Linköping Studies in Science and Technology

Dissertation No. 1127

Energy Efficient and Predictable Design of Real-Time Embedded Systems

by

Alexandru Andrei

Linköpings universitet

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2007

Acknowledgments

First and foremost I would like to thank my adviser Professor Petru Eles. His passion and thoroughness made this thesis possible. I will always remember the nights before the paper submission deadlines when Petru was always there, actively working to improve the papers. His commitment will always inspire me.

I would like to extend my gratitude towards my secondary adviser, Professor Zebo Peng. By always challenging my ideas, he contributed significantly to my progress as a researcher.

The former and present colleagues from the Embedded Systems Laboratory have provided a friendly environment. Special thanks to my former office colleague, Marcus Schmitz, who taught me how to write technical papers.

Gunilla Mellheden, Anne Moe and Lillemor Walgreen have been invaluable in their efforts to simplify all the administrative details.

I would like to acknowledge the financial support of CUGS (Swedish National Research School of Computer Science), SSF (Swedish Foundation for Strategic Research via the STRINGENT program) and ARTIST Network of Excellence in Embedded Systems. This work would not have been possible without their funding.

My friends from all over the world, are an endless source of joy and inspiration. I would not be the same without them.

I am deeply grateful to Diana and to my family for their constant support. This thesis is dedicated to them.

Alexandru Andrei
Linköping, September 2007

Abstract

This thesis addresses several issues related to the design and optimization of embedded systems. In particular, in the context of time-constrained embedded systems, the thesis investigates two problems: the minimization of the energy consumption and the implementation of predictable applications on multiprocessor system-on-chip platforms.

Power consumption is one of the most limiting factors in electronic systems today. Two techniques that have been shown to reduce the power consumption effectively are dynamic voltage selection and adaptive body biasing. The reduction is achieved by dynamically adjusting the voltage and performance settings according to the application needs. Energy minimization is addressed using both offline and online optimization approaches. Offline, we solve optimally the combined supply voltage and body bias selection problem for multiprocessor systems with imposed time constraints, explicitly taking into account the transition overheads implied by changing voltage levels. The voltage selection technique is applied not only to processors, but also to buses with repeaters and fat wires. We investigate the continuous voltage selection as well as its discrete counterpart. While the above mentioned methods minimize the active energy, we propose an approach that combines voltage selection and processor shutdown in order to optimize the total energy.

In order to take full advantage of slack that arises from variations in the execution time, it is important to recalculate the voltage and performance settings during run-time, i.e., online. However, voltage scaling is computationally expensive, and, thus, performed at runtime, significantly hampers the possible energy savings. To overcome the online complexity, we propose a quasi-static voltage scaling scheme, with a constant online time complexity $O(1)$. This allows to increase the exploitable slack as well as to avoid the energy dissipated due to online recalculation of the voltage settings.

Worst-case execution time (WCET) analysis and, in general, the predictability of real-time applications implemented on multiprocessor systems has been ad-

dressed only in very restrictive and particular contexts. One important aspect that makes the analysis difficult is the estimation of the system's communication behavior. The traffic on the bus does not solely originate from data transfers due to data dependencies between tasks, but is also affected by memory transfers as result of cache misses. As opposed to the analysis performed for a single processor system, where the cache miss penalty is constant, in a multiprocessor system each cache miss has a variable penalty, depending on the bus contention. This affects the tasks' WCET which, however, is needed in order to perform system scheduling. At the same time, the WCET depends on the system schedule due to the bus interference. In this context, we propose, an approach to worst-case execution time analysis and system scheduling for real-time applications implemented on multiprocessor SoC architectures.

This work has been supported by CUGS–Swedish National Graduate School of Computer Science– SSF–Swedish Foundation for Strategic Research-via the STRINGENT program–and ARTIST– Network of Excellence on Embedded Systems Design.

Contents

List of Figures

List of Tables

Part I

Preliminaries

Chapter 1

Introduction

The electronic industry has grown in an unprecedented way, from the invention of the transistor in 1947. As a result, we are surrounded today by various gadgets, ranging from mobile phones, digital cameras and PDAs to complex electronic control units in automobiles and planes or powerful computers. Due to the ever decreasing feature size, the number of transistors in a chip doubles every 18 month. This development, predicted by Gordon Moore [Moo65] in 1965 and known as Moore's law, is the main factor driving this growth.

The design of such complex systems is a difficult task. The heavy competition is escalating the demand for small, high-performance, low-power consumer electronics products that are affordable and, at the same time, offer new functionality at each new generation. These characteristics will increasingly conflict, as advanced features consume power and area, as well as increasing development costs. This challenge is hitting a critical point at the sub-90nm realm, resulting in an ever-widening productivity gap [ITR].

The best way to close this gap and cost-effectively meet new consumer demands is through the use of advanced electronic design automation (EDA) tools that already address these challenges at early design stages.

We can differentiate two big classes of electronic systems: general purpose computer systems and embedded systems.

In this thesis, we will restrict the discussion to the class of embedded systems. Embedded systems must not only implement the desired functionality but must also satisfy diverse constraints (power and energy consumption, performance, safety, size, cost, flexibility, etc.) that typically compete with each other. Moreover, the ever increasing complexity of embedded systems combined with small time-to-market windows poses great challenges to the design comunity.

This chapter briefly presents some issues related to the the embedded systems design flow. In particular, the chapter emphasizes the issue of power consumption and introduces some of the possible solutions that will be further explored in the thesis. The challenges of such an endeavor are discussed and the contributions of the thesis are highlighted. The section concludes by presenting the outline of the thesis.

1.1 Generic Design Flow for Embedded Systems

Fig. 1.1 presents a generic design flow for embedded systems development. The design usually starts from an informal specification, that describes the desired functionality as well as possible constraints (physical size of the device, performance, energy consumption, lifetime, etc.). This informal specification is later refined in a model of the system. The model can be validated against the specification by performing formal verification or functional simulation.

Assuming that the model is correct, the next step is the selection of the hardware architecture. This step is crucial, because it impacts the cost of the final product. Moreover, it has a big impact on other parameters, such as performance and energy consumption, restricting the possible choices that are made in the next steps. Implicitly, at this stage of the design, the functionality is partitioned in time-critical components that require dedicated hardware (ASICs) and software components (tasks) that will be running on programmable processors.

Once the architecture is selected, we proceed with mapping of the software tasks to the programable processors. The processors composing the hardware architecture may come from different families or even from different manufacturers. Thus, they can have different characteristics. For example, the processors can have different instruction sets, can potentially operate at different frequencies, or they might have different cache parameters. This leads to potentially different execution times of a certain software task, depending on the processor where the task is mapped. During the next step the tasks are scheduled, i.e. the order of execution, priorities, and, possibly, the times when the tasks will start are decided. During this stage, several issues have to be considered. A key factor that must be taken into account is the set of dependencies that might exist between the tasks. Such a dependency states, for example, that a certain task can only start when all the tasks it depends on have finished. In time-constrained systems, where some of the tasks must finish before a certain deadline, mapping and scheduling are closely coupled with an analysis that decides if the timing constraints are met. If this is not the case, other schedules and mappings are explored. These decisions can be made at

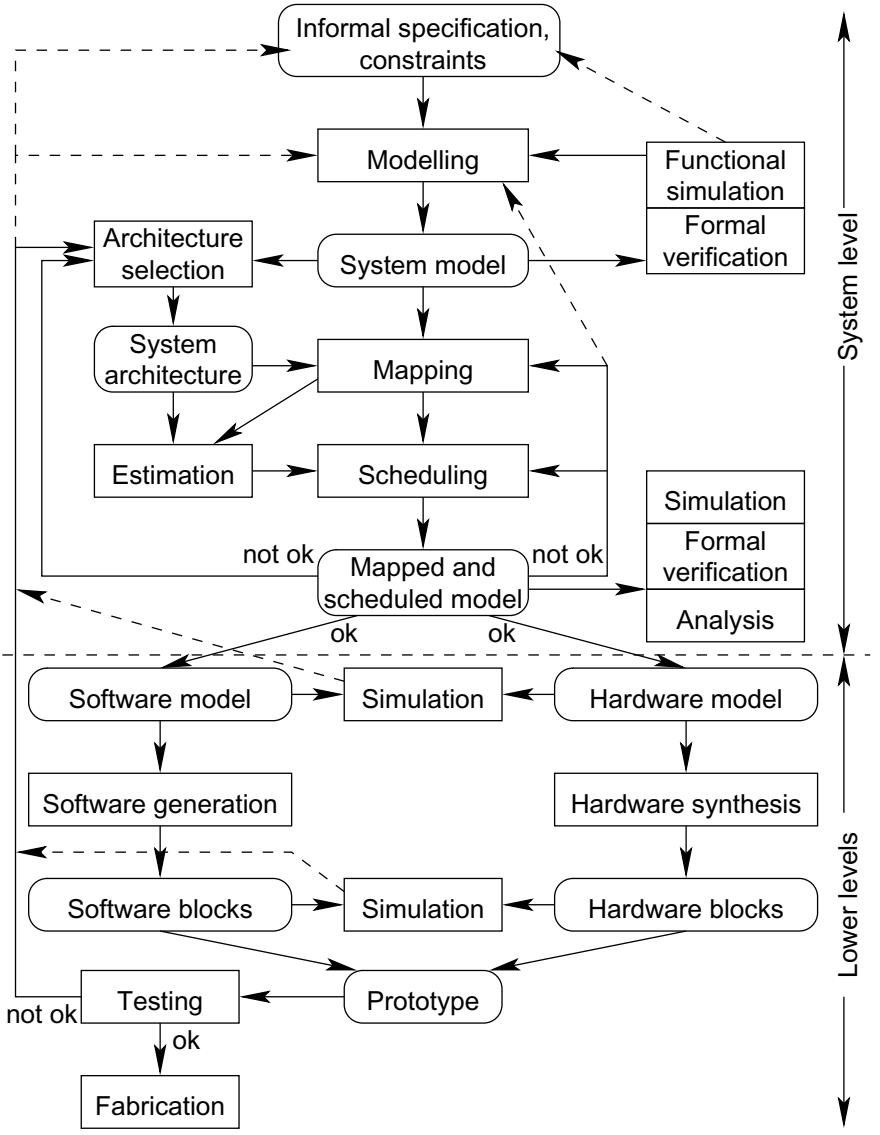


Figure 1.1: Generic Embedded System Design Flow

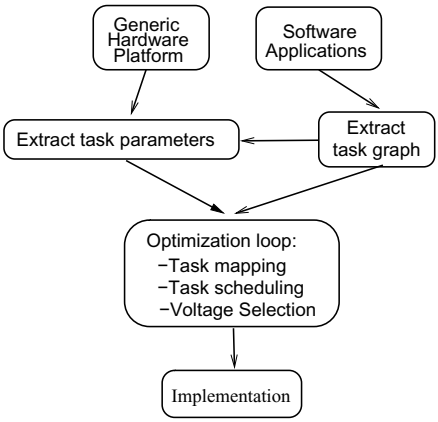


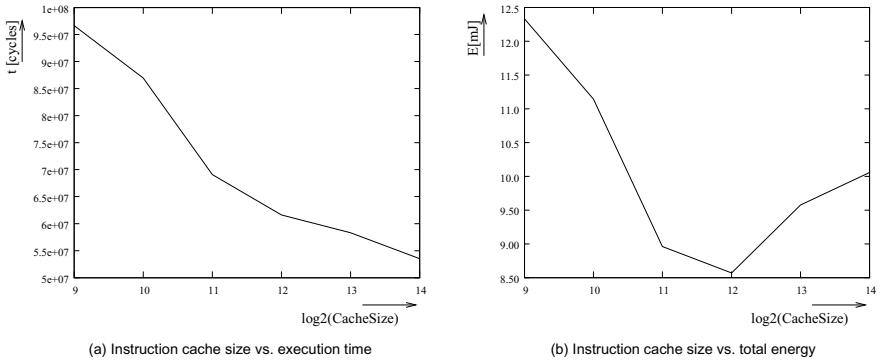
Figure 1.2: Simplified Embedded System Design Flow

design time, because embedded systems have a known functionality, as opposed to general purpose computers that must work with a variety of unknown applications.

The system level design phase is considered finished when a feasible mapped and scheduled model is produced. At this point, we can proceed with generating the software, synthesizing the custom hardware and finally producing a prototype after the integration of all the components. During this phase, before the prototype production, validation can be performed via simulation and formal verification. The validation of the prototype is performed via testing.

1.2 System Level Design

In the following we will concentrate on some of the key system level steps from the design flow introduced in Fig. 1.1. In order to simplify the explanation, let us consider a simplified flow, as illustrated in Fig. 1.2. We assume that the target embedded system consists only of programable processors and memories, interconnected by a communication infrastructure (buses, point-to-point connections or network). The starting point of the design flow is the functionality of the system, specified in a high-level programming language (such as C or C++). We also consider the hardware platform as given (possibly as a result of legacy from an earlier product). Even with the generic hardware platform fixed, some of its parameters are still subject to optimization. Such a parameter, for example, can be the size of the instruction or data cache. The selection of the size of the instruction cache can be performed by running the software application on an adequate platform simula-



(a) Instruction cache size vs. execution time (b) Instruction cache size vs. total energy

Figure 1.3: Instruction Cache Size Selection for an MP3 Decoder

tor. Fig. 1.3 presents the results obtained for an MP3 decoder running on an ARM7 processor. In Fig. 1.3(a), we present the execution time necessary to decode one MP3 frame, as a function of the size of the cache. As expected, when the cache size increases, the execution time decreases. It is interesting to note that the improvements in execution time are modest for cache sizes larger then 4kbytes. If we examine the energy values in Fig. 1.3(b), we observe that increasing the size of the instruction cache is only efficient up to a point. In case of the MP3 decoder running on the ARM7 processor, a cache of 4kbytes is optimal from the energy point of view. Smaller caches consume more energy due to a longer execution time. On the other hand, larger caches have a higher energy overhead (the energy consumed by the cache circuit itself) that cancels the potential benefits.

