-metal hydride cell and asymmetrical pulses of width 7.2 s for high load and 29.8 s for low load, as well as the ratio 50 ms:207 ms. In summary, capacity results were the same for all pulse widths and all battery samples used.

The law proposing that the termination condition of the discharge solely determines the real capacity was found to hold. Extensive measurements for the SANYO HR4U nickel-metal hydride cell revealed errors of no more than 4.1%.

Measurement results were consistent, although, there were significant variations, when the measurements with nickel-metal hydride batteries were repeated, and the computation of Peukert-coefficients for these cells have a high standard variation.

9. Conclusion

The work presented in this thesis extends the state-of-the-art regarding the application of processing-timeconsuming energy optimisation of hard real-time task sets form periodic to non-periodic earliest deadline first scheduled task systems having arbitrary but fixed deadlines. Properties influencing battery operating life were studied, especially regarding the shape of the discharge current, and a simple battery model for today's customarily available batteries was found.

9.1. Summary

The used approximating real-time feasibility test provided low errors even for an early starting approximation. Examples show diminishing errors for an increasing number of steps. Approximation with 10 steps per task provides a performance which is only slightly improved by using more approximation steps. For a step count of 10, the complexity of the test is only 10 times the number of tasks.

The application of processor slowdown, also known as dynamic voltage scaling, or dynamic voltage and frequency scaling, can be done in several granularities regarding the partition of software. If the software is partitioned into independent code fragments – tasks – that when instantiated represent jobs to be scheduled, then the presented work enables the application of processor slowdown per task (i.e. local) or per task set (i.e. global). This applies regardless whether tasks are instantiated periodic, periodic with jitter, sporadic, or in another way, as long as this is described by an event stream, and regardless of the tasks' deadlines as long as these are positive and do not change during run-time.

For processor slowdown, experiments show that the amount of long-term idle time not exploited by slowdown, either per task or per task set, is no indicator for a solution to be power optimal.

The optimisation problems that determine slowdown factors per task or per task set were integrated into the real-time feasibility test. The test allows for a set of linear optimisation constraints. The problem to optimise with a common slowdown factor was shown to be solvable with a linear objective, whereas to optimise with a slowdown factor individually for each task was shown to require a non-linear but convex objective. That is we have one linear optimisation problem and one convex optimisation problem, thus both will provide the optimal solution.

Additionally, the integration of the real-time feasibility test allows for a complexity of the constraints that is linear in the product of the number of tasks and the number of approximation steps per task.

The application of processor shutdown in case of a hard real-time system requires to integrate the cost for switching the processor mode into the optimisation algorithms. Duration of stay in a low-power mode and rate of mode switching determine the efficiency of a solution. In

this thesis, two methods were presented: periodic shutdown which introduces a periodically instantiated low-power task, and task-dependent shutdown which introduces a low-power task that instantiates in dependence of another task which was already present in the task set.

Both methods optimise for duration of stay in low-power mode and rate of mode switching. Both methods allow for the other tasks to instantiate arbitrary as long as this is described by an event stream, and both methods allow for task deadlines to be arbitrary positive and fixed for run-time.

Both methods are offline optimisations and require only an external timer as extra hardware besides the processor.

For processor shutdown, experiments show that presented methods perform worse than online methods for low break-even times, but the results become better and finally supersede the results of the online methods for long break-even times. Additionally, although presented methods optimise offline, their results close to the results of ideal online methods.

Since every presented energy optimisation method is computed offline, the methods allow for an early estimation of power consumption, early at design time, after processor and task set have been specified.

Regarding the battery model, several assumptions were falsified by experiments with common batteries. Lithium-based as well as nickel-based batteries show that pulsed discharge with equidistant pulses does not perform better than constant discharge of half the amplitude, and pulse duration variation does not matter either. Both were shown for pulse durations in the range of 50 μs to 10 ms, pulses occurring in modern radio applications like the DECT or GSM communication standards.

The battery model found is simple. Because of the measurements made, a detailed knowledge of a time dependent discharge curve ranging over one complete discharge cycle could be omitted. The model presents a closed formula to predict the battery's operating life. The model does not incorporate changes of ambient temperature during run-time, but allows for an operating life prediction for a given constant temperature. A discharge profile denotes for each current that is drawn during run-time the relative amount of time it is drawn. Given the discharge current at end of discharge and the system's discharge profile, the model computes the resulting operating life.

As for processor shutdown, the battery model requires small hardware: an energy counter and a table that contains a relation of discharge currents to operating life.

In the very brief

The presented work assumes an earliest deadline first scheduled set of independent tasks which are assigned to one processor supplied by one battery. Each task of the task set is supposed to be described by an event stream and is allowed to have an arbitrary but fixed deadline. The processor is supposed to be voltage scaling enabled and has one or more low-power modes.

This work is the first to present:

a) Voltage scaling per task and per task set applied to non-periodic task sets with arbitrary but fixed deadlines, including a reduced complexity by integrating the real-time feasibility test into the optimisation problems.

b) Processor shutdown applied to non-periodic task sets with arbitrary but fixed deadlines, and with minimal hardware requirements compared to related work: here just a timer is required.

c) Offline optimisation of processor slowdown and shutdown that allows for predicting the energy demand already at design time, and without computational intensive simulation.

d) A battery model in a closed form allowing for predicting operating life out of power and timing description of task set and processor. The model is evaluated by experiments with common batteries and loads of today's common embedded systems.

9.2. Outlook

Several questions were answered in this thesis, some questions that are left open are listed in the following:

The presented optimisations for energy saving use the same real-time feasibility test. The optimisations are subject to the constraints on the processor demand which the test provides. All optimisations tighten the constraints: for one or more interval sizes, the idle time is reduced to zero. The question arises, if processor slowdown that is per task (i.e. local) can further improve the optimisation solutions of periodic and task-dependent shutdown. The former is a likely candidate, because it produces an exploitation of idle time that is similar to global processor slowdown.

Each of the presented optimisation methods produces a task set as output, and the information is with similar semantic as the input. Thus, the resulting task set of one optimisation method may serve as input for the next optimisation method; methods may be combined.

Nevertheless, only a representative set of real cases will show if a combination is useful.

Regarding processor shutdown, presented methods show a drawback for task sets with short deadlines, and they cannot make use of offset information given in the task set specification. Both problems may be solved or tackled with the theory of two always sequentially executed (never overlapping) jobs (Subsection 6.3.1, page 89). An extension of the real-time feasibility test employing this theory may provide a solution.

Regarding a battery's operating life, the shown concept requires a given setup of task set and processor that provides timing and power information. The result is a choice for a method to save energy and its configuration. The optimisation is done offline and allows for adjusting precision. This setting may integrate well into system design tools, especially in combination with worst-case execution time estimators to foster early design decisions.

The presented battery model is simple and has been evaluated. Measurements show a maximal error of 6.7% for the tested lithium-iron-phosphate cell and 4.1% for the tested nickel-metal-hydride cell, but significant deviations may arise for battery types that are not studied, for example prismatic cells (studied geometry: round cells). Other types of power supply like fuel cells were not considered either and require an evaluation whether they can be included into the model or require a modification. For each not studied type of battery, extensive measurements should be made to validate the proposed-law.