1.2.1 Task Graph Extraction

The task graph is extracted from the input specification (written in a high-level programming language such as C or C++). Such a task graph is illustrated in Fig. 1.4(b). Nodes $\tau_i \in \Pi$ correspond to tasks. Edges $\gamma \in \Gamma$ indicate data dependencies between these tasks. The dependencies also capture the restrictions imposed to the order of execution. An important aspect that must be highlighted at this stage is the potential parallelism between the tasks. On a multiprocessor hardware platform, tasks that are not restricted by dependencies can be executed in parallel. This leads to a shorter execution time. There are no strict rules on how to partition the code in tasks. [VJ03] presents an automatic approach for task graph extraction. A study regarding the partitioning of the MPEG2 decoder into tasks, exposing the task level parallelism is presented in [Ogn07].

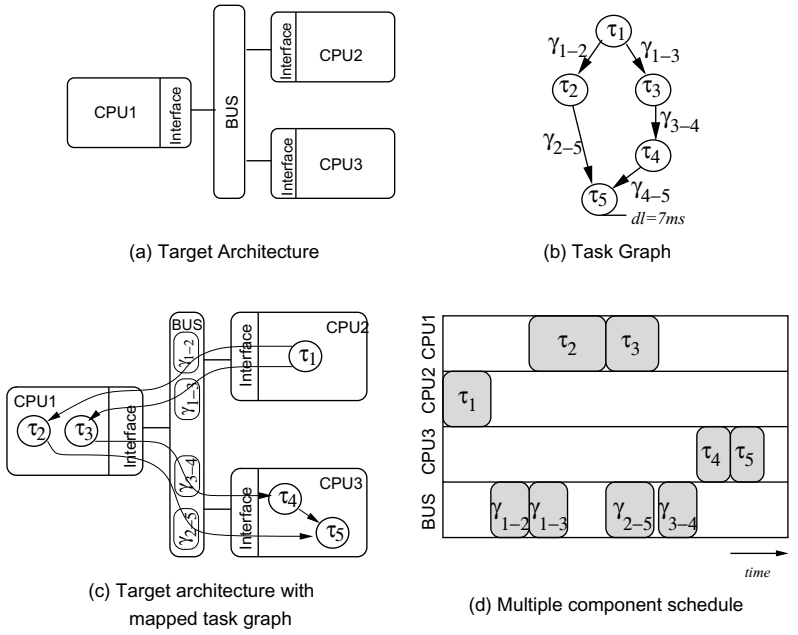


Figure 1.4: Application Mapping and Scheduling on a Target Architecture

It is important to select the ”right” granularity for the tasks, such that the right balance between the potential parallelism and the resulting number of tasks is achieved. A large number of tasks might offer an increased flexibility. However, this comes with a cost. The complexity of any system level optimization depends strongly on the number of tasks. Furthermore, the number of context switches strongly depends on the number of tasks. Thus, the size of the tasks has to be chosen such that overheads are comparatively small.

1.2.2 Task Parameters

Given the task graph and the target hardware architecture, certain properties of the tasks (the task parameters) have to be extracted. For example, for each task, two key parameters are the execution time and the power consumption. The task average power consumption can be derived via simulation. In hard real-time systems, we are interested in a particular execution time, the so called worst-case execution time. The worst-case execution time (WCET) is an upper bound of all possible execution times and is needed in order to guarantee that any possible scenario of

execution will not lead to deadline misses. While average task execution times can be derived via simulation [And06], the worst-case execution time is obtained by performing worst-case execution time analysis [PB00, TFW00, RM05, SSE05]. In real-time systems, where delivering a result within a specified time frame is an intrinsic aspect of the correct functionality, worst-case execution time analysis is a key issue. In Part III of this thesis we will further explore this topic.

1.2.3 Task Mapping and Scheduling

Given a task graph (Fig. 1.4(b)) and a target hardware platform (Fig. 1.4(a)), the designer has to map and schedule the tasks on the processors. Mapping is the step in which the tasks are assigned for execution to the processors and the communications to the bus(es). In Fig. 1.4(c), we have depicted a possible mapping for the task graph in Fig. 1.4(b). The next step is to compute a schedule for the system. In the case of static cyclic scheduling this implies to decide in which order to run the tasks mapped on the same processor. One important set of constraints that have to be respected during mapping and scheduling are the precedence constraints given by the dependencies in the task graph. An example schedule is depicted in Fig. 1.4(d). Please note that task τ_2, for example, starts only after task τ_1 and the communication γ_{1-2} have finished. Most embedded applications must also respect the real-time constraints, such as the application deadline. Computing the task mapping and schedule for a set of tasks with precedence constraints on a multiprocessor architecture is in general an NP complete problem [GJ79]. Nevertheless many algorithms have been proposed to solve the problem [VM03, HM03, SHE05, SAHE04, SAHE02, DJ98, DJ99, ACD74, WG90, OH96, PP92, SL93, KA99, BJM97, BGM+06, RGA+06]. Some of the approaches propose exact, optimal solutions, while others are heuristics producing suboptimal results. In Chapter 4 we will present two approaches where on top of guaranteeing the timing constraints, the objective of minimizing the energy consumption is added.

We illustrate the relation between mapping and the energy consumption, using an MPEG2 decoder that has to be implemented on a multiprocessor platform. The number of ARM7 processor cores, as well as the voltage/frequency of the platform can be statically configured. From the energy perspective, a low clock speed is desirable. The real-time constraint is to finish decoding each video frame in 40ms. The design space exploration has to decide between using many processors at a low voltage/frequency or few processors that run fast. The parallelism of the application is key in selecting the right configuration. The results are presented in Fig. 1.5. Fig. 1.5(a) presents the normalized execution time as a function of the number of processors. The execution time for decoding one frame for each core

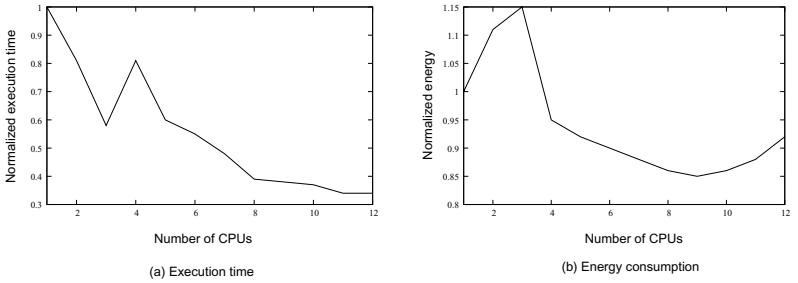


Figure 1.5: Design Space Exploration for an MPEG2 Decoder

count is normalized against the execution time obtained for the execution on one single processor. We notice that there is no strict monotonicity relation between the number of processors and the resulting execution time. Nevertheless, the execution time can be improved by more then 60% if more then 8 cores are used. The energy consumption achieved for each number of processors is shown in Fig. 1.5(b). The energy obtained for a certain number of processors is normalized against the energy consumed by a single processor implementation. For each number of processors, experiments were performed using several frequencies of the platform. The results with the lowest energy are the ones reported in Fig. 1.5(b). We observe that using 9 processors provides the best energy savings. When using a smaller number of processors, the platform has to be clocked a higher frequency/voltage and thus consumes more. Adding more processors, due to the extra hardware and the fact that there is no more parallelism to exploit, results in increased energy.

1.3 Energy Optimization

The number of battery powered embedded devices as well as their complexity continues to grow. In [ITR] it is projected that the amount of power required by new devices increases by 35-40% per year. However, the capacity of the batteries increases by only 10-15% per year, leaving a gap that must be filled by various optimization techniques. Energy can be improved at various stages during the embedded system design flow, from the system level, down to the circuit level [AMR+06, SAHE04, BD00]. In this thesis we will concentrate on energy optimization techniques at the system level.

Although, until recently, the dynamic power dissipation has been dominating, the trend to reduce the overall circuit supply voltage and, consequently, threshold voltage, is raising concerns about the leakage currents [Bor99, KR02, MFMB02,

HASM+03]. In this thesis we propose algorithms that target the minimization of both dynamic and leakage energy.

In the previous section, we have shown that energy consumption can be reduced by an intelligent mapping of the tasks to the processors. Even with a good mapping, the energy consumption can be further optimized. During architecture selection and mapping, the best processors that can provide the required performance are selected. Nevertheless, due to a finite set of available processors, the selected ones are always more powerful then required. Furthermore, many applications have a variable execution time, but the hardware has to be powerful enough to accommodate the worst-case scenario. Thus, a certain amount of slack is present in the task schedules. We will present in this thesis algorithms that are exploiting this slack and thus, reduce the energy consumption.

1.4 Contributions

In the vast context of system-level design of embedded systems, the contributions of this thesis are the following:

1. Offline energy minimization technique:
   (a) We consider both supply voltage and body-bias voltage selection at the system-level, where several tasks with dependencies execute a time-constrained application on a multiprocessor system.
   (b) Four different voltage selection schemes are formulated as nonlinear programming (NLP) and mixed integer linear programming (MILP) problems which can be solved optimally. The formulations are equally applicable to single and multiprocessor systems.
   (c) We prove that discrete voltage selection with and without the consideration of transition overheads in terms of energy and time is strongly NP-hard, while the continuous voltage selection cases can be solved in polynomial time (with an arbitrary given approximation $\varepsilon > 0$).
   (d) We solve the combined voltage selection problem for processing elements and communications links. To allow an effective voltage selection on the communication links, we outline a set of delay and energy models. Further, we take into account the possibility of dynamic voltage swing scaling on fat wires and address the leakage power dissipation in bus repeaters.
   (e) Since voltage selection for components that operate with discrete voltages is proofed to be NP-hard, we introduce a simple yet effective

heuristic based on the NLP formulation for the continuous voltage selection problem.

(f) We study the combined voltage selection and processor shutdown problem. In particular, we demonstrate that the processor shutdown is an NP complete problem even isolated from the voltage selection. We propose two solutions that integrate the shutdown with the continuous and respectively with the discrete voltage selection.

2. Online energy minimization technique:

   (a) Two quasi-static voltage selection algorithms for multi-task applications are proposed. Both continuous and discrete voltage selection are investigated.

   (b) We propose online algorithms for systems consisting of both single and multiprocessors

   (c) We perform an evaluation of the impact of the overhead of different dynamic voltage scaling approaches on realistic applications.

3. Predictability

   (a) We identify the inaccuracies of classical worst-case execution time analysis techniques when used for the analysis of tasks implemented on multiprocessor platforms with a shared bus.

   (b) We propose a TDMA-based bus scheduling policy that provides a predictable bus access.

   (c) We propose a new framework that integrates system level task scheduling, bus access optimization and worst-case execution time analysis for real-time applications implemented on multiprocessor systems.

1.5 List of papers

Parts of the contents of this dissertation have been presented in the following papers:

- [AERP07]: Alexandru Andrei, Petru Eles, Zebo Peng, Jakob Rosen "Predictable Implementation of Real-Time Applications on Multiprocessor Systems on Chip", submitted.

- [AEP+07b]: Alexandru Andrei, Petru Eles, Zebo Peng, Marcus Schmitz, Bashir Al-Hashimi "Voltage Selection for Time-Constrained Multiprocessor

Systems on Chip", chapter in "Designing Embedded Processors: A Low Power Perspective", pages 259-284, edited by J. Henkel, S.Parameswaran, Springer 2007.

- [AEP+07a]: Alexandru Andrei, Petru Eles, Zebo Peng, Marcus Schmitz, Bashir Al-Hashimi "Energy Optimization of Multiprocessor Systems on Chip by Voltage Selection", IEEE Transactions on Very Large Scale Integration Systems, volume 15, number 3, pages 262-275, March, 2007.

- [RGA+06]: Martino Ruggiero, Pari Gioia, Guerri Alessio, Luca Benini , Michela Milano, Davide Bertozzi, Alexandru Andrei "A Cooperative, Accurate Solving Framework for Optimal Allocation, Scheduling and Frequency Selection on Energy-Efficient MPSoCs", International Symposium on System on Chip, pages 1-4, 2006, Tampere, Finland.

- [ASE+05a]: Alexandru Andrei, Marcus Schmitz, Petru Eles, Zebo Peng, Bashir Al-Hashimi "Overhead-Conscious Voltage Selection for Dynamic and Leakage Energy Reduction of Time-Constrained Systems", IEE Proceedings Computers & Digital Techniques, special issue with the best contributions from the DATE 2004 Conference, Volume 152, Issue 01, pages 28-38, January, 2005

- [ASE+05b]: Alexandru Andrei, Marcus Schmitz, Petru Eles, Zebo Peng, Bashir Al-Hashimi "Quasi-Static Voltage Scaling for Energy Minimization with Time Constraints", Design Automation and Test in Europe (DATE), pages 514-519, 2005, Munchen, Germany.

- [ASE+04b]: Alexandru Andrei, Marcus Schmitz, Petru Eles, Zebo Peng, Bashir Al-Hashimi "Simultaneous Communication and Processor Voltage Scaling for Dynamic and Leakage Energy Reduction in Time-Constrained Systems", The International Conference on Computer Aided Design (ICCAD), pages 362-369, 2004, San Jose, USA.