Temperature was assumed to be known and constant during operation. Non-constant temperature is to be studied: this requires a thermally controlled lab and detailed knowledge on battery chemistry.

However, the measurements presented in this thesis already consumed about two months consecutive measuring, not including measurements for testing and adjusting the device.

The presented concept is designed for earliest deadline first scheduled task sets. Since the used real-time feasibility test was already shown to be able to test preemptive task sets with fixed priorities, the work presented in this thesis may be extended by future work for these systems, or even for non-preemptive task systems.

A. Efficiency of DC-DC-Converters

The following tables denote how efficient commercial DC-DC converters work. The electric load drawn from the battery is measured against the electric current that flows solely to drive the converter: the fraction of load by driving current denotes the converter's efficiency. Efficiency might come to an issue in cases where voltage scaling or processor shutdown is applied, because in these situations the efficiency changes with the voltage, and the decreased efficiency causes parts of the energy saving to be lost.

Figure A.1 is a sketch of the designated circuit. The dc-dc-converter is in series with load and battery, though it is also in parallel to the battery. The current I_{dc} depicts the converters power dissipation: it is the current that the converter draws to convert the voltage; the current is in parallel to the load. The conversion efficiency is defined by $1 - I_{\mathrm{dc}}/I_{\mathrm{load}}$. Thus a converter with a conversion efficiency of 80% draws the current

$$I_{\mathrm{bat}} = I_{\mathrm{dc}} + I_{\mathrm{load}} = (1 - 0.8) \cdot I_{\mathrm{load}} + I_{\mathrm{load}} = 1.2 \cdot I_{\mathrm{load}}$$

from the battery.

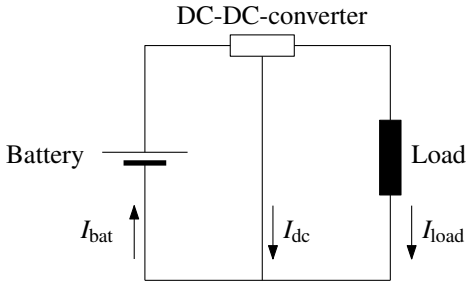


Figure A.1.: DC-DC-Converter in Circuit

The tables list the efficiency directly for switching converters and indirectly, as ground current (I_{dc}), for linear converters. The ground current is drawn from source to ground by the converter's control circuit. All listed regulators are designed for the use in PDAs, Palmtops, Wireless Handhelds, Cell Phones etc.

Model	Voltage/Max. Current	I_{dc}	
ADP 1706/7/8	$0.75-3.3\,V$	$90\,\mu A$	$(V_{in}=3.8\,V, I_{load}=1\,mA)$,
	$1\,A$	$110\,\mu A$	$(V_{in}=3.8\,V, I_{load}=10\,mA)$,
		$1.3\,mA$	$(V_{in}=3.8\,V, I_{load}=1\,A)$
ADP 1710/11	$0.75-3.3\,V$	$50\,\mu A$	$(V_{in}=3.8\,V, I_{load}=1\,mA)$,
	$150\,mA$	$110\,\mu A$	$(V_{in}=3.8\,V, I_{load}=10\,mA)$,
		$0.95\,mA$	$(V_{in}=3.8\,V, I_{load}=150\,mA)$
ADP	$0.75-3.3\,V$	$90\,\mu A$	$(V_{in}=3.8\,V, I_{load}=1\,mA)$,
1712/13/14	$300\,mA$	$110\,\mu A$	$(V_{in}=3.8\,V, I_{load}=10\,mA)$,
		$0.35\,mA$	$(V_{in}=3.8\,V, I_{load}=300\,mA)$
ADP 1715/16	$0.75-3.3\,V$	$80\,mA$	$(V_{in}=3.8\,V, I_{load}=1\,mA)$,
	$500\,mA$	$100\,mA$	$(V_{in}=3.8\,V, I_{load}=10\,mA)$,
		$0.4\,mA$	$(V_{in}=3.8\,V, I_{load}=500\,mA)$
MAX8516,	$0.5\,V-3.4\,V$	$340\,\mu A$	$(V_{in}=1.8\,V,$
8517, 8518	$1\,A$		$V_{out}=1.5\,V$
			$I_{load}=10\,mA-1\,A)$
MAX8869	$0.8-5\,V$	$0.5\,mA$	$(V_{out}=2.5\,V, V_{in}=3\,V, 5\,V,$
	$1\,A$		$I_{load}=10\,mA)$,
		$2.5\,mA$	$(V_{out}=2.5\,V, V_{in}=3\,V,$
			$I_{load}=1\,A)$,
		$3.2\,mA$	$(V_{out}=2.5\,V, V_{in}=5\,V)$
1175fd	$1\,V-23\,V$	$1.5\,mA$	$(V_{out}=V_{in}-2V, I_{load}=100\,mA)$,
	$500\,mA$	$4\,mA$	$(V_{out}=V_{in}-2V, I_{load}=500\,mA)$
1964f	$1.2\,V-20\,V$	$0.4\,mA$	$(V_{out}=1.22\,V, V_{in}=1.8\,V-10\,V,$
	$200\,mA$		$I_{load}=10\,mA)$,
		$2.6\,mA$	$(V_{out}=1.22\,V, V_{in}=1.8\,V,$
			$I_{load}=200\,mA)$,
		$2.2\,mA$	$(V_{out}=1.22\,V, V_{in}=10\,V,$
			$I_{load}=200\,mA)$

Table A.1.: Efficiency of some Linear DC-DC-Converters, Manufactured by Analog Devices (Models AD...), MAXIM Dallas (Models MAX...), and Linear Technology (Models 1...)

Model	Voltages / I_{max}	Max. Eff. @$V_{out} = 1.2\,\mathrm{V}$		Eff. @$1.2\,\mathrm{V}, I = 1\,\mathrm{mA}$	Eff. @$V_{out} = 1.2\,\mathrm{V}, I_{max}$	
MAX 1702B	$0.7\,\mathrm{V} - V_{in}$ 400 mA	90%	($V_{in} = 2.6$ −5.5 V, $V_{out} = 1.3\,\mathrm{V}$ $I_{load} = 200\,\mathrm{mA}$)	61%	($V_{in} = 2.6\,\mathrm{V}$ −5.5 V)	88%	($V_{in} = 2.6\,\mathrm{V}$ −5.5 V)
MAX 1820, MAX 1821X	$0.4\,\mathrm{V} - 3.4\,\mathrm{V}$ 600 mA	95%	($V_{in} = 2.7\,\mathrm{V}$ $V_{out} = 1.5\,\mathrm{V}$ $I_{load} = 300\,\mathrm{mA}$ −400mA)	9% 14%	($V_{in} = 3.6\,\mathrm{V}$, $V_{out} = 1.5\,\mathrm{V}$), ($V_{in} = 2.7\,\mathrm{V}$, $V_{out} = 1.5\,\mathrm{V}$)	90%	($V_{in} = 2.7\,\mathrm{V}$ −5 V, $V_{out} = 1.5\,\mathrm{V}$)
MAX 1927	$0.75\,\mathrm{V} - 5\,\mathrm{V}$ 800 mA	90%	($V_{in} = 2.7V$, $V_{out} = 1.5V$, $I_{load} = 200\,\mathrm{mA}$)	60% 65%	($V_{in} = 3.6\,\mathrm{V}$, $V_{out} = 1.5V$) ($V_{in} = 2.7V$, $V_{out} = 1.5V$)	76%	($V_{in} = 2.7\,\mathrm{V}$, $V_{out} = 1V$)
MAX 1928	$0.75\,\mathrm{V} - 5\,\mathrm{V}$ 800 mA	91%	($V_{in} = 2.7\,\mathrm{V}$ $V_{out} = 1.5\,\mathrm{V}$ $I_{load} = 130\,\mathrm{mA}$)	65% 71%	($V_{in} = 3.6\,\mathrm{V}$, $V_{out} = 1.5\,\mathrm{V}$); ($V_{in} = 2.7\,\mathrm{V}$, $V_{out} = 1.5\,\mathrm{V}$)	80%	($V_{in} = 2.7\,\mathrm{V}$, $V_{out} = 1.5\,\mathrm{V}$)
MAX 1973, MAX 1974	$0.75\,\mathrm{V} - V_{in}$ 1 A	94%	($V_{in} = 3.3\,\mathrm{V}$ $V_{out} = 2.5\,\mathrm{V}$ $I_{load} = 200\,\mathrm{mA}$)	30%	($V_{in} = 3.3\,\mathrm{V}$, $V_{out} = 1\,\mathrm{V}$ $I_{load} = 10\,\mathrm{mA}$)	72%	($V_{in} = 3.3\,\mathrm{V}$)
MAX8500, MAX8504	$2.6\,\mathrm{V} - 5.5\,\mathrm{V}$ 600 mA	90%	($V_{in} = 3.6\,\mathrm{V}$, $V_{out} = 1.5\,\mathrm{V}$, $I_{load} = 200\,\mathrm{mA}$)	67%	($V_{in} = 3.6V$, $V_{out} = 1.5\,\mathrm{V}$, $I_{load} = 10\,\mathrm{mA}$)	71%	($V_{in} = 3.6\,\mathrm{V}$, $V_{out} = 1.5\,\mathrm{V}$)
MAX8506, MAX8508	$0.4\,\mathrm{V} - 3.4\,\mathrm{V}$ 600 mA	89%	($V_{in} = 3.6\,\mathrm{V}$ $V_{out} = 1.2\,\mathrm{V}$ $I_{load} = 200\,\mathrm{mA}$)	51% 52%	($V_{in} = 3.6\,\mathrm{V}$, $I_{load} = 10mA$ PWM-Mode); ($V_{in} = 3.6\,\mathrm{V}$, $I_{load} = 1\,\mathrm{mA}$ Normal-Mode)	74%	($V_{in} = 3.6V$)

Table A.2.: Efficiency of some Switching DC-DC-Converters, Manufactured by MAXIM Dallas

Model	Voltages / I_{max}	Max. Eff. @$V_{out} = 1.2\,\mathrm{V}$		Eff. @$1.2\,\mathrm{V}, I = 1\,\mathrm{mA}$		Eff. @$V_{out} = 1.2\,\mathrm{V}, I_{max}$	
MAX8560, MAX8562	$0.6\,\mathrm{V} - 2.5\,\mathrm{V}$ 500 mA	88%	($V_{in} = 3.6V$ $I_{load} = 120mA$)	40%	($V_{in} = 3.6\,\mathrm{V}$, $I_{load} = 0.1\,\mathrm{mA}$);	70%	
				72%	($V_{in} = 3.6\,\mathrm{V}$, $I_{load} = 1mA$)		
MAX8581, MAX8582	$0.4\,\mathrm{V} - 5.5\,\mathrm{V}$ 600 mA	88%	($V_{in} = 3.6\,\mathrm{V}$, $I_{load} = 200\,\mathrm{mA}$)	49%	($V_{in} = 3.6\,\mathrm{V}$, $I_{load} = 10\,\mathrm{mA}$)	78%	($V_{in} = 3.6V$)
MAX8620Y	$0.6\,\mathrm{V} - 3.3\,\mathrm{V}$ 500 mA	90%	($V_{in} = 3.7\,\mathrm{V}$ $V_{out} = 2.6\,\mathrm{V}$ $I_{load} = 100\,\mathrm{mA}$)	60%	($V_{in} = 3.7\,\mathrm{V}$, $V_{out} = 2.6\,\mathrm{V}$)	78%	($V_{in} = 3.7\,\mathrm{V}$, $V_{out} = 2.6\,\mathrm{V}$)
MAX8621Y, MAX8621Z	$0.6\,\mathrm{V} - 3.3\,\mathrm{V}$ 500 mA	88%	($V_{in} = 3.6\,\mathrm{V}$ $V_{out} = 1.375\,\mathrm{V}$ $I_{load} = 120\,\mathrm{mA}$)	60%	($V_{in} = 3.6\,\mathrm{V}$)	66%	($V_{in} = 3.6\,\mathrm{V}$)

Table A.3.: Efficiency of further Switching DC-DC-Converters, Manufactured by MAXIM Dallas

Model	Voltages / I_{max}	Max. Eff.	@$V_{out} = 1.2\,\mathrm{V}$	Eff. @$1.2\,\mathrm{V}, I = 1\,\mathrm{mA}$	Eff. @$V_{out} = 1.2\,\mathrm{V}, I_{max}$
ADP 2102	$0.8\,\mathrm{V} - 3.3\,\mathrm{V}$	86%	($V_{in} = 2.7\,\mathrm{V}$,	66%	($V_{in} = 4.5\,\mathrm{V}$),	75-80%	($V_{in} = 2.7\,\mathrm{V}$
	600mA		$V_{out} = 1.2\,\mathrm{V}$,	72%	($V_{in} = 3.6\,\mathrm{V}$)		$-4.5\,\mathrm{V}$)
			$I_{load} = 100\,\mathrm{mA}$)				
ADP 3051	$0.8\,\mathrm{V} - 5.5\,\mathrm{V}$	92%	($V_{in} = 2.5\,\mathrm{V}$,	55%	($V_{in} = 3.6\,\mathrm{V}$),	86-88%	($V_{in} = 2.7\,\mathrm{V}$
	500mA		$V_{out} = 1.2\,\mathrm{V}$,	61%	($V_{in} = 2.7\,\mathrm{V}$)		$-5.5\,\mathrm{V}$)
			$I_{load} = 200\,\mathrm{mA}$)				
ADP 2105	$0.8\,\mathrm{V} - V_{in}$	91%	($V_{in} = 2.7\,\mathrm{V}$,	63%	($V_{in} = 5.5\,\mathrm{V}$),	80-83%	($V_{in} = 2.7\,\mathrm{V}$
	1A		$V_{out} = 1.2V$,	70%	($V_{in} = 4.2\,\mathrm{V}$),		$-5.5\,\mathrm{V}$)
			$I_{load} = 200\,\mathrm{mA}$)	78%	($V_{in} = 2.7\,\mathrm{V}$)		
3520f	$0.8\,\mathrm{V} - 5.5\,\mathrm{V}$	92%	($V_{in} = 3\,\mathrm{V}$,	21%	($V_{in} = 3\,\mathrm{V}$	90%	($V_{in} = 3\,\mathrm{V}$,
	600mA		$V_{out} = 1.8\,\mathrm{V}$,		$V_{out} = 1.8\,\mathrm{V}$);		4.2 V
			$I_{load} = 200\,\mathrm{mA}$)	63%	($V_{in} = 4.2\,\mathrm{V}$ | | $V_{out} = 1.8\,\mathrm{V}$)
					$V_{out} = 1.8\,\mathrm{V}$)		
3522fa	$0.6\,\mathrm{V} - 5.5\,\mathrm{V}$	93%	($V_{in} = 2.7\,\mathrm{V}$,	50%	($V_{in} = 2.7\,\mathrm{V}$	90%	($V_{in} = 2.7\,\mathrm{V}$,
	200mA		$V_{out} = 1.8\,\mathrm{V}$,		$V_{out} = 1.8\,\mathrm{V}$),		4.2 V,
			$I_{load} = 100\,\mathrm{mA}$)	40%	($V_{in} = 4.2\,\mathrm{V}$ | | $V_{out} = 1.8\,\mathrm{V}$)
					$V_{out} = 1.8\,\mathrm{V}$)		
3670f	$0.8\,\mathrm{V} - 5.5\,\mathrm{V}$	85%	($V_{in} = 2.5\,\mathrm{V}$,	74%	($V_{in} = 3.6\,\mathrm{V}$),	80%	($V_{in} = 3.6\,\mathrm{V}$),
	400mA		$I_{load} = 10\,\mathrm{mA}$	80%	($V_{in} = 2.5\,\mathrm{V}$)	75%	($V_{in} = 2.5\,\mathrm{V}$)
			$-100\,\mathrm{mA}$)				