- [ASE+04a]: Alexandru Andrei, Marcus Schmitz, Petru Eles, Zebo Peng, Bashir Al-Hashimi "Overhead-Conscious Voltage Selection for Dynamic and Leakage Energy Reduction of Time-Constrained Systems", Design Automation and Test in Europe (DATE), pages 518-523, 2004, Paris, France.

Other papers where the author of the thesis was involved:

- [RAEP07]: Jakob Rosen, Alexandru Andrei, Petru Eles, Zebo Peng "Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems on Chip", Real-Time Systems Symposium (RTSS), 2007, Tucson, USA.

- [And06]: Alexandru Andrei "System Design of Embedded Systems Running on an MPSoC Platform", Technical report Linkoping University, 2006.
- [PPE+06]: Traian Pop, Paul Pop, Petru Eles, Zebo Peng, Alexandru Andrei "Timing Analysis of the FlexRay Communication Protocol", Euromicro Conference on Real-Time Systems (ECRTS), 2006, pages 203-213 ,Dresden, Germany.
- [ASEP04] Alexandru Andrei, Marcus Schmitz, Petru Eles, Zebo Peng, Bashir Al-Hashimi "Simultaneous Communication and Processor Voltage Scaling for Energy Reduction in Time-Constrained Systems", Power Aware Real-Time Computing Workshop (PARC), 2004, Pisa, Italy.

1.6 Thesis organization

The thesis is organized as follows. In the first Part, in Chapter 1, we present a generic design flow for real-time embedded systems. This design flow serves as a general framework for the following parts.

In Part II, we define energy minimization as a problem for today's battery operated embedded systems. Chapter 2 gives an overview to energy/speed trade-offs in general and introduces supply voltage scaling and adaptive body biasing as the two techniques that can be used efficiently at the system level in order to minimize the energy consumption. The energy minimization problem is addressed with offline and online algorithms. In Chapter 3 we solve optimally the combined supply voltage and body bias selection problem for multiprocessor systems with imposed time constraints, explicitly taking into account the transition overheads implied by changing voltage levels. Moreover, we show that voltage selection can be applied not only to processors, but also to the communication infrastructure.

The mapping of the tasks on the processors and the schedule have a big impact on the achievable energy savings. In Chapter 4, we present an integrated approach. The algorithms described in Chapter 3 are used within two system level optimization frameworks that perform architecture selection, task mapping and scheduling.

The previously mentioned approaches belong to the offline category. The optimization is performed at design time, assuming worst-case execution times. However, many applications exhibit variations of their execution time, which lead to a certain amount of dynamic slack, that is known only during runtime. In order to exploit this additional slack, an online recalculation of the voltages is needed. We present in Chapter 5 such an approach. Since the complexity of any online algorithm is critical, we propose a quasi-static solution that calculates offline the task voltages for several possible execution times and stores them in look-up tables.

The online algorithm is using the precalculated values from the look-up table, depending on the actual execution times.

In Part III, Chapter 6, we identify the estimation of the worst-case execution time as a potential problem for systems with several processors and memories connected by a shared bus. In this context, we propose an approach to worst-case execution time analysis and system scheduling for real-time applications.

Part IV presents the conclusions and sketches the future work.

Part II

Energy Minimization by Voltage Selection

Chapter 2

Introduction

An obvious trend in the last years is to pack more and more functionality into smaller and smaller electronic devices. A typical example are mobile phones with digital cameras and media players. This leads to an increase in the amount of power needed to run all these applications. Since a large fraction of such embedded systems are powered by batteries, energy consumption becomes a major design issue. The gap between the amount of power provided by advances in battery technologies and the power demanded by new functionality is increasing. This motivates the work on energy minimization techniques presented in Chapters 3, 4, 5.

2.1 Energy/Speed Trade-off

Embedded computing systems need to be energy efficient, yet they have to deliver adequate performance to computational expensive applications, such as voice processing and multimedia. Energy minimization can be performed at different levels during the design. We have shown in Chapter 1 how mapping can be used to improve the energy consumption. Another orthogonal approach is based on the fact that the workload imposed on an embedded system is non-uniform over time. This introduces slack times during which the system can reduce its performance and thus save energy.

Let us examine the schedule depicted in Fig. 2.1(a) (obtained for the task graph from Fig. 1.4). If the tasks are running at the highest speed, τ_5 finishes before the deadline *dl* and thus reveals a certain amount of slack. In real-time systems, the task execution times must not exceed their deadlines, but there is no reward for fin-

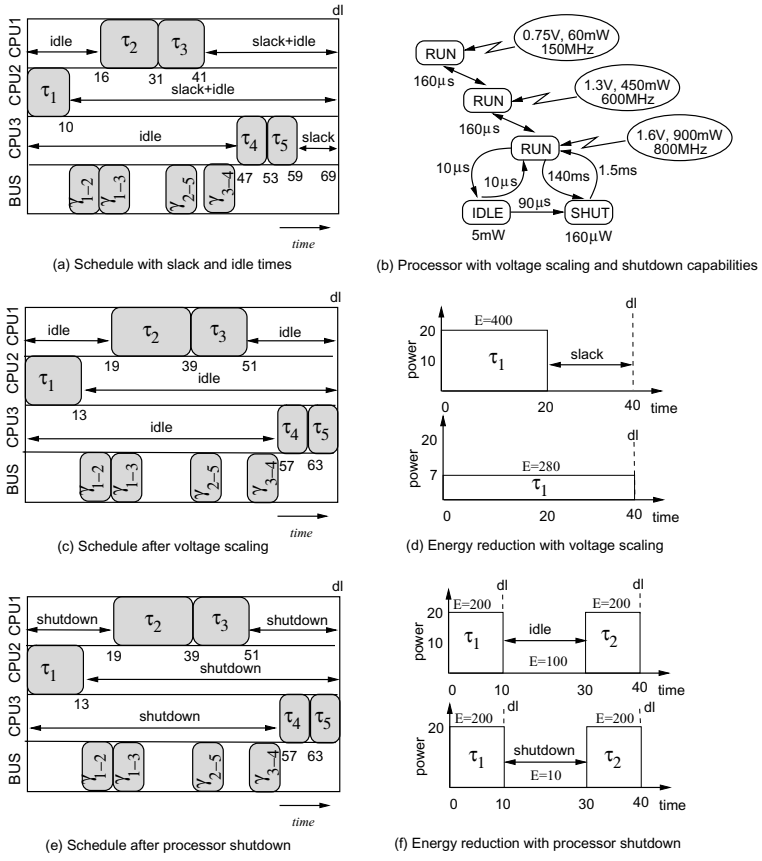


Figure 2.1: Schedule with Idle and Slack Times

ishing earlier. On the other hand, due to the dependencies, task τ_2 running $CPU1$ can start only after the message γ_{1-2} sent at the end of τ_1 is transmitted. This results in a certain amount of idle time on $CPU1$, from time 0 until 16, when τ_2 can be started. The slack and idle times are key factors that influence the achievable energy savings. Many processors produced today (general purpose mobile processors as well as embedded ones) have the capability to dynamically change their frequency [Kla00, pow00, xsc00] at runtime. Using a high frequency results in faster execution times and a higher power consumption then using lower frequencies. Moreover, during idle periods when no instruction has to be executed, it is possible to save the current state of the processor, shut it down in order to

save the energy and then restart executing. A simplified diagram of the possible power states of such a processor (Intel Xscale [xsc00]) is depicted in Fig.2.1(b). Tasks can be executed using 3 performance modes. Each mode is characterized by a certain frequency (800, 600, 150MHz) and a corresponding power consumption (900, 450, 60mW). At runtime, any combination of these modes can be used to execute a task. Switching between two performance modes comes with a certain time and energy penalty. Two other states can be used when the processor is not executing any task. If the first one (Idle) is used, the processor consumes 5mW, as opposed to the lowest power consumption of 60mW that can be achieved during the execution of a task. During this state, clock gating is activated, and so there is no switching activity in the processor. The overhead associated with a transition to this state is very small. If the period when the processor is not executing any task is longer, there exists a state when it consumes only $160muW$. The overhead associated with switching to this state (140ms) is high, so it must be used only after a careful analysis.

The usage of voltage scalable processors opens the possibility for various energy/speed trade-offs. We will show in the following how to exploit the available slack and idle times in order to reduce the energy consumption, in the context of real-time systems. Throughout the thesis, we will use the terms voltage scaling, voltage selection, frequency scaling and frequency selection interchangeably.

Let us focus on the example depicted in Fig. 2.1(d). Task τ_1 is executed at 100MHz and finishes in the worst-case at 20ms, while its deadline is 40ms. The power consumption at 100MHz is 20mW, resulting in an energy consumption for τ_1 of $400\mu J$. If voltage scaling is performed and τ_1 is executed at $50MHz$, it finishes exactly at the deadline, using all the available slack. With a power consumption of 7mW at 50MHz, an energy of $280\mu J$ is consumed, 30% less then the nominal case. Performing voltage scaling for a multi-task system is a complex issue, due to the potential dependencies between the tasks that influence the distribution of the slack. Let us consider performing voltage scaling for the schedule in Fig. 2.1(a). Please note that τ_5 finishes its execution at time 59, before the deadline that is set at 69 and thus yielding a slack of 10 time units. This slack can be exploited by voltage scaling. The question that needs to be addressed at this point is how to distribute this slack among the 5 tasks. Fig. 2.1(c) shows one possibility. τ_1 executed at a lower frequency, finishes in 13 time units instead of 10 at the nominal frequency. τ_2 that needs 15 time units at the nominal frequency uses 20 time units at a lower frequency. τ_3 is extended with 3 time units. If we propagate the dependencies and calculate the new end times, we observe that the deadline is met, but tasks τ_4 and τ_5 cannot be scaled. We will present in Chapter 3, both optimal and heuristic algorithms for the voltage selection problem.

The examples from Fig. 2.1(c) and (d) have illustrated the efficiency of voltage scaling for the minimization of the energy consumed by the tasks. We will refer to this energy in the following as active energy. Let us focus now on the minimization of the energy that is consumed when the processor is not running any task. A small example is depicted in Fig. 2.1(f). Let us assume that τ_1 has a deadline at 10ms, τ_2 can start at 30ms and must finish at 40ms. As a result, the processor is idle (not running any task) between 10 and 30ms. Assuming that during idle times the processor consumes 5mW, the energy spent idling is $100\mu J$. If the processor can be shut down during this time, energy is consumed only to save and later restore the state of the processor. In our case this energy is $10\mu J$. So overall, by shutting down the processor we save 18% of the total energy.

An examination of the schedule resulted after performing voltage scaling from Fig. 2.1(c), shows that even if there is no more slack, there exists a certain amount of idle time on each of the 3 processors. If the idle times are long enough (ie. the achievable savings are higher than the shutdown overhead), the energy can be minimized if the processors are shutdown during these time intervals. The resulting schedule is illustrated in Fig. 2.1(e). In general, deciding when to shutdown and furthermore, the integration of voltage scaling with processor shutdown is not trivial. An efficient algorithm is presented in Chapter 3.

2.2 Voltage Selection Techniques

Two system-level approaches that allow an energy/performance trade-off during run-time of the application are dynamic voltage selection (DVS) [IY98, MFMB02, YDS95] and adaptive body biasing (ABB) [KR02, MFMB02]. While DVS aims to reduce the dynamic power consumption by scaling down operational frequency and circuit supply voltage V_{dd}, ABB is effective in reducing the leakage power by scaling down frequency and increasing the threshold voltage V_{th} through body-biasing. Up to date, most research efforts at the system-level were devoted to DVS, since the dynamic power component *had been* dominating. Nonetheless, the trend in deep-submicron CMOS technology to reduce the supply voltage levels and consequently the threshold voltages (in order to maintain peak performance) is resulting in the fact that a substantial portion of the overall power dissipation will be due to leakage currents [Bor99, KR02]. This makes the adaptive body-biasing approach and its combination with dynamic voltage selection attractive for energy-efficient designs in the foreseeable future.

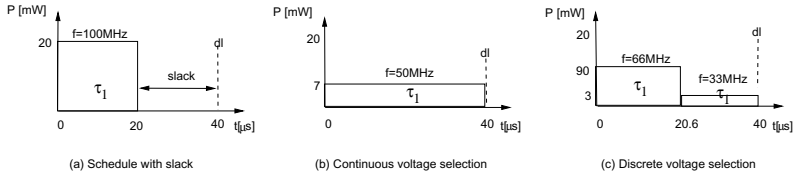


Figure 2.2: Continuous and Discrete Voltage Selection

2.3 Offline and Online Voltage Selection

Voltage selection approaches can be broadly classified into online and offline techniques.

Offline techniques perform the optimization statically. This is useful for real-time systems, where one of the most important issues is guaranteeing that the timing constraints are met. In the context of voltage selection, offline means that the calculation of the voltages to be assigned to each task is performed at design time. These values are then used, without any additional computational effort, at runtime. The fact that the optimization is performed before runtime has several advantages. First, even if long optimization times are not desired, they can often be afforded. So, complex algorithms can be used. In many cases, the computer system where the optimization is performed is powerful, as opposed to the target embedded system. However, offline optimizations have disadvantages. The most important is the lack of flexibility. Let us assume, for example, that voltage selection was performed offline for a real-time system. In order to guarantee the correct timing, worst-case execution time had to be used for each task. However, at runtime most of the tasks finish before their estimated worst-case. This creates a certain amount of dynamic slack, known only at runtime, that is not exploited by the voltages calculated offline. In order to exploit this dynamic slack, an online recalculation of the voltages is needed. Since this calculation is performed at runtime, it has to be very efficient. We will present both offline and online approaches in Chapters 3 and 5.