Table A.4.: Efficiency of further Switching DC-DC-Converters, Manufactured by Analog Devices (Models AD...), and Linear Technology (Models 3...)

B. Hierarchical Event Streams with Sequences

Suppose, the software of an embedded system contains three tasks: the first triggers the second; the second triggers the third. Given event streams for the first two, $\left(a_1^{(n)}\right)_{n\in\mathbb{N}}, \left(a_2^{(n)}\right)_{n\in\mathbb{N}}$, then how to determine the third one, $\left(a_3^{(n)}\right)_{n\in\mathbb{N}}$?

The answer is computing the cross product of the first two, $\left(a_3^{(n)}\right)_{n\in\mathbb{N}} = \left(a_1^{(n)}\right)_{n\in\mathbb{N}} \times \left(a_2^{(n)}\right)_{n\in\mathbb{N}}$, which is:

$$\begin{aligned}
a_{1,1} &= a_1^{(1)} + a_2^{(1)} \\
a_{1,2} &= a_1^{(1)} + a_2^{(2)} \\
a_{1,3} &= a_1^{(1)} + a_2^{(3)} \\
&\dots \\
a_{1,m} &= a_1^{(1)} + a_2^{(m)} \\
a_{2,1} &= a_1^{(2)} + a_2^{(1)} \\
&\dots
\end{aligned}$$

Define $a_3^{(1)} := a_{1,1}, \dots, a_3^{(m)} := a_{1,m}, a_3^{(m+1)} := a_{2,1}$. This yields the correct event stream for task 3 under the assumptions: the stream for the second task is finite; it is $a_2^{(m)} < a_1^{(2)}$; and it is $\left(a_3^{(n)}\right)_{n\in\mathbb{N}}$ subadditive.

It may be necessary to rearrange the elements of $\left(a_3^{(n)}\right)_{n\in\mathbb{N}}$ if the sequence is not yet subadditive.

If the second stream is infinite, then task three can have multiple events on the same time point, that is:

$$\#\text{events} = \max\{i : a_1^{(i)} - a_2^{(m)} < 0\}. \tag{B.1}$$

The number '#events' denotes the overlap – measured in number of events – of the last element of sequence two on sequence one. The elements of the cross product need to be rearranged to make it a subadditive sequence.

The case that both the second and the first sequence are infinite will result in an infinite number of events of task 3 occurring at the same point in time. If at least one of the first two sequences is finite, then task 3 will have at most events in the same instant as is denoted by Equation (B.1). Exchange the roles of sequence 1 and 2 if necessary.

C. Deterioration of a SANYO HR4U cell

Suppose the following charge sequence:

a) Charge with a constant current of 1 A.

b) Stop charging when either the cell temperature increase exceeds 1°C per minute, or the cell voltage increase exceeds 10 mV per minute.

c) Charge with a topping charge at a current of 100 mA for one hour.

And then start the discharge phase. This combination repeatedly applied to a SANYO HR4U cell excessively stresses the cell that it becomes progressively worse.

Table C.1.: Capacity Degradation while Discharging a Battery of Type SANYO HR4U to 1 V Cut-Off Potential, Days of Measurement: 2009-07-08 and 2009-07-09, Temperature: 20°C

No.	Pre-Discharge	I_{term} [A]	C_{rest} [As]	C_{total} [As]	Diff. [As]
1	none	X	-	3337	0
2	2 A*350 s= 700 As	2 A	1189	1887	0
3	2 A*350 s= 700 As	1.8 A	1273	1971	0
4	2 A*350 s= 700 As	1.2 A	1968	2666	0
5	2 A*350 s= 700 As	1.0 A	2159	2857	0
6	2 A*350 s= 700 As	0.5 A	2524	3223	0
7	none	X	-	3307	30
8	1 A*700 s= 700 As	2 A	1103	1801	88
9	1 A*700 s= 700 As	1.8 A	1209	1907	64
10	1 A*700 s= 700 As	1.2 A	1907	2605	61
11	1 A*700 s= 700 As	1.0 A	2107	2806	59
12	1 A*700 s= 700 As	0.5 A	2507	3205	18
13	none	X	-	3301	36
14	1,2 A*583 s= 700 As	2 A	1059	1735	153
15	1,2 A*583 s= 700 As	1.8 A	1117	1814	157
16	1,2 A*583 s= 700 As	1.2 A	1579	2556	110
17	1,2 A*583 s= 700 As	1.0 A	1710	2772	85
18	1,2 A*583 s= 700 As	0.5 A	1941	3192	31
19	none	1.0 A	-	3288	49

Tables C.1 and C.2 show measurements for the first used cell sample of type SANYO HR4U. The Measurements that are reported within each table were taken consecutively. Between the last measurement of Table C.1 and the charge before the first measurement of Table C.2, there was a gap of about 22 hours.

The column "Diff." shows the difference in capacity between the current measurement to the first measurement of the kind: for example, kind "X" was first used as discharge profile in measurement 1 of Table C.1, and then again it was used in measurement 7 of the same table – the difference was 30 As; or as second example, measurements 2 and 8 are of the same kind – both are terminated with the same discharge current – the difference is 88 As.

For explanation, the discharge profile "X" is to discharge with a sequence of constant currents to cut-off potential, and to switch to the next current without prior charge when cut-off voltage is reached. The sequence is 2 A, 1.8 A, 1.2 A, 1 A, 0.9 A, 0.6 A, 0.5 A, 0.45 A, 0.3 A, 0.25 A.

The idea behind profile "X" was to study whether the proposed law holds for each of the currents. The results should be validated by several measurements, but as the cell started to deteriorate, the results could not be used.