2.4 Continuous and Discrete Voltage Selection

Depending on the assumption regarding the scale of available voltages and frequencies on the target processor, two voltage selection problems are formulated. First, if the task voltages and frequencies can be chosen within a continuous interval, the resulting problem is called continuous voltage selection. Second, if the variables can be selected from a discrete set, the problem is called discrete voltage selection.

These two flavors are illustrated with the example from Fig. 2.2. Fig. 2.2(a) shows the execution of the task τ_1 at the nominal speed of 100MHz. With a worst-case number of 2000 clock cycles, τ_1 finishes at 20μs, before the deadline at 40μs. If continuous voltage scaling is used, the frequency is selected for τ_1 such that it finishes exactly at the deadline, like in Fig. 2.2(b). For 2000 clock cycles that are executed in 40μs, a frequency of 50MHz is needed.

Discrete voltage selection is illustrated in Fig. 2.2(c). Let us assume that the processor is capable of operating in three different performance modes, using 3 discrete frequencies: 100MHz, 66MHz and 33MHz. Moreover, τ_1 is executed cycle by cycle, and, during each cycle, a different frequency can potentially be used. [IY98] presents a heuristic for the calculation of the performance modes and the corresponding number of clock cycles for a task, given an available execution time. After the calculation of the optimal voltage assuming the continuous case, it proposes the usage of the two voltages corresponding to frequencies that surround the continuous one. For the example in Fig. 2.2, where the calculated continuous frequency is 50MHz, the discrete modes are 66 and 33MHz. In order to calculate the number of clock cycles to be executed in each of these modes, a system of two equations has to be solved:

$$\frac{NC_1}{66} + \frac{NC_2}{33} = 40$$
$$NC_1 + NC_2 = 2000$$

The first equation establishes that the times executed in the two modes has to sum up to the available execution time for the task. The second equation states that all the task's clock cycles have to be distributed between the two modes. For the example from Fig. 2.2, it will result in 1640 clock cycles to be executed at 66MHz and 370 clock cycles to be executed at 33MHz.

For systems consisting of more then one task, with possible dependencies and a different amount of power consumed by each task, the voltage selection problem is not trivial. This classification in continuous and discrete voltage selection was done due to complexity reasons. While real processors can operate using a discrete range of performance modes, computationally, the continuous voltage selection algorithms are easier (polynomial) then their discrete counterparts (NP hard). These aspects will be addressed in Chapter 3.

Chapter 3

Offline Energy Optimization by Voltage Selection

Dynamic voltage selection and adaptive body biasing have been shown to reduce dynamic and leakage power consumption effectively. In this chapter, we restrict to offline techniques, where the scaled supply voltages are calculated at design time and then applied at run-time according to the pre-calculated voltage schedule. We present an optimal approach for the combined supply voltage and body bias selection problem for multiprocessor systems with imposed time constraints, explicitly taking into account the transition overheads implied by changing voltage levels. Both energy and time overheads are considered. The voltage selection technique achieves energy efficiency by simultaneously scaling the supply and body bias voltages in the case of processors and buses with repeaters, while energy efficiency on fat wires is achieved through dynamic voltage swing scaling. We investigate the continuous voltage selection as well as its discrete counterpart, and we prove strong NP-hardness in the discrete case. Furthermore, the continuous voltage selection problem is solved using nonlinear programming with polynomial time complexity, while for the discrete problem we use mixed integer linear programming and a polynomial time heuristic. We propose an approach that combines voltage selection and processor shutdown in order to optimize the total energy.

3.1 Related Work

There has been a considerable amount of work on dynamic voltage selection. Yao et al. [YDS95] proposed the first DVS approach for single processor systems

which can change the supply voltage over a continuous range. Ishihara and Yasuura [IY98] modeled the discrete voltage selection problem using an integer linear programming (ILP) formulation. Kwon and Kim [KK05] proposed a linear programming (LP) solution for the discrete voltage selection problem with uniform and non-uniform switched capacitance. Although this work gives the impression that the problem can be solved optimally in polynomial time, we will show in this chapter that the discrete voltage selection problem is indeed strongly NP-hard and, hence, no optimal solution can be found in polynomial time, for example using LP. Dynamic voltage selection has also been successfully applied to heterogeneous distributed systems, mostly using heuristics [GK01, LJ03, SAH01]. Zhang et al. [ZHC02] approached continuous supply voltage selection in distributed systems using an ILP formulation. They solved the discrete version of the problem through an approximation.

While the approaches mentioned above scale only the supply voltage V_{dd} and neglect leakage power consumption, Kim and Roy [KR02] proposed an adaptive body-biasing approach (in their work referred to as dynamic V_{th} scaling) for active leakage power reduction. They demonstrate that the efficiency of ABB will become, with advancing CMOS technology, comparable to DVS. Duarte et al. [DVI$^+$02] analyze the effectiveness of supply and threshold voltage selection, and show that simultaneously adjusting both voltages provides the highest savings. Martin et al. [MFMB02] presented an approach for combined dynamic voltage selection and adaptive body-biasing. At this point we should emphasize that, as opposed to these three approaches, we investigate in this chapter how to select voltages for a set of tasks, possibly with dependencies, which are executed on multiprocessor systems under real-time constraints. Furthermore, as opposed to our work, the techniques mentioned above *neglect* the energy and time overheads imposed by voltage transitions. Noticeable exceptions are [HQPS98, MHQ02, MHQ07, ZHC03], yet their algorithms ignore leakage power dissipation and body-biasing, and further they do not guarantee optimality. In this work, we consider simultaneous supply voltage selection and body biasing, in order to minimize dynamic as well as leakage energy. In particular, we investigate four different notions of the combined dynamic voltage selection and adaptive body-biasing problem — considering continuous and discrete voltage selection with and without transition overheads. A similar problem for continuous voltage selection has been formulated in [YLJ05]. However, it is solved using a suboptimal heuristic. The combination of dynamic supply voltage selection and processor shutdown was presented in [RJ05] for single processor systems. The authors demostrate the existence of a critical speed, under which scaling the processor frequency becomes energy inefficient, due to the fact that the leakage energy increases faster than the dynamic energy

decreases. The leakage energy reduction is achieved there by shutting down the processor during the idle intervals, without performing adaptive body biasing.

To fully exploit the potential performance provided by multiprocessor architectures (e.g. systems-on-a-chip), communication has to take place over high performance buses, which interconnect the individual components, in order to prevent performance degradation through unnecessary contention. Such global buses require a substantial portion of energy, on top of the energy dissipated by the computational components [Sve01, SK01]. The minimization of the overall energy consumption requires the combined optimization of both the energy dissipated by the computational processors as well as the energy consumed by the interconnection infrastructure.

A negative side-effect of the shrinking feature sizes is the increasing *RC* delay of on-chip wiring [IF99, SK01]. The main reason behind this trend is the ever-increasing line resistance. In order to maintain high performance it becomes necessary to "speed-up" the interconnects. Two implementation styles which can be applied to reduce the propagation delay are: (a) The insertion of *repeaters* and (b) the usage of *fat wires*. In principle, repeaters split long wires into shorter (faster) segments [IF99, KCS02, SK01, CTH05] and fat wires reduce the wire resistance [Sve01, SK01]. Techniques for the determination of the optimal quantity of repeaters are introduced in [IF99, KCS02]. An approach to calculate the optimal voltage swing on fat wires has been proposed in [Sve01]. Similar to processors with supply voltage selection capability, approaches for link voltage scaling were presented in [SPJ02, WKL+00]. An approach for communication speed selection was outlined in [LCB02]. Another possibility to reduce communication energy is the usage of bus encoding techniques [BMM+98]. In [HP02], it was demonstrated that shared-bus splitting, which dynamically breaks down long, global buses into smaller, local segments, also helps to improve energy savings. An estimation framework for communication switching activity was introduced in [FSS99].

Until now, energy estimation for system-level communication was treated in a largely simplified manner [LCB02, VM03] and based on naive models that ignore essential aspects such as bus implementation technique (repeaters, fat wires), leakage power, and voltage swing adaption. This, however, very often leads to oversimplifications which affect the correctness and relevance of the proposed approaches and, consequently, the accuracy of results. On the other hand, issues like optimal voltage swing and increased leakage power due to repeaters are not considered at all for implementations of voltage-scalable embedded systems.

As mentioned earlier, in this chapter we will concentrate on off-line voltage selection techniques, that make use of the static slack existing in the application. In Chapter 5 we present an efficient technique that dynamically makes use of slack

created online, due to the fact that tasks execute less then their worst case number of clock cycles.

The remainder of this chapter is organized as follows: Preliminaries regarding the system specification, the processor power and delay models are given in Sections 3.2 and 3.3. This is followed by a motivational example in Section 3.4. The four investigated processor voltage selection problems are formulated in Section 3.5. Continuous and discrete voltage selection problems are discussed in Sections 3.6 and 3.7, respectively. We study the combined voltage selection and shutdown problem in Section 3.8. Power and delay models for the communication links are given and the general problem of voltage selection for processors and the communication is addressed in Section 3.9. Extensive experimental results are presented in Section 3.10.

3.2 System and Application Model

We consider embedded systems which are realized as heterogeneous distributed architectures. Such architectures consist of several different processing elements (PEs), such as programmable microprocessors, ASIPs, FPGAs, and ASICs, some of which feature DVS and ABB capability. These computational components communicate via an infrastructure of communication links (CLs), like buses and point-to-point connections. We define $\mathcal{P}$ and $\mathcal{L}$ to be the sets of all processing elements and all links, respectively. An example architecture is shown in Fig. 1.4(a). The functionality of applications is captured by task graphs $G(\Pi, \Gamma)$, as in Fig. 1.4(b). Nodes $\tau \in \Pi$ in these directed acyclic graphs represent computational tasks, while edges $\gamma \in \Gamma$ indicate data dependencies between these tasks (communications). Tasks τ_i require in the worst case WNC_i clock cycles to be executed, depending on the PE to which they are mapped. Further, tasks are annotated with deadlines dl_i that have to be met at run-time.

If two dependent tasks are assigned to different PEs, p_x and p_y with $x \neq y$, then the communication takes place over a CL, involving a certain amount of time and power.

We assume that the task graph is mapped and scheduled on the target architecture, i.e., it is known where and in which order tasks and communications take place. Fig. 1.4(c) shows the task graph from Fig. 1.4(b) that has been mapped onto the architecture in Fig. 1.4(a). Fig. 1.4(d) depicts a possible execution order.

To tie the execution order into the application model, we perform the following transformation on the original task graph. First, all communications that take place over communication links are captured by communication tasks, as indicated by squares in Fig. 3.1. For instance, communication γ_{1-2} is replaced by task τ_6 and

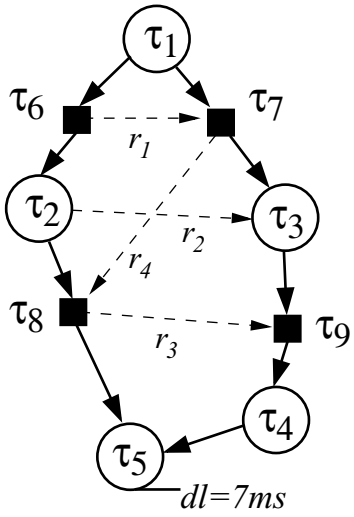


Figure 3.1: System model: Extended task graph

the edges connecting τ_6 to τ_1 and τ_2 are introduced. $\mathcal{K}$ defines the set of all such communication tasks and $\mathcal{C}$ the set of graph edges obtained after the introduction of the communication tasks. Furthermore, we denote with $\mathcal{T} = \Pi \cup \mathcal{K}$ the set of all computations and communications. Second, on top of the precedence relations given by data dependencies between tasks, we introduce additional precedence relations $r \in \mathcal{R}$, generated as result of scheduling tasks mapped to the same PE and communications mapped on the same CL. In Fig. 3.1, corresponding to the initial task graph from Fig. 1.4(b) and the schedule from Fig. 1.4(d), the dependencies $\mathcal{R}$ are represented as dotted edges. We define the set of all edges as $\mathcal{E} = \mathcal{C} \cup \mathcal{R}$. We construct the mapped and scheduled task graph $G(\mathcal{T}, \mathcal{E})$. Further, we define the set $\mathcal{E}^\bullet \subseteq \mathcal{E}$ of edges, as follows: an edge $(i, j) \in \mathcal{E}^\bullet$ if it connects task τ_i with its immediate successor τ_j (according to the schedule), where τ_i and τ_j are mapped on the same PE or CL.