Table C.2.: Further Capacity Degradation while Discharging a Battery of Type SANYO HR4U to 1 V Cut-Off Potential, Continued Measurements, Date: 2009-07-10 through 07-12, Temperature: 20°C

No.	Pre-Discharge	I_{term} [A]	C_{rest} [As]	C_{total} [As]	Diff. [As]
1	none	X	-	3338	-1
2	2 A*350 s= 700 As	2 A	947	1645	242
3	2 A*350 s= 700 As	1.8 A	1010	1708	263
4	2 A*350 s= 700 As	1.2 A	1775	2473	193
5	2 A*350 s= 700 As	1.0 A	2007	2706	151
6	2 A*350 s= 700 As	0.5 A	2468	3167	56
7	none	X	-	3278	49
8	1 A*700 s= 700 As	2 A	838	1537	350
9	1 A*700 s= 700 As	1.8 A	903	1601	270
10	1 A*700 s= 700 As	1.2 A	1682	2380	286
11	1 A*700 s= 700 As	1.0 A	1926	2624	233
12	1 A*700 s= 700 As	0.5 A	2443	3142	81
13	none	X	-	3262	85
14	1,2 A*583 s= 700 As	2 A	675	1374	513
15	1,2 A*583 s= 700 As	1.8 A	763	1462	509
16	1,2 A*583 s= 700 As	1.2 A	1615	2314	352
17	1,2 A*583 s= 700 As	1.0 A	1880	2579	278
18	1,2 A*583 s= 700 As	0.5 A	2432	3130	113
19	none	1.0 A	-	3258	79

Bibliography

[ABS2006] Karsten Albers, Frank Bodmann and Frank Slomka. Hierarchical event streams and event dependency graphs: A new computational model for embedded real-time systems. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 97–106, 2006.

[ABS2007] Karsten Albers, Frank Bodmann and Frank Slomka. Run-time efficient feasibility analysis of uni-processor systems with static priorities. In *Proceedings of the International Embedded Systems Symposium*, 2007.

[Abs2009] AbsInt Angewandte Informatik GmbH, Science Park 1, 66123 Saarbrücken, Germany. ait – worst-case execution time analyzers. http://www.absint.com/, 2009. Last visited: 16th October 2009.

[AEP+2007] Alexandru Andrei, Petru Eles, Zebo Peng, Marcus T. Schmitz and Bashir M. Al Hashimi. Energy optimization of multiprocessor systems on chip by voltage selection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(3), March 2007.

[AS2004] Karsten Albers and Frank Slomka. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 187–195, 2004.

[AS2005] Karsten Albers and Frank Slomka. Efficient feasibility analysis for real-time systems with EDF scheduling. In *Proceedings of the Design, Automation and Test in Europe Conference*, 2005.

[ASE+2004] Alexandru Andrei, Marcus Schmitz, Petru Eles, Zebo Peng and Bashir M. Al-Hashimi. 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.

[BAS2006] Frank Bodmann, Karsten Albers and Frank Slomka. Analyzing the timing characterics task activations. In *Proceedings of the first IEEE Symposium on Industrial Embedded Systems*, 2006.

[BCGM1999] Sanjoy Baruah, Deji Chen, Sergey Gorinsky and Aloysius Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 7 1999.

[BCM+2000] Luca Benini, G. Castelli, A. Macii, Enrico Macii, M. Poncino and R. Scarsi. A discrete-time battery model for high-level power estimation. In *Proceedings*

of the Design, Automation and Test in Europe Conference, pages 35–41, New York, USA, 2000. ACM Press.

[BdM2000] Luca Benini and Giovanni de Micheli. System-level power optimization: techniques and tools. *ACM Transactions on Design Automation of Electronic Systems*, 5(2):115–192, 2000.

[BKN1999] H.J. Bergveld, W.S. Krujit and P.H.L. Notten. Electronic-network modelling of rechargeable NiCd cells and its applications to the design of battery management systems. *Journal of Power Sources*, 77:143–158, Feb 1999.

[BRH1990] Sanjoy K. Baruah, Louis E. Rosier and Rodney R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, 1990.

[BW1995] Alan Burns and Andy Wellings. *HRT-HOOD™: A Structured Design Method for Hard Real-Time Ada Systems*, volume 3 of *Real-Time Safety Critical Systems*. Elsevier Science B.V., Amsterdam, 1995.

[CG2005] Hui Cheng and Steve Goddard. Integrated device scheduling and processor voltage scaling for system-wide energy conservation. In *The Second International Workshop On Power-Aware Real-Time Computing (PARC)*, 2005.

[Chr2009] INCHRON GmbH, August-Bebel-Straße 88, 14482 Potsdam, Germany. Real-time simulator chronsim - add-on module chronest. http://www.inchron.de/chronest.html, 2009. Last visited: 16th October 2009.

[CR1999] Carla Fabiana Chiasserini and Ramesh R. Rao. A model for battery pulsed discharge with recovery effect. In *Proceedings of the IEEE Wireless Communications & Networking Conference*, 1999.

[CT2005] Samarjit Chakraborty and Lothar Thiele. A new task model for streaming applications and its schedulability analysis. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 486–491, Washington, DC, USA, 2005. IEEE Computer Society.

[CYVA1996] Anantha Chandrakasan, Isabel Yang, Carlin Vieri and Dimitri Antoniadis. Design considerations and tools for low-voltage digital system design. In *Proceedings of the Design Automation Conference*, 1996.

[Dev2003] UmaMaheswari Devi. An improved schedulability test for uniprocessor periodic task systems. In *Proceedings of the 15th Euromicro conference on Real-Time Systems*, pages 23–30, 2003.

[DFN1993] Marc Doyle, Thomas F. Fuller and John Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of the Electrochemical Society*, 140:1526–1533, 1993.

[DS2006] Dennis Doerffel and Suleiman Abu Sharkh. A critical review of using the Peukert equation for determining the remaining capacity of lead-acid and lithium-ion batteries. *Journal of Power Sources*, 155(2):395–400, 2006.

[ESP2006] Noel Eisley, Vassos Soteriou and Li-Shiuan Peh. High-level power analysis for multi-core chips. In *CASES '06: Proceedings of the 2006 international conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 389–400, New York, NY, USA, 2006. ACM.

[ETM2008] Vahid Esfahanian, Farschad Torabi and Ali Mosahebi. An innovative computational algorithm for simulation of lead-acid batteries. *Journal of Power Sources*, 176:373–380, Jan 2008.

[FDN1994a] Thomas F. Fuller, Marc Doyle and John Newman. Relaxation phenomena in lithium-ion-insertion cells. *Journal of the Electrochemical Society*, 141:982–990, 1994.

[FDN1994b] Thomas F. Fuller, Marc Doyle and John Newman. Simulation and optmization of the dual lithium ion insertion cell. *Journal of the Electrochemical Society*, 141:1–10, 1994.

[FF2005] Antoni Ferré and Juan Figueras. Leakage in CMOS nanometric technologies. In Christian Piguet, editor, *Low-Power Electronics Design*, chapter 3. CRC Press, Boca Raton, London, New York, Washington, D.C., 2005.

[FP2002] Farzan Fallah and Massoud Pedram. Circuit and system level power management. In Massoud Pedram and Jan M. Rabaey, editors, *Power Aware Design Methodologies*, chapter 13. Kluwer Academic Publishers, Boston Dordrecht London, 2002.