3.3 Processor Power and Delay Models

Digital CMOS circuitry has two major sources of power dissipation: (a) dynamic power P_{dyn}, which is dissipated whenever active computations are carried out (switching of logic states), and (b) leakage power P_{leak} which is consumed whenever the circuit is powered, even if no computations are performed. The dynamic power

is expressed by [CB95, MFMB02]:

$$P_{dyn} = C_{eff} \cdot f \cdot V_{dd}^2 \tag{3.1}$$

where C_{eff}, f, and V_{dd} denote the effective charged capacitance, operational frequency, and circuit supply voltage, respectively. Although, until recently, dynamic power dissipation has been dominating, the trend to reduce the overall circuit supply voltage and consequently threshold voltage is raising concerns about the leakage currents. For near future technology ($< 65nm$) it is expected that leakage will account for a significant part of the total power. The leakage power is given by [MFMB02]:

$$P_{leak} = L_g \cdot V_{dd} \cdot K_3 \cdot e^{K_4 \cdot V_{dd}} \cdot e^{K_5 \cdot V_{bs}} + |V_{bs}| \cdot I_{Ju} \tag{3.2}$$

where V_{bs} is the body-bias voltage and I_{Ju} represents the body junction leakage current (constant for a given technology). The fitting parameters K_3, K_4 and K_5 denote circuit technology dependent constants and L_g reflects the number of gates. For clarity reasons we maintain the same indices as used in [MFMB02], where also actual values for these constants are given. Please note that the leakage power is stronger influenced by V_{bs} than by V_{dd}, due to the fact that the constant K_5 is larger than the constant K_4 (e.g., for the Crusoe processor described in [MFMB02], $K_5 = 4.19$ while $K_4 = 1.83$).

Nevertheless, scaling the supply and the body-bias voltage for power saving, has a side-effect on the circuit delay d and hence the operational frequency [CB95, MFMB02]:

$$f = \frac{1}{d} = \frac{((1+K_1) \cdot V_{dd} + K_2 \cdot V_{bs} - V_{th1})^{\alpha}}{K_6 \cdot L_d \cdot V_{dd}} \tag{3.3}$$

where α reflects the velocity saturation imposed by the used technology (common values $1.4 \leq \alpha \leq 2$), L_d is the logic depth, and K_1, K_2, K_6 and V_{th1} are circuit dependent constants.

Another important issue, which often is overlooked, is the consideration of transition overheads, i.e., each time the processor's supply and body bias voltage are altered, the change requires a certain amount of extra energy and time. These energy $\varepsilon_{k,j}$ and delay $\delta_{k,j}$ overheads, when switching from V_{dd_k} to V_{dd_j} and from V_{bs_k} to V_{bs_j}, are given by: [MFMB02],

$$\varepsilon_{k,j} = C_r \cdot |V_{dd_k} - V_{dd_j}|^2 + C_s \cdot |V_{bs_k} - V_{bs_j}|^2 \tag{3.4}$$

$$\delta_{k,j} = \max(p_{Vdd} \cdot |V_{dd_k} - V_{dd_j}|, p_{Vbs} \cdot |V_{bs_k} - V_{bs_j}|) \tag{3.5}$$

where C_r denotes power rail capacitance, and C_s the total substrate and well capacitance. Since transition times for V_{dd} and V_{bs} are different, the two constants p_{Vdd}

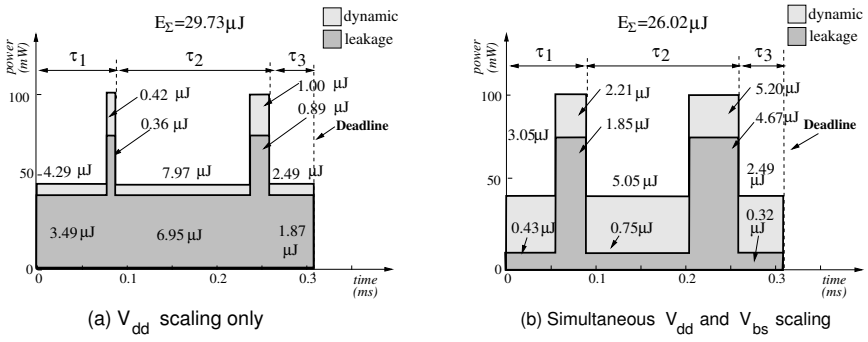


Figure 3.2: Influence of V_{bs} scaling

and p_{Vbs} are used to calculate both time overheads independently. Considering that supply and body-bias voltage can be scaled in parallel, the transition overhead $\delta_{k,j}$ depends on the maximum time required to reach the new voltage levels.

In the following, we assume that the processors can operate in several execution modes. An execution mode m_z is characterized by a pair of supply and body bias voltages: $m_z = (V_{dd_z}, V_{bs_z})$. As a result, an execution mode has an associated frequency and power consumption (dynamic and leakage) that can be calculated using Eq. 3.3 and respectively Eq. 3.1 and 3.2. Upon a mode change, the corresponding delay and energy penalties are computed using Eq. 3.5 and 3.4.

Tasks that are mapped on different processors communicate over one or more shared buses. In Sections 3.4-3.8 we assume that the buses are not voltage scalable and thus working at a given frequency. Each communication task has a fixed execution time and energy consumption depeding proportionally on the amount of communication. For simplicity of the explanations, in Sections 3.4-3.8 we will not differentiate between computation and communication tasks. A more refined communication model, as well as the benefits of simultaneously scaling the voltages of the processors and communication links is introduced in Section 3.9.

3.4 Motivational Examples

3.4.1 Optimizing the Dynamic and Leakage Energy

Fig. 3.2 shows two optimal voltage schedules for a set of three tasks (τ_1, τ_2, and τ_3), executing in two possible voltage modes. While the first schedule relies on V_{dd} scaling only (i.e., V_{bs} is kept constant), the second schedule corresponds to the

simultaneous scaling of V_{dd} and V_{bs}. Please note that the figures depict the dynamic and the leakage power dissipation as a function of time. For simplicity we neglect transition overheads in this example. Further, we consider processor parameters that correspond to CMOS technology ($< 65nm$) which leads to a leakage power consumption close to 40% of the total power consumed (at the mode with the highest performance).

Let us consider the first schedule in which the tasks are executed either at $V_{dd1} = 1.8V$, or $V_{dd2} = 1.5V$, while V_{bs1} and V_{bs2} are kept at $0V$. In accordance, the system dissipates $P_{dyn1} = 100mW$ and $P_{leak1} = 75mW$ in mode 1 running at $700MHz$, while $P_{dyn2} = 49mW$ and $P_{leak2} = 45mW$ in mode 2 running at $525MHz$, as observable from the figure. We have also indicated the individual energy consumed in each of the active modes, separating between dynamic and leakage energy. The total leakage and dynamic energies of the schedule in Fig. 3.2(a) are $13.56\mu J$ and $16.17\mu J$, respectively. This results in a total energy consumption of $29.73\mu J$.

Consider now the schedule given in Fig. 3.2(b), where tasks are executed at two different voltage settings for V_{dd} and V_{bs} ($m_1 = (1.8V, 0V)$ and $m_2 = (1.5V, -0.4V)$). Since the voltage settings for mode m_1 did not change, the system runs at $700MHz$ and dissipates $P_{dyn1} = 100mW$ and $P_{leak1} = 75mW$. In mode m_2 the system performs at $480Mhz$ and dissipates $P_{dyn2} = 49mW$ and $P_{leak2} = 5mW$. There are two main differences to observe compared to the schedule in Fig. 3.2(a). Firstly, the leakage power consumption during mode m_2 is considerably smaller than in Fig. 3.2(a); this is due to the fact that in mode m_2 the leakage is reduced through a body-bias voltage of $-0.4V$ (see Eq. (3.2)). Secondly, the high voltage mode m_1 is active for a longer time; this can be explained by the fact that scaling V_{bs} during mode m_2 requires the reduction of the operational frequency (see Eq. (3.3)). Hence, in order to meet the system deadline, the high performance mode m_1 has to compensate for this delay. Although here the dynamic energy was increased from $16.17\mu J$ to $18.0\mu J$, compared to the first schedule, the leakage was reduced from $13.56\mu J$ to $8.02\mu J$. The overall energy dissipation is $26.02\mu J$, a reduction by 12.5%. This example illustrates the advantage of simultaneous V_{dd} and V_{bs} scaling compared to V_{dd} scaling only.

3.4.2 Considering the Transition Overheads

We consider a single processor system that offers three voltage modes, $m_1 = (1.8V, -0.3V)$, $m_2 = (1.5V, -0.45V)$, and $m_3 = (1.2V, -0.8V)$, where $m_z = (V_{dd_z}, V_{bs_z})$. The rail and substrate capacitance are given as $C_r = 10\mu F$ and $C_s = 40\mu F$. The processor needs to execute two consecutive tasks (τ_1 and τ_2) with a deadline of $0.225ms$. Fig. 3.3(a) shows a possible voltage schedule. Each of

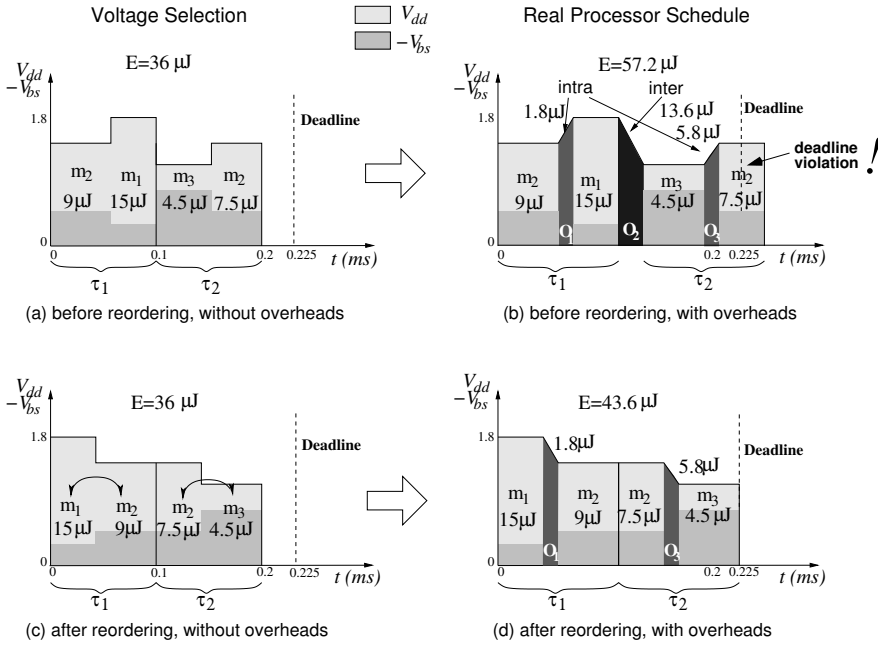


Figure 3.3: Influence of transition overheads

the two tasks is executed in two different modes: task τ_1 executes first in mode m_2 and then in mode m_1, while task τ_2 is initially executed in mode m_3 and then in mode m_2. The total energy consumption of this schedule is $E = 9\mu J + 15\mu J + 4.5\mu J + 7.5\mu J = 36\mu J$. However, if this voltage schedule is applied to a *real* voltage-scalable processor, the resulting schedule will be affected by transition overheads, as shown in Fig. 3.3(b). The processor requires a given time to adapt to the new execution mode. During this adaption no computations can be performed [xsc00, pow00], which increases the schedule length such that the imposed deadline is violated. Moreover, transitions do not only require time, they also cause an additional energy dissipation. For instance, in the given schedule, the first transition overhead O_1 from mode m_2 and m_1 requires an energy of $10\mu F \cdot (1.8V - 1.5V)^2 + 40\mu F \cdot (0.3V - 0.45V)^2 = 1.8\mu J$, based on Eq. (3.4). Similarly, the energy overheads for transitions O_2 and O_3 can be calculated as $13.6\mu J$ and $5.8\mu J$, respectively. The overall energy dissipation of the schedule from Fig. 3.3(b) accumulates to $36\mu J + 1.8\mu J + 13.6\mu J + 5.8\mu J = 57.2\mu J$.

Compared to the schedule in Fig. 3.3(a), the mode activation order in Fig. 3.3(c) has been swapped for both tasks. As long as the transition overheads are neglected, the energy consumption of the two schedules is identical. However, considering transition overheads would result in the schedule shown in Fig. 3.3(d). We can observe that this schedule exhibits only two mode transitions (O_1 and O_3) within the tasks (intra task switches), while the switch between the two tasks (inter task switch) has been eliminated. The overall energy consumption has been reduced to $E = 43.6\mu J$, a reduction by 23.8% compared to the schedule given in Fig. 3.3(b). Further, the elimination of transition O_2 reduces the overall schedule length, such that the imposed deadline is satisfied. With this example we have illustrated the effects that transition overheads can have on the energy consumption and the timing behavior and the impact of taking them into consideration when elaborating the voltage schedule.

3.5 Problem Formulation

Consider a set of tasks $\mathcal{T} = \{\tau_i\}$ with precedence constraints, that have been mapped and scheduled on a set of variable voltage processors. For each task τ_i its deadline dl_i, its worst case number of clock cycles to be executed WNC_i and the switched capacitance C_{eff_i} are given. Each processor can vary its supply voltage V_{dd} *and* body bias voltage V_{bs} within certain continuous ranges (for the continuous problem), or, within a set of discrete voltage pairs $m_z = \{(V_{dd_z}, V_{bs_z})\}$ (for the discrete problem). The power dissipations (leakage and dynamic) and the cycle time (processor speed) depend on the selected voltage pair (mode). Tasks are executed cycle by cycle, and each cycle can potentially execute at a different voltage pair, i.e., at a different speed. Our goal is to find voltage pair assignments for each task such that the individual task deadlines are met and the total energy consumption is minimal. Furthermore, whenever the processor has to alter the settings for V_{dd} and/or V_{bs}, a transition overhead in terms of energy and time is required (see Eqs. (3.4) and (3.5)).

For reasons of clarity we introduce the following four distinctive problems which will be considered in this chapter: (a) Continuous voltage selection with no consideration of transition overheads (CNOH), (b) continuous voltage selection with consideration of transition overheads (COH), (c) discrete voltage selection with no consideration of transition overheads (DNOH), and (d) discrete voltage scaling with consideration of transition overheads (DOH).

3.6 Continuous Voltage Selection

In this section we consider that the supply and body-bias voltage of the processors can be selected within a certain continuous range. We first formulate the problem neglecting transition overheads (Section 3.6.1, CNOH) and then extend this formulation to include the energy and delay overheads (Section 3.6.2, COH).

3.6.1 Continuous Voltage Selection without Overheads (CNOH)

We model the continuous voltage selection problem, excluding the consideration of transition overheads (the CNOH problem), using the following nonlinear problem formulation.
Minimize

$$\sum_{k=1}^{|\mathcal{T}|} \Big(\underbrace{WNC_k \cdot C_{eff_k} \cdot V_{dd_k}^2}_{E_{dyn_k}} + \underbrace{L_g (K_3 \cdot V_{dd_k} \cdot e^{K_4 \cdot V_{dd_k}} \cdot e^{K_5 \cdot V_{bs_k}} + I_{Ju} \cdot |V_{bs_k}|) \cdot t_k}_{E_{leak_k}} \Big) \tag{3.6}$$

subject to

$$t_k = WNC_k \cdot \frac{(K_6 \cdot L_d \cdot V_{dd_k})}{((1+K_1) \cdot V_{dd_k} + K_2 \cdot V_{bs_k} - V_{th_1})^{\alpha}} \tag{3.7}$$

$$D_k + t_k \leq D_l \quad \forall (k,l) \in \mathcal{E} \tag{3.8}$$

$$D_k + t_k \leq dl_k \quad \forall\, \tau_k \text{ that have a deadline} \tag{3.9}$$

$$D_k \geq 0 \tag{3.10}$$

$$V_{dd_{min}} \leq V_{dd_k} \leq V_{dd_{max}} \text{ and } V_{bs_{min}} \leq V_{bs_k} \leq V_{bs_{max}} \tag{3.11}$$

The variables that need to be determined are the task execution times t_k, the task start times D_k as well as the voltages V_{dd_k} and V_{bs_k}. The total energy consumption, which is the sum of dynamic and leakage energy, has to be minimized, as in Eq. (3.6)[1]. The task execution time has to be equivalent to the number of clock cycles of the task multiplied by the circuit delay for a particular V_{dd_k} and V_{bs_k} setting, as expressed by Eq. (3.7). Given the execution time of the tasks, it becomes possible to express the precedence constraints between tasks (Eq. (3.8)), i.e., a task τ_l can only start its execution after all its predecessor tasks τ_k have finished their execution ($D_k + t_k$). Predecessors of task τ_l are all tasks τ_k for which there exists

[1] Note that *abs* and *max* operations cannot be used directly in mathematical programming, yet there exist standard techniques to overcome this limitation by equivalent formulations [Wil99].

an edge $(k,l) \in \mathcal{E}$ in the mapped and scheduled task graph. Similarly, tasks with deadlines have to be completed ($D_k + t_k$) before their deadlines dl_k (Eq. (3.9)). Task start times have to be positive (Eq. (3.10)) and the imposed voltage ranges should be respected (Eq. (3.11)). It should be noted that the objective (Eq. (3.6)) as well as the task execution time (Eq. (3.7)) are convex functions. Hence, the problem falls into the class of general convex nonlinear optimization problems. Such problems can be efficiently solved in polynomial time (given an arbitrary precision $\varepsilon > 0$), [NN94].

3.6.2 Continuous Voltage Selection with Overheads (COH)

In this section we modify the previous formulation in order to take transition overheads into account (COH problem). The following formulation highlights the modifications.

Minimize

$$\underbrace{\sum_{k=1}^{|\mathcal{T}|} (E_{dyn_k} + E_{leak_k})}_{\text{Task energy dissipation}} + \underbrace{\sum_{(k,j)\in\mathcal{E}^\bullet} \varepsilon_{k,j}}_{\text{Transition energy overhead}} \tag{3.12}$$

subject to

$$D_k + t_k + \delta_{k,j} \leq D_j \quad \forall (k,j) \in \mathcal{E}^\bullet \tag{3.13}$$

$$\delta_{k,j} = \max(p_{Vdd} \cdot |V_{dd_k} - V_{dd_j}|, p_{Vbs} \cdot |V_{bs_k} - V_{bs_j}|) \tag{3.14}$$

$$\varepsilon_{k,j} = C_r \cdot |V_{dd_k} - V_{dd_j}|^2 + C_s \cdot |V_{bs_k} - V_{bs_j}|^2 \tag{3.15}$$

The objective function Eq. (3.12) now additionally accounts for the transition overheads in terms of energy. The energy overheads can be calculated according to Eq. (3.4) for all consecutive tasks τ_k and τ_j on the same processor ($\mathcal{E}^\bullet$ is defined in Section 3.2). However, scaling voltages does not only require energy but it introduces delay overheads as well. Therefore, we introduce an additional constraint similar to Eq. (3.8), which states that a task τ_j can only start after the execution of its predecessor τ_k ($D_k + t_k$) on the same processor and after the new voltage mode is reached ($\delta_{k,j}$). This constraint is given in Eq. (3.13). The delay $\delta_{k,j}$ and energy $\varepsilon_{k,j}$ penalties are introduced as a set of new variables and are constrained subject to Eq. (3.14) and Eq. (3.15). Similar to the CNOH formulation, the COH model is a convex nonlinear problem, i.e., it can be solved in polynomial time.

3.7 Discrete Voltage Selection

The approaches presented in the previous section provide a theoretical upper bound on the possible energy savings. In reality, however, processors are restricted to a discrete set of V_{dd} and V_{bs} voltage pairs. In this section we investigate the discrete voltage selection problem without and with the consideration of overheads. We will also analyze the complexity of the discrete voltage selection problem.

3.7.1 Problem Complexity

Theorem 1 *The discrete voltage selection problem is NP-hard.*

Proof 1 *We proof by restriction. The discrete time-cost trade-off (DTCT) problem is known to be NP-hard [DDGW97]. By restricting the discrete voltage selection problem (DNOH) to contain only tasks that require an execution of one clock cycle, it becomes identical to the DTCT problem. Hence, DTCT $\in$ DNOH which leads to the conclusion DNOH $\in$ NP.*

The details of the proof are given in Appendix A. The problem is NP-hard, even if restricted to supply voltage selection (without adaptive body-biasing) and even if transition overheads are neglected. It should be noted that this finding renders the conclusion of [KK05][2] impossible, which states that the discrete voltage selection problem (considered in [KK05] without body-biasing and overheads) can be solved optimally in polynomial time.

3.7.2 Discrete Voltage Selection without Overheads (DNOH)

In the following we will give a mixed integer linear programming (MILP) formulation for the discrete voltage selection problem without overheads (DNOH). We consider that processors can run in different modes $m \in \mathcal{M}$. Each mode m is characterized by a voltage pair (V_{dd_m}, V_{bs_m}), which determines the operational frequency f_m, the normalized dynamic power P_{dnom_m}, and the leakage power dissipation P_{leak_m}. The frequency and the leakage power are given by Eqs. (3.3) and (3.2), respectively. The normalized dynamic power is given by $P_{dnom_m} = f_m \cdot V_{dd_m}^2$. Accordingly, the dynamic power of a task τ_k operating in mode m is computed as $C_{eff_k} \cdot P_{dnom_m}$. Based on these definitions, the problem is formulated as follows:

[2] The flaw in [KK05] lies in the fact that the number of clock cycles spent in a mode is not restricted to be integer.

Minimize

$$\sum_{k=1}^{|\mathcal{T}|} \sum_{m \in \mathcal{M}} \left(C_{eff_k} \cdot P_{dnom_m} \cdot t_{k,m} + P_{leak_m} \cdot t_{k,m} \right) \tag{3.16}$$

subject to

$$D_k + \sum_{m \in \mathcal{M}} t_{k,m} \leq dl_k \ \forall \tau_k \text{ with deadline} \tag{3.17}$$

$$D_k + \sum_{m \in \mathcal{M}} t_{k,m} \leq D_l \quad \forall (k,l) \in \mathcal{E} \tag{3.18}$$

$$c_{k,m} = t_{k,m} \cdot f_m \quad \text{and} \quad \sum_{m \in \mathcal{M}} c_{k,m} = WNC_k \qquad c_{k,m} \in \mathbb{N} \tag{3.19}$$

$$D_k \geq 0 \quad \text{and} \quad t_{k,m} \geq 0 \tag{3.20}$$

The total energy consumption, expressed by Eq. (3.16), is given by two sums. The inner sum indicates the energy dissipated by an individual task τ_k, depending on the time $t_{k,m}$ spent in each mode m. The outer sum adds up the energy of all tasks. Unlike the continuous voltage selection case, we do not obtain the voltage V_{dd} and V_{bs} directly, but rather we find out how much time to spend in each of the modes. Therefore, task execution time $t_{k,m}$ and the number of clock cycles $c_{k,m}$ spent within a mode become the optimization variables in the MILP formulation. The number of clock cycles $c_{k,m}$ is restricted to the integer domain. We exemplify this model graphically in Figures 3.4(a) and 3.4(b). The first figure shows the schedule of two tasks executing each at two different voltage settings (two modes out of three possible). Task τ_1 executes for 20 clock cycles in mode m_2 and for 10 clock cycles in m_1, while task τ_2 runs for 5 clock cycles in m_3 and 15 clock cycles in m_2. The same is captured in Fig. 3.4(b) in what we call a mode model. The modes that are not active during a task's runtime have the corresponding time and number of clock cycles 0 (mode m_3 for τ_1 and $m1$ for τ_2). The overall execution time of task τ_k is given as the sum of the times spent in each mode ($\sum_{m \in \mathcal{M}} t_{k,m}$). Eq. (3.17) ensures that all the deadlines are met and Eq. (3.18) maintains the correct execution order given by the precedence relations. The relation between execution time and number of clock cycles as well as the requirement to execute all clock cycles of a task are expressed in Eq. (3.19). Additionally, task start times D_k and task execution times have to be positive (Eq. (3.20)).

3.7.3 Discrete Voltage Selection with Overheads (DOH)

We now proceed with the incorporation of transition overheads into the MILP formulation given in Section 3.7.2. The order in which the modes are activated has

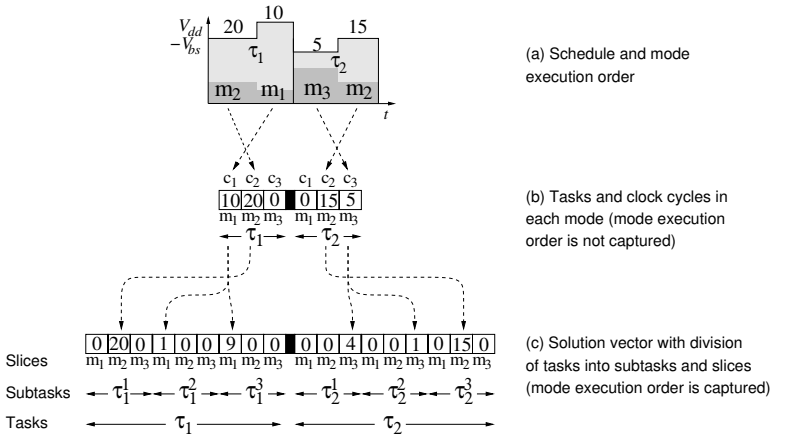


Figure 3.4: Discrete mode model

an influence on the transition overheads, as we have illustrated in Section 3.4.2. Nevertheless, the formulation in Section 3.7.2 does not capture the order in which modes are activated, it solely expresses how many clock cycles are spent in each mode. We introduce the following extensions needed in order to take both delay and energy overheads into account. Given m operational modes, the execution of a single task τ_k can be subdivided into m subtasks $\tau_k^s, s = 1,...,m$. Each subtask is executed in one and only one of the m modes. Subtasks are further subdivided into m slices, each corresponding to a mode. This results in $m \cdot m$ slices for each task. Fig. 3.4(c) depicts this model, showing that task τ_1 runs first in mode m_2, then in mode m_1, and that τ_2 runs first in mode m_3, then in m_2. This ordering is captured by the subtasks: the first subtask of τ_1 executes 20 clock cycles in mode m_2, the second subtask executes one clock cycle in m_1 and the remaining 9 cycles are executed by the last subtask in mode m_1; τ_2 executes in its first subtask 4 clock cycles in mode m_3, 1 clock cycle is executed during the second subtask in mode m_3, and the last subtask executes 15 clock cycles in the mode m_2. Note that there is no overhead between subsequent subtasks that run in the same mode. The following gives the modified MILP formulation:

Minimize

$$\underbrace{\sum_{k=1}^{|\mathcal{T}|}\sum_{s\in\mathcal{M}}\sum_{m\in\mathcal{M}}\left(C_{eff_k}\cdot P_{dnom_m}\cdot t_{k,s,m}+P_{leak_m}\cdot t_{k,s,m}\right)}_{\text{Task energy dissipation}}$$
$$+\underbrace{\sum_{k=1}^{|\mathcal{T}|}\sum_{s\in\mathcal{M}}\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{M}}\left(b_{k,s,i,j}\cdot EP_{i,j}\right)}_{\text{Transition energy overhead}} \tag{3.21}$$

subject to

$$\delta_k=\sum_{s\in\mathcal{M}^*}\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{M}}b_{k,s,i,j}\cdot DP_{i,j} \tag{3.22}$$

$$\delta_{k,l}=\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{M}}b_{k,m,i,j}\cdot DP_{i,j}\quad\text{where }(k,l)\in\mathcal{E}^\bullet \tag{3.23}$$