[GIS2003] Rajesh K. Gupta, Sandy Irani and Sandeep K. Shukla. Formal methods for dynamic power management. In *ICCAD '03: Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design*, page 874, Washington, DC, USA, 2003. IEEE Computer Society.

[Gla1996] M.C. Glass. Battery electrochemical nonlinear/dynamic SPICE model. In *Proceedings of the 31st Intersociety Energy Conversion Engineering Conference (IECEC)*, pages 292–297, Aug 1996.

[GMS+2007] Parthasarathy M. Gomadam, Don R. Merrit, Erik R. Scott, Craig L. Schmidt, Paul M. Skarstad and John M. Weidner. Modeling lithium/hybrid-cathode batteries. *Journal of Power Sources*, 174:872–876, Dec 2007.

[Gol1997] S. Gold. A PSPICE macromodel for lithium-ion batteries. In *Twelfth Annual Battery Conference on Applications and Advances*, pages 215–222, Jan 1997.

[Gre1993] Klaus Gresser. *Echtzeitnachweis Ereignisgesteuerter Realzeitsysteme*. VDI Verlag, Düsseldorf, 1993. Dissertation.

[GW2000] W. B. Gu and C. Y. Wang. Thermal-electrochemical modeling of battery systems. *Journal of the Electrochemical Society*, 147:2910–2922, 2000.

[Hag1993] Steven C. Hageman. Simple PSpice models let you simulate common battery types. *Electronic Design News (EDN)*, page 117, Oct 1993.

[Hag1995] Steven C. Hageman. PSpice models nickel-metal-hydride cells. *Electronic Design News (EDN)*, Feb 1995.

[Her2004] James Herr. An accurate battery gas gauge introduction. *Linear Technology Magazine*, pages 11–12, 2004.

[HKQ+1998] Inki Hong, Darko Kirovski, Gang Qu, Miodrag Potkonjak and Mani B. Srivastava. Power optimization of variable voltage core-based systems. In *Proceedings of the 35th annual Conference on Design Automation*, pages 176–181, 1998.

[HL1999] John N. Harb and Rodney M. LaFolette. Mathematical model of the discharge behavior of a spirally wound lead-acid cell. *Journal of the Electrochemical Society*, 146:809–818, 1999.

[HW1997] Chi-Hong Hwang and Allen C.-H. Wu. A predictive system shutdown method for energy saving of event-driven computation. In *ICCAD '97: Proceedings of the 1997 IEEE/ACM International Conference on Computer-Aided Design*, pages 28–32, Washington, DC, USA, 1997. IEEE Computer Society.

[ISG2003] Sandy Irani, Sandeep Shukla and Rajesh Gupta. Algorithms for power savings. In *SODA 2003: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 37–46, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[ITR2007] International technology roadmap for semiconductors 2007 edition – design. http://www.itrs.net, 2007. Last visited: 16th October 2009.

[IY1998] Tohru Ishihara and Hiroto Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the international symposium on Low power electronics and design*, pages 197–202, 1998.

[JG2004] Ravindra Jejurika and Rajesh Gupta. Optmized slowdown in real-time task systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 155–164, 2004.

[JLSM2003] Nathalie Julien, Johann Laurent, Eric Senn and Eric Martin. Power consumption modeling and characterization of the TI C6201. *IEEE Micro*, 23(5):40–49, 2003.

[JPG2004] Ravindra Jejurika, Cristiano Pereira and Rajesh Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st annual Conference on Design Automation*, pages 275–280, New York, NY, USA, 2004. ACM.

[KFM+1996] Tadahiro Kuroda, Tetsuya Fujita, Shinji Mita, Tetsu Nagamatsu, Shinichi Yoshioka, Kojiro Suzuki, Fumihiko Sano, Masayuki Norishima, Masayuki Murota, Makoto Kako, Masaaki Kinugawa, Masakazu Kakumu and Takayasu Sakurai. A 0.9-V, 150-MHz, 10-mW, 4 mm, 2-D discrete cosine transform core processor with variable threshold-voltage (VT) scheme. *IEEE Journal of Solid-State Circuits*, 31(11):1770–1779, 11 1996.

[LAS2006] Henrik Lipskoch, Karsten Albers and Frank Slomka. Battery discharge aware energy feasibility analysis. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pages 22–27. ACM Press, 2006.

[LAS2007] Henrik Lipskoch, Karsten Albers and Frank Slomka. Fast calculation of permissable slowdown factors for hard real-time systems. In Nadine Azemard and Lars Svensson, editors, *Power and Timing Modeling, Optimization and Simulation, 17th International Workshop*, number 4644 in Lecture Notes in Computer Science, pages 495–504. Springer Verlag, 2007.

[LDWG2005] Sheng Liu, Roger A. Dougal, John W. Weidner and Lijun Gao. A simplified physics-based model for nickel hydrogen battery. *Journal of Power Sources*, 141:326–339, Mar 2005.

[LHW2002] Tin-Man Lee, Jörg Henkel and Wayne Wolf. 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.

[LJ2000] Jiong Luo and Niraj K. Jha. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In *ICCAD '00: Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design*, pages 357–364, Piscataway, NJ, USA, 2000. IEEE Press.

[LJSM2004] Johann Laurent, Nathalie Julien, Eric Senn and Eric Martin. Functional level power analysis: An efficient approach for modeling the power consumption of complex processors. In *Proceedings of the Design, Automation and Test in Europe Conference*, page 10666, Washington, DC, USA, 2004. IEEE Computer Society.

[LL1973] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[LM2001] Yung-Hsiang Lu and Giovanni De Micheli. Comparing system-level power management policies. *IEEE Design & Test of Computers*, 18(2):10–19, 2001.

[LMW1999] Yau-Tsun Steven Li, Sharad Malik and Andrew Wolfe. Performance estimation of embedded software with instruction cache modeling. *ACM Transactions on Design Automation of Electronic Systems*, 4(3):257–279, 1999.

[LR2002] David Linden and Thomas B. Reddy. *Handbook of Batteries*. McGraw-Hill, New York, 3. edition, 2002.

[LRK2003] Yann-Hang Lee, K.P. Reddy and C.M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 105–112, July 2003.

[LS2009] Henrik Lipskoch and Frank Slomka. Task-dependent processor shutdown for hard real-time systems. In Achim Rettberg, Mauro C. Zanella, Michael Amann, Michael Keckeisen and Franz J. Rammig, editors, *Analysis, Architectures and Modeling of Embedded Systems – Third IFIP TC 10 International Embedded Systems Symposium, IESS 2009*, number 310 in IFIP Advances in Information and Communication Technology, pages 127–138. Springer Verlag, 2009.

[LTMF1995] Mike Tien-Chien Lee, Vivek Tiwari, Sharad Malik and Masahiro Fujita. Power analysis and low-power scheduling techniques for embedded DSP software. In *Proceedings of the 8th international symposium on System synthesis*, New York, NY, USA, 1995. ACM Press.

[Max2003] Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086, USA. *APPLICATION NOTE 2096: DS2740 High-Precision Coulomb Counter Reference Design*, 06 2003. http://www.maxim-ic.com.

[MB2009] J. M. Martinez and E. G. Birgin. Trustable algorithms for nonlinear general optimization. the University of Campinas and University of São Paulo joint project for optimization software development. http://www.ime.usp.br/∼egbirgin/tango/index.php, 2009. Last visited: 16th October 2009.

[MKT2004] Alexander Maxiaguine, Simon Künzli and Lothar Thiele. Workload charaterization model for tasks with variable execution demand. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 1040–1045, Washington, DC, USA, 2004. IEEE Computer Society.

[MM2002] Diana Marculescu and Radu Marculescu. System and microarchitectural level power modeling, optimization, and their implications in energy aware computing. In Massoud Pedram and Jan M. Rabaey, editors, *Power Aware Design Methodologies*, chapter 9. Kluwer Academic Publishers, Boston Dordrecht London, 2002.

[MR2003] Saibal Mukhopadyay and Kaushik Roy. Modeling and estimation of total leakage current in nano-scaled CMOS devices considering the effect of parameter variation. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 172–175, 2003.

[New2009] John Newman. FORTRAN programs for the simulation of electrochemical systems. http://www.cchem.berkeley.edu/jsngrp/fortran.html, 2009. Last visited: 16th October 2009.

[NM1997] Wolfgang Nebel and Jean Mermet, editors. *Low Power Design in Deep Submicron Electronics*, volume 337 of *NATO ASI Series*. Kluwer Academic Publishers, Dordrecht / Boston / London, 1997.

[NQ2004] Linwei Niu and Gang Quan. Reducing both dynamic and leakage energy consumption for hard real-time systems. In *CASES '04: Proceedings of the 2004 international conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 140–148, New York, NY, USA, 2004. ACM.

[OP2004] Jingzhao Ou and Viktor K. Prasanna. Rapid energy estimation of computations on FPGA based soft processors. In *Proceedings of IEEE International System-on-a-Chip Conference*, 2004.

[Pan2005] Panasonic Corporation of North America, One Panasonic Way, Secaucus, NJ 07094 USA. *Valve-Regulated Lead Acid Batteries: Individual Data Sheet LC-R061R3P*, 08 2005. http://www.panasonic.com/industrial/battery/oem/chem/vrla/.

[Pan2008] Panasonic Corporation of North America, One Panasonic Way, Secaucus, NJ 07094 USA. *Lithium Ion Batteries: Individual Data Sheet, CGR 26650 A: High Rate Cylindrical Model*, 12 2008. http://www.panasonic.com/industrial/battery/oem/chem/lithion/.

[Peu1897] W. Peukert. Ueber die Abhängigkeit der Kapacität von der Entladestromstärke bei Bleiakkumulatoren. *Elektrotechnische Zeitschrift*, 18(20):287–288, 1897.

[Pig2005] Christian Piguet, editor. *Low-Power Electronics Design*. CRC Press, 2005.

[PR2002] Massoud Pedram and Jan Rabaey, editors. *Power Aware Design Methodologies*. Kluwer Academic Publishers, Norwell (MA), USA, Dordrecht, The Netherlands, 2002.

[PSW2002] Y.H. Pan, V. Srinivasan and C.Y Wang. An experimental and modeling study of isothermal charge/discharge behavior of commercial Ni-MH cells. *Journal of Power Sources*, 112:298–306, Oct 2002.

[PW1999] Massoud Pedram and Qing Wu. Design considerations for battery-powered electronics. In *DAC '99: Proceedings of the 36th ACM/IEEE Conference on Design Automation*, pages 861–866, New York, NY, USA, 1999. ACM.

[RE2002] Kai Richter and Rolf Ernst. Event model interfaces for heterogeneous system analysis. In *Proceedings of the Design, Automation and Test in Europe Conference*, Washington, DC, USA, 2002. IEEE Computer Society.

[RE2008] Jonas Rox and Rolf Ernst. Modeling event stream hierarchies with hierarchical event models. In *Proceedings of the Design, Automation and Test in Europe Conference*, 2008.

[RF1900] C. A. Rossander and E. A. Forsberg. Ueber die Vorausbestimmung der erforderlichen Kapacität von Akkumulatorenbatterien. *Elektrotechnische Zeitschrift*, 21(43):881–883, 1900.

[RHE+2006] Razvan Racu, Arne Hamann, Rolf Ernst, Bren Mochocki and Xiaobo Hu. Methods for power optimization in distributed embedded systems with real-time requirements. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 379–388, 2006.

[RJE2005] Razvan Racu, Marek Jersak and Rolf Ernst. Applying sensitivity analysis in real-time distributed systems. In *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 160–169, Washington, DC, USA, 2005. IEEE Computer Society.

[RMMM2003] Kaushik Roy, Saibal Mukhopadyay and Hamid Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submircometer CMOS circuits. *Proceedings of the IEEE*, 91(2), February 2003.

[RP2003] Peng Rong and Massoud Pedram. An analytical model for predicting the remaining battery capacity of Lithium-Ion batteries. In *Proceedings of the Design, Automation and Test in Europe Conference*, Washington, DC, USA, 2003. IEEE Computer Society.

[RP2006] Peng Rong and Massoud Pedram. 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.

[RV2001] Daler N. Rakhmatov and Sarma B. K. Vrudhula. An analytical high-level battery model for use in energy management of portable electronic systems. In *ICCAD '01: Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, pages 488–493, Piscataway, NJ, USA, 2001. IEEE Press.

[RVR2003] Ravishankar Rao, Sarma Vrudhula and Daler N. Rakhmatov. Battery modeling for energy aware system design. *Computer*, 36(12):77–87, Dec. 2003.

[Saf2006] Saft Specialty Battery Group, 12, rue Sadi Carnot, 93170 Bagnolet - France. *Primary lithium battery LM 33600*, 10 2006. http://www.saftbatteries.com/.

[SAN2003] SANYO Energy (USA) Corporation, Sanyo Fisher Sales Europe GmBH, Stahlgruberring 4, 81829 München. *Cell Type UR18650F Specifications, Lithium ion*, 08 2003. http://us.sanyo.com/batteries/specs.cfm.

[SAN2007] SANYO Energy (USA) Corporation, Sanyo Fisher Sales Europe GmBH, Stahlgruberring 4, 81829 München. *Cell Type CP-3600CR Specifications*, 05 2007. http://us.sanyo.com/batteries/specs.cfm.

[SC1997] K.C. Syracuse and W.D.K. Clark. A statistical approach to domain performance modeling for oxyhalide primary lithium batteries. In *Twelfth Annual Battery Conference on Applications and Advances*, pages 163–170, Jan 1997.

[SCB1996] Mani B. Srivastava, Anantha P. Chandrakasan and R. W. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(1):42–55, 1996.

[Sch2009] Nils Schröder. Softwarekongurierbare Batteriemessstation für automatische Messungen. Individuelles Projekt (student's thesis), Carl von Ossietzky Universität Oldenburg, 2009.

[SCI2001] Vishnu Swaminathan, Krishnendu Chakrabarty and S. S. Iyengar. Dynamic I/O power management for hard real-time systems. In *CODES '01: Proceedings of the ninth international symposium on Hardware/software codesign*, pages 237–242, New York, NY, USA, 2001. ACM.