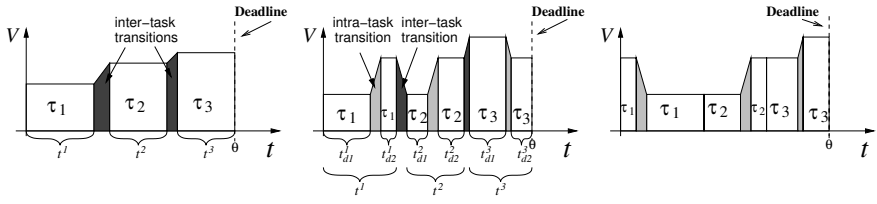
$$D_k+\sum_{s\in\mathcal{M}}\sum_{m\in\mathcal{M}}t_{k,s,m}+\delta_k\le dl_k\quad\forall\tau_k\text{ with deadline} \tag{3.24}$$

$$D_k+\sum_{s\in\mathcal{M}}\sum_{m\in\mathcal{M}}t_{k,s,m}+\delta_k+\delta_{pl,l}\le D_l\quad\forall(k,l)\in\mathcal{E},(pl,l)\in\mathcal{E}^\bullet \tag{3.25}$$

$$c_{k,s,i}=t_{k,s,i}\cdot f_i\quad s\in\mathcal{M},i\in\mathcal{M},c_{k,s,i}\in\mathbb{N} \tag{3.26}$$

$$\sum_{s\in\mathcal{M}}\sum_{i\in\mathcal{M}}c_{k,s,i}=WNC_k \tag{3.27}$$

In order to capture the energy overheads in the objective function (Eq. (3.21)), we introduce the boolean variables $b_{k,s,i,j}$. In addition, we introduce an energy penalty matrix EP, which contains the energy overheads for all possible mode transitions, i.e., $EP_{i,j}$ denotes the energy overhead necessary to change form mode i to j. These overheads are precomputed based on the available modes (voltage pairs) and Eq. (3.4). The overall energy overhead is given by all intratask and intertask transitions. The intratask and intertask delay overheads, given in Eq. (3.22) and (3.23), are calculated based on a delay penalty matrix $DP_{i,j}$, which, similarly to the energy penalty matrix, can be precomputed based on the available modes and Eq. (3.5). For a task τ_k and for each of its subtasks τ_k^s, except the last one, the variable $b_{k,s,i,j}=1$ if mode i of subtask τ_k^s and mode j of τ_k^{s+1} are both active (s in $1,...,|\mathcal{M}|-1$, i,j in $1,...,m$). These are used in order to capture the intratask overheads, as in Eq. (3.22). For intertask overheads, we are interested in the last mode of task τ_k and the first mode of the subsequent task τ_l (running on the same processor). Therefore, $b_{k,m,i,j}=1$ if the mode i of the last subtask τ_k^m and the



(a) Continuous Voltage Schedule (solely inter−task mode transitions)

(b) Discrete Voltage Schedule (5 performance mode transitions)

(c) Reordered Discrete Schedule (3 performance mode transitions)

Figure 3.5: VS heuristic: mode reordering

mode j of first subtask τ_i^1 are both active. For the example given in Fig. 3.4(c), $b_{1,1,2,1}$, $b_{1,2,1,1}$, $b_{1,3,1,3}$, $b_{2,1,3,3}$, $b_{2,2,3,2}$ are all 1 and the rest are 0. Deadlines and precedence relations, taking the delay overheads into account, have to be respected according to Eq. (3.24) and (3.25). Here $\sum_{s\in\mathcal{M}}\sum_{m\in\mathcal{M}} t_{k,s,m}$ represents the total execution time of a task τ_k, based on the number of cycles in each of the subtasks and modes. Eq. (3.26) and (3.27) are a reformulation of Eq. (3.19), which expresses the relation between the execution time and the number of clock cycles and the requirement to execute all clock cycles of a task. To ease the explanation, the above given MILP formulation has been simplified to a certain degree. We have omitted here details on the computation of the b variables as well as the constraints that make sure that one and only one mode is used by a subtask. The complete MILP model can be found in Appendix B.

3.7.4 Discrete Voltage Selection Heuristic

As shown earlier, discrete voltage selection is NP-hard. Thus, solving it using the presented MILP formulation for large instances is time consuming. We propose a heuristic to effectively solve the discrete voltage selection problem. The main idea behind this heuristic is to perform a continuous voltage selection (as outlined in Section 3.6). As a result of this calculation, for each task, a continuous voltage pair $(V_{dd_{con}}, V_{bs_{con}})$, as well as the corresponding frequency f_{con} will be determined. Using the approach introduced in [IY98], for each task the two surrounding discrete performance modes are chosen such that $f_{d1} < f_{con} < f_{d2}$. That is, the execution of a task is split into two regions with t_{d1} and t_{d2} being the execution times in the mode with f_{d1} and f_{d2}, respectively. Fig. 3.5(a) and 3.5(b) illustrate this transformation for an application with three tasks. In the continuous scaling case, Fig. 3.5(a), each task executes at a single voltage level, i.e., the voltages are changed only between tasks. In the discrete case, the voltage setting is changed during the task execution.

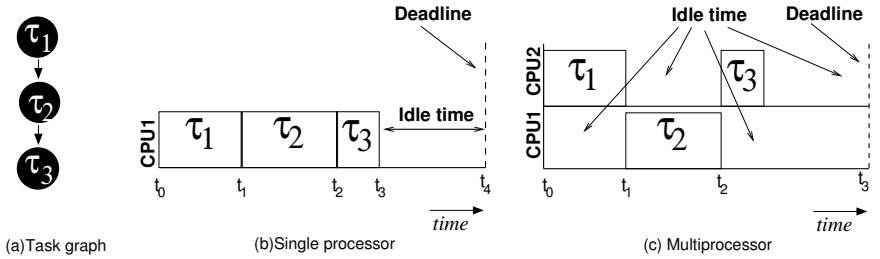


Figure 3.6: Schedules with idle times

Of course, the required time overhead δ_i for the mode change has to be considered as well, i.e., $t^i = t^i_{d1} + t^i_{d2} + \delta_i$, where t^i is the execution time with continuous voltage setting of the task τ_i. In general, executing tasks in two performance modes, determined as above, leads to close to optimal discrete voltage selection. Having determined the discrete performance mode settings, the inter-task transition overheads are reduced by reordering the mode sequence of each task. We reorder the modes in a greedy manner, such that the inter-task overhead between consecutive tasks is minimized. This is outlined in Fig. 3.5(c). While this reordering technique is optimal for processors that offer two performance modes, this is not true for components with three or more modes. Nevertheless, as demonstrated by our experiments, this heuristic is fast and efficient.

3.8 Voltage Selection with Processor Shutdown

In this section we discuss the integration of two system level energy minimization techniques: voltage selection and processor shutdown. Voltage selection is effective in minimizing the active energy consumption (the energy consumed while executing a certain task). However, specially in multiprocessor environments, processors alternate between active and idle periods. During idle times, a certain amount of energy, proportional to the length of the idle period is consumed. A solution for saving this energy is to shutdown the processor. The transition to the shutdown state and from shutdown back to operation implies a time and an energy overhead.

Idle times may be present due to multiple reasons, even after performing voltage selection. Consider, for example, the three tasks in Fig. 3.6(a). They are considered to run periodically with a deadline equal to the period. If the application runs on a single processor system at the lowest speed, it still finishes before the deadline, as depicted in Fig. 3.6(b). In the idle interval between the finishing time

and the deadline, the processor consumes energy. In this situation, we could shut down the processor and thus save energy. In the case of a single processor system with tasks that do not have arbitrary arrival times, deciding weather or not to shutdown and for how long is relatively easy. In [RJ05], the notion of threshold time interval is defined as the minimul length of an idle period that would provide energy savings by shuting down. A shutdown is decided if the idle interval available is larger than the threshold time.

Imagine now a more complex case, when the application runs on two processors, as in Fig. 3.6(c). Due to dependencies between tasks that are mapped on different processors, there is a certain amount of slack that cannot be exploited by voltage selection. For example, task τ_2 can start only after task τ_1 has finished. Consequently, there is an idle interval on CPU_1 from time 0, until the start of τ_2. Deciding in this case weather or not to shutdown is a complex problem that will be addressed in the following section.

Even though voltage selection aims at optimizing the active energy, while processor shutdown minimizes the energy consumed during idle periods, these two techniques are not orthgonal. Let us consider an application consisting of 3 tasks, τ_1, τ_2 and τ_3, as in Fig. 3.7(a). The tasks are mapped on two processors CPU_1 and CPU_2. The resulting schedule, after performing voltage selection is depicted in Fig. 3.7(b), with all the 3 tasks running at the lowest speeds. Task τ_1 is running for $2ms$ with $200mW$, while τ_2 and τ_3 run at $400mW$ for $1.5ms$ and respectively $2ms$. A brief analysis of the idle times present after voltage selection on both processors, allows us to further reduce the energy consumption by shutting down CPU_1 after the execution of τ_1 and of CPU_2 after τ_3. The energy overhead for shutdown is $75\mu J$ on CPU_1 and $125\mu J$ on CPU_2. We notice the idle interval of $0.5ms$ on CPU_2, between the executions of τ_2 and τ_3. The idle power on CPU_2 is $250mW$, resulting in an energy consumption of $125\mu J$. Please note that the energy consumed during this idle period equals the energy overhead of a shutdown, so it would not pay off to shutdown after τ_2. However, let us consider the possibility of running τ_1 faster, such that it finishes in $1.5ms$. The power consumption that corresponds to this frequency is $300mW$. This slight increase on CPU_1 is compensated by the fact that we can now execute task τ_3 immediately after τ_2, use one shutdown operation to exploit all the idle time on CPU_2 and thus save $125\mu J$.

3.8.1 Processor Shutdown: Problem Complexity

The shutdown problem without voltage selection (SNVS) is formulated as follows:

Consider a set of tasks with precedence constraints $\mathcal{T} = \{\tau_i\}$ that have been mapped and scheduled on a set of processors. Each processor operates at a given fixed frequency. For each task τ_i, its deadline dl_i and number of clock cycles to

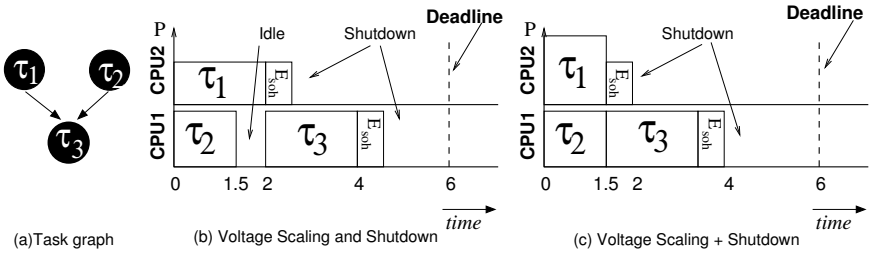


Figure 3.7: Voltage Selection with Shutdown

be executed WNC_i are given. The start time of each task is variable (with the constraints imposed by the precedences in the scheduled task graph). When a processor is idle, an amount of energy proportional to the length of the idle interval is consumed. In order to save energy, during such an idle interval the particular processor can be shut down. A shutdown operation comes with a fixed time and energy penalty. Our goal is to minimize the energy consumed by the system while the processors are idle. This translates into spending as much as possible of the idle time in the shutdown state. In order to be energy efficient, the best solution will assign the task start times such that idle times are grouped together in long intervals that can be covered with few shutdown operations.

Theorem 2 *The shutdown problem (SNVS) is NP-complete.*

The proof is given in Appendix C. It is based on the fact that the multiple choice continuous knapsack problem can be reduced to the SNVS problem. If the simple shutdown problem without performing voltage selection is NP complete, then the combined voltage selection problem with shutdown (even in the case with continuous voltages) is NP complete as well.