[SHW2005] Andrew S. Stamps, Charles E. Holland and Ralph E. White. Analysis of capacity fade in a lithium ion battery. *Journal of Power Sources*, 150:229–239, Oct 2005.

[SK2005] Dongkun Shin and Jihong Kim. Intra-task voltage scheduling on DVS enabled hard real-time systems. *IEEE - Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(10):1530–1549, October 2005.

[SKL2006] Jaewon Seo, Taewhan Kim and Joonwon Lee. Optimal intratask dynamic voltage-scaling technique and its practical extensions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(1):47–57, January 2006.

[ST 2007] ST Microelectronics, Worldwide Headquarters STMicroelectronics; 39, Chemin du Champ des Filles; C. P. 21; CH 1228 Plan-Les-Ouates; GENEVA, Switzerland. *Datasheet for the STM32F103 microcontroller*, 4. edition, nov 2007. http://www.st.com.

[ST 2009] ST Microelectronics, Worldwide Headquarters STMicroelectronics; 39, Chemin du Champ des Filles; C. P. 21; CH 1228 Plan-Les-Ouates; GENEVA, Switzerland. *Datasheet for STC3100: Battery monitor IC with Coulomb counter/gas gauge*, jan 2009. www.st.com.

[TC1994] Ken Tindell and John Clark. 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):117–134, 1994.

[TCN2000] Lothar Thiele, Samajit Chakraborty and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of the International Symposium on Circuits and Systems*, pages 101–104, 2000.

[TMW1994] Vivek Tiwari, Sharad Malik and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. In *ICCAD '94: Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design*, pages 384–390, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[VAR2003] VARTA Microbattery GmbH, Daimlerstr. 1, D-73479 Ellwangen/Jagst. *VH 2700 A High Cap, Rechargeable Ni-MH Cylindrical, Data Sheet*, 05 2003. http://www.varta.de.

[Wei2008] Thomas Weißmüller. Entwicklung und Aufbau einer Batteriemessstation mit anschließenden Experimenten an ausgewählten Zellen. Master's thesis, Carl von Ossietzky Universität Oldenburg, 2008.

[WMT2005] Ernesto Wandeler, Alexandre Maxiaguine and Lothar Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. *Real-time Systems*, 29(2):205–225, March 2005.

[ZW2008] Qi Zhang and Ralph E. White. Capacity fade analysis of a lithium ion cell. *Journal of Power Sources*, 179:793–798, May 2008.

Index

W

Curriculum Vitae

Personal Data

Date of birth:	29/10/1979
Place of birth:	Varel, Friesland, Germany

Education

04/2006 – 02/2010	Doctorand, department of computer science Carl von Ossietzky Universität Oldenburg
12/2005 – 11/2008	Scholarship holder of the graduate school “TrustSoft”
10/1999 – 11/2005	Studies in mathematics and computer science
11/2005	Diploma-degree in mathematics Carl von Ossietzky Universität Oldenburg
01/2003 – 06/2003	Abroad studies with the european Sokrates/Erasmus programme in Linköping, Sweden
04/2002	Prediploma in both computer science and mathematics
06/1999	“Allgemeinen Hochschulreife” at Lothar-Meyer-Gymnasium Varel (general qualification for university)

Teachings

Bachelor’s thesis, supervisor and reviewer

2008/2009	Nils Schröder “Softwarekonfigurierbare Batteriemessstation für automatische Messungen”
2007	Daniel Schlitt “Simulative Energieabschätzung auf Taskebene am Beispiel des SA-Radars”
2007	Johannes Fischer “Vorhersage der Leistungsaufnahme von Softcores auf dem FPGA”
2006/2007	Kilian Kempf “Entwurf einer Einchip-Multiprozessorplattform mit Nios-Prozessoren”
2006	Thomas Weißmüller “Skalierbare und rekonfigurierbare Hardwareplattform für prototypische Roboteranwendungen”

Master’s thesis, supervisor

2008	Thomas Weißmüller “Entwicklung und Aufbau einer Batteriemessstation mit anschließenden Experimenten an ausgewählten Zellen”

Projektgruppe (student group)

2006/2007	Co-supervisor of the student group “Smart Power Grids”

Lectures

WS 2007/2008, WS 2008/2009	Guest presentations on “Electrochemical cells for mobile embedded systems” within the lecture “Low Energy System Design” (energy efficiency group, Prof.Dr.-Ing.Nebel, department of computer science, Carl von Ossietzky Universität Oldenburg,)

Publications

[LS 2009] H. Lipskoch and F. Slomka. Task-dependent processor shutdown for hard real-time systems. In A. Rettberg, M. C. Zanella, M. Amann, M. Keckeisen, and F. J. Rammig, editors, *Analysis, Architectures and Modeling of Embedded Systems – Third IFIP TC 10 International Embedded Systems Symposium, IESS 2009*, number 310 in IFIP Advances in Information and Communication Technology, pages 127–138. Springer Verlag, 2009.

[Lip2009] H. Lipskoch. Optimisation of battery operating life considering software tasks and their timing behaviour, 2009. Poster Presentation at EDAA PhD Forum, together with the Design, Automation and Test in Europe Conference, Nice.

[DLMS2008] M. Diehl, H. Lipskoch, R. Meyer, and C. Storm, editors. *Proceedings des gemeinsamen Workshops der Graduiertenkollegs 2008*. Trustworthy Software Systems. Gito-Verlag, Berlin, May 2008.

[LAS2007] H. Lipskoch, K. Albers, and F. Slomka. Fast calculation of permissable slowdown factors for hard real-time systems. In N. Azemard and L. Svensson, editors, *Power and Timing Modeling, Optimization and Simulation, 17th International Workshop*, number 4644 in Lecture Notes in Computer Science, pages 495–504. Springer Verlag, 2007.

[Lip2007] H. Lipskoch. Energy optimisation of embedded hardware / software systems (abstract of Ph.D. thesis). In *Dagstuhl "'zehn plus eins"'*, page 75. Verlagshaus Mainz GmbH, Aachen, 06 2007.

[Lip2006] H. Lipskoch. Energy optimisation of embedded hardware / software systems (abstract of Ph.D. thesis). In J. Happe, H. Koziolek, M. Rohr, C. Storm, and T. Warns, editors, *Proceedings of the International Research Training Group Workshop*, volume 3, page 2. GITO-Verlag, 2006.

[LAS2006] H. Lipskoch, K. Albers, and F. Slomka. Battery discharge aware energy feasibility analysis. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pages 22–27. ACM Press, 2006.

[WKHL2006] D. Winteler, H. Koziolek, J. Happe, and H. Lipskoch. Die urheberrechtliche Problematik geschlossener Linux Kernelmodule aus Sicht des deutschen Rechts. In C. Hochberger and R. Liskowsky, editors, *Informatik 2006, Informatik für den Menschen, Band 2, ISOS Workshop*, Lecture Notes in Informatics, pages 77–82. Gesellschaft für Informatik, 10 2006.

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe.
Ebenso versichere ich, dass ich diese Disseration nur in diesem Promotionsverfahren eingereicht habe und dass diesem Promotionsverfahren keine anderen endgültig nicht bestandenen Promotionsverfahren vorausgegangen sind.

Henrik Lipskoch, Oldenburg, den 23.02.2010