3.8.2 Continuous Voltage Selection with Processor Shutdown (CVSSH)

In this section we present an exact integer nonlinear formulation as well as a polynomial time heuristic for the voltage selection with processor shutdown [3]. The following gives the modified nonlinear programming formulation (CVSSH):

[3]For simplicity of the presentation, we omit here the consideration of voltage transition overheads. Nevertheless, these overheads can be easily included, as shown in Section 3.6.2

Minimize

$$\underbrace{\sum_{k=1}^{|\mathcal{T}|} WNC_k \cdot C_{eff_k} \cdot V_{dd_k}^2}_{E_{dyn}}$$
$$+ \underbrace{\sum_{k=1}^{|\mathcal{T}|} L_g \cdot (K_3 \cdot V_{dd_k} \cdot e^{K_4 \cdot V_{dd_k}} \cdot e^{K_5 \cdot V_{bs_k}} + I_{Ju} \cdot |V_{bs_k}|) \cdot t_k}_{E_{leak}}$$
$$+ \underbrace{\sum_{k=1}^{|\mathcal{T}|} xi_k \cdot t_{idle_k} \cdot P_{idle_k} + xs_k \cdot (E_{soh_k} + t_{off_k} \cdot P_{off_k})}_{E_{idle}+E_{off}} \tag{3.28}$$

subject to

$$t_k = WNC_k \cdot \frac{(K_6 \cdot L_d \cdot V_{dd_k})}{((1+K_1) \cdot V_{dd_k} + K_2 \cdot V_{bs_k} - V_{th_1})^{\alpha}} \tag{3.29}$$

$$\begin{aligned}
D_k + t_k &\leq D_l \quad \forall (k,l) \in \mathcal{E} - \mathcal{E}^{\bullet} && (3.30)\\
D_k + t_k + xi_k \cdot t_{idle_k} &= D_l \quad \forall (k,l) \in \mathcal{E}^{\bullet} && (3.31)\\
D_k + t_k + xs_k \cdot T_{soh_k} + t_{off_k} &= D_l \quad \forall (k,l) \in \mathcal{E}^{\bullet} && (3.32)\\
xi_k + xs_k &= 1 \quad \forall \tau_k && (3.33)\\
D_k + t_k &\leq dl_k \quad \forall\, \tau_k \text{ with dl} && (3.34)\\
D_k &\geq 0 && (3.35)\\
xi_k, xs_k &\in \{0,1\} && (3.36)
\end{aligned}$$

$$V_{dd_{min}} \leq V_{dd_k} \leq V_{dd_{max}} \text{ and } V_{bs_{min}} \leq V_{bs_k} \leq V_{bs_{max}} \tag{3.37}$$

There are two noticeable differences between this formulation and the one in Section 3.6.1: the inclusion in the objective (Eq. 3.28) of the energy spent during idle and shutdown intervals and Eq. 3.32 and 3.31 introduced in order to account for the idle and off times. P_{idle_k}, P_{off_k}, E_{soh_k} and T_{soh_k} are constants for each task τ_k and capture the power consumed by the processor on which τ_k is mapped, during idle and shutdown time intervals and respectively the energy and the time overhead associated to a shutdown operation. Please note the usage in Eq. 3.28, 3.31 and 3.32 of binary variables xi_k and xs_k, associated to each task, with the following semantics: if task τ_k is followed by a shutdown, then $xs_k = 1$ and $xi_k = 0$, otherwise $xi_k = 1$ and $xs_k = 0$. In case of a shutdown, t_{off_k} captures the amount of time the

processor is off. If there is no shutdown after the execution of τ_k, t_{idle_k} captures the amount of idle time (t_{idle_k} is 0 if the next task starts immediately after τ_k).

The binary variables xi_k and xs_k change the complexity of this nonlinear programming formulation, compared to the ones presented in Sections 3.6.1 and 3.6.2. While the problems presented there are convex nonlinear, the CVSSH problem is integer nonlinear. Indeed, as shown in the previous section, the voltage selection with shutdown problem is NP complete, even in the case when continuous voltage selection is used. Therefore, in the following, we propose a heuristic to efficiently solve the problem.

Let us consider particular instances of the CVSSH problem, where xi_k and xs_k are given constants for each task τ_k. We denote this simplified problem CVSI. Such a particular instance can be solved in polynomial time and computes the optimal voltages for a system in which we know the position of the shutdown operations. For example, if $xi_k = 1$, for all the tasks τ_k, CVSI computes the task voltages such that the energy is minimized, taking into account the idle energy, without performing any shutdown. Running CVSI for all possible combinations for xs_k and xi_k and selecting the one with the minimum energy, provides the optimal solution for the voltage selection with shutdown problem. This is, practically, not possible, of course. We will present in the following a heuristic that solves the CVSSH problem in polynomial time. The pseudocode of the heuristic is given in Fig. 3.8. The algorithm takes as input the mapped and scheduled task graph with each task characterized as in Section 3.5. It returns, the supply and body bias voltage for each task as well as the position and length of each shutdown operation and idle time.

As a first step (line 02), we perform voltage selection, using the CVSI nonlinear formulation. This will optimize the active and idle energy, without performing any shutdown operation ($xs_k = 0$ and $xi_k = 1$).

In a second step, (lines 03-11), the idle intervals are inspected one by one, and, if an interval is large enough (line 08) a shutdown is introduced. In more detail, we find iteratively the idle time with the highest energy that is large enough to allow a shutdown. For this purpose, we compute, for each task τ_k, the earliest finishing time EFT_k and the latest start time LST_k (line 04-05), assuming that each task is running at a fixed speed using the voltages computed by CVSI at line 02 or in the previous iteration at line 10. We select for shutdown the idle time that consumes the most energy (line 08-09). We set the corresponding binary variables $xs_k = 1$ and $xi_k = 0$ in order to schedule a shutdown after the task τ_k. Then we run CVSI with the updated values for xi and xs (line 10). At each new iteration the global energy consumption is improved.

```
Algorithm:  CONT_VS_SHUT_HEU
Input: - Mapped and scheduled task graph
       - For each task:  WNC_k, C_eff_k, dl_k
Output:- V_dd_k, V_bs_k, xs_k, xi_k, toff_k, tidle_k
01:  for all τ_k  xs_k = 0, xi_k = 1
02:  E_current=call CVSI
03:  while(1) {
04:   for all τ_k  EFT_k=earliest_start_time(τ_k)
05:   for all τ_k  LST_k=latest_start_time(τ_k)
06:   for all (k,l) ∈ E•  t_idle_k = LST_l − EFT_k
07:   if ∀τ_k  t_idle_k · P_idle_k ≤ E_soh_k  break
08:   *select τ_k with t_idle_k · P_idle_k = max{t_idle_l · P_idle_l | τ_l ∈ T}
09:   set xs_k = 1, xi_k = 0
10:   E_current=call CVSI
11:  }
12:  while(1) {
13:   for all τ_k  EFT_k=earliest_start_time(τ_k)
14:   for all τ_k  LST_k=latest_start_time(τ_k)
15:   for all (k,l),(l,m) ∈ E•  t_idle_k,l,m = LST_m − t_l − EFT_k
16:   if ∀(k,l),(l,m) ∈ E•,  t_idle_k,l,m · P_idle ≤ E_soh_k  break
17:   *select set σ_k,l,m with
        t_idle_k,l,m · P_idle_k = max{t_idle_h,i,j · P_idle_h | (h,i),(i,j) ∈ E•}
18:   set xs_k = 1, xi_k = 0, xs_l = 0, xi_l = 1
19:   E_1=call CVSI
20:   set xs_k = 0, xi_k = 1, xs_l = 1, xi_l = 0
21:   E_2=call CVSI
22:   *set (xs_k = 1, xs_l = 0) if E_1 > E_current & E_1 > E_2
23:   *set (xs_k = 0, xs_l = 1) if E_2 > E_current & E_2 > E_1
24:   *set (xs_k = 0, xs_l = 0) if E_1 < E_current & E_2 < E_current
25:   E_current = min{E_current, E_1, E_2}
26:  }
27:  return (V_dd_k, V_bs_k, xs_k, xi_k, t_off_k, tidle_k)
```

Figure 3.8: Voltage Selection with Shutdown Heuristic

When the algorithm exits the loop from lines 03-11, there is no idle interval that is large enough to produce energy savings by a shutdown (line 07). However, in principle, there are two ways to further reduce the consumed energy:

1) Increase the voltages of some tasks such that the idle intervals following them become longer and, thus, can be exploited by shutdowns.

2) Increase the voltages of some tasks such that several idle intervals can be merged and exploited by a single shutdown.

The first alternative can be excluded based on a simple reasoning. Let us assume that we have a task τ_k that runs in mode m_1 and consumes a certain amount energy E_k^1. Task τ_k is followed by an idle interval of length $t_{idle_k}^1$, that is too small to provide savings via shutdown:$t_{idle_k}^1 \cdot P_{idle_k} < E_{soh_k}$. The total energy consumed in this case is $E_k^1 + t_{idle_k}^1 \cdot P_{idle_k}$. Consider that we increase the speed of τ_k by running it with execution mode m_2 instead of m_1. In this case τ_k will consume E_k^2 ($E_k^2 > E_k^1$) and the idle interval becomes long enough to make a shutdown operation efficient. As a result the total energy is $E_k^2 + E_{soh_k}$. Since $E_k^2 > E_k^1$ and $E_{soh_k} > t_{idle_k}^1 \cdot P_{idle_k}$, the energy of the system obtained by running τ_k in execution mode m_2 with a shutdown during the idle time is actually higher than the energy of the system obtained by running τ_k in execution mode m_1 without a shutdown. As a conclusion, increasing the speed of a task such that an idle interval becomes large enough for a shutdown does not provide any energy savings.

The second alternative is illustrated in Fig. 3.7. The energy is reduced by speeding up certain tasks in order to create the possibility of merging several small idle intervals. In this way, the resulting idle interval can be exploited by a single shutdown operation. This alternative is explored as the third step of our heuristic (lines 12-26). We inspect all the groups of three consecutive tasks mapped on the same processor, τ_k, τ_l and τ_m with $(k,l),(l,m) \in \mathcal{E}^\bullet$ and explore the savings achievable by merging t_{idle_k} and t_{idle_l}. More exactly, for all sets of three tasks $\sigma_{k,l,m} = \{(\tau_k,\tau_l,\tau_m) | (k,l),(l,m) \in \mathcal{E}^\bullet\}$, we compute the maximum set idle time $t_{idle_{k,l,m}}$ as the difference between the latest start time of task τ_m, the execution time of τ_l and the earliest finishing time of τ_k (line 15). We select the set $\sigma_{k,l,m}$ with the highest energy (line 17). For this set, there are two candidate locations of the shutdown operation: after the execution of τ_k or after the execution of τ_l. Our algorithm explores both possibilities (lines 18-21). Using CVSI, we first compute the energy considering the showdown after τ_k (E_1) and secondly after τ_l (E_2). If both E_1 and E_2 are higher then the energy obtained without a shutdown after τ_k and τ_l, no shutdown is scheduled during this iteration (line 24). Otherwise, the algorithm schedules a shutdown after τ_k or after τ_l (lines 22-23). The global energy is improved at each iteration (line 25). The loop exits when no idle time corresponding to a set is large enough to produce savings via shutdown (line 16).

This heuristic relies on a continuous formulation for the computation of the task voltages. We use the heuristic presented in Section 3.7.4 in order to translate the computed voltage levels into the discrete ones available on the processors. While this is the practical way to solve the problem, for completion, we present in the next section a MILP formulation for the discrete voltage selection with processor shutdown.

3.8.3 Discrete Voltage Selection with Processor Shutdown

In the following we will give a mixed integer linear programming (MILP) formulation for the combined processor shutdown problem with discrete voltage selection. For the clarity of the explanation, we will not include in the mathematical programming model the overheads corresponding to transitions between different execution modes.

As in the Section 3.7.2, we consider that processors can run in different modes $m \in \mathcal{M}$. Each mode m is characterized by a voltage pair (V_{dd_m}, V_{bs_m}), which determines the operational frequency f_m, the normalized dynamic power P_{dnom_m}, and the leakage power dissipation P_{leak_m}. A processor shutdown operation has both a time and an energy penalty t_{soh} and, respectively, E_{soh}. During the shutdown state, the power consumption is P_{off}. Alternatively, the power consumed while the processor is idle is P_{idle}. t_{off_k} denotes the time the processor is shut down after the execution of task τ_k. t_{idle_k} denotes the time the processor is idle after the execution of task τ_k. The binary variable xs_k corresponding to task τ_k is 1 if a shutdown occurs after task τ_k and 0 otherwise.

Based to these definitions, the MILP problem is formulated as:
Minimize

$$\sum_{k=1}^{|\mathcal{T}|} \sum_{m \in \mathcal{M}} \left(C_{eff_k} \cdot P_{dnom_m} \cdot t_{k,m} + P_{leak_m} \cdot t_{k,m} \right) + \sum_{k=1}^{|\mathcal{T}|} xs_k \cdot (E_{soh} + t_{off_k} \cdot P_{off}) + (1 - xs_k) \cdot t_{idle_k} \cdot P_{idle} \tag{3.38}$$

subject to

$$D_k + \sum_{m \in \mathcal{M}} t_{k,m} \leq dl_k \tag{3.39}$$

$$D_k + \sum_{m \in \mathcal{M}} t_{k,m} \leq D_l \quad \forall (k,l) \in \mathcal{E} \tag{3.40}$$

$$D_k + \sum_{m \in \mathcal{M}} t_{k,m} + xs_k \cdot t_{soh} + t_{off_k} \leq D_l \text{ and} \tag{3.41}$$

$$D_l - D_k - \sum_{m \in \mathcal{M}} t_{k,m} = t_{idle_k} \; \forall (k,l) \in \mathcal{E}^{\bullet} \tag{3.42}$$

$$c_{k,m} = t_{k,m} \cdot f_m \quad \text{and} \quad \sum_{m \in \mathcal{M}} c_{k,m} = WNC_k \qquad c_{k,m} \in \mathbb{N} \tag{3.43}$$

$$D_k \geq 0 \quad \text{and} \quad t_{k,m} \geq 0 \tag{3.44}$$

The total energy consumption, expressed by Eq. (3.38), is given by two sums. The first sum adds up the energy dissipated by an individual task τ_k, running in different modes. The second sum adds up the energy of all the idle and shutdown intervals[4]. If a shutdown operation is decided after the end of task τ_k, then $xs[k] = 1$ and t_{off_k} will equal the amount of time the processor is off. If there is no shutdown, t_{idle_i} captures the corresponding idle time. The objective will select the energy corresponding to one of these two alternatives.

3.9 Combined Voltage Selection for Processors and Communication Links

In this section, we consider the supply and body bias voltage selection problem for processors and communication links. We introduce a set of communication models for energy and delay estimation. We study two different bus implementations and show the implication of the bus implementation type on the voltage selection strategy. We introduce a nonlinear model of the continuous voltage selection problem, which is optimally solvable in polynomial time, while for the discrete voltage selection case we use a heuristic similar to the one presented in Section 3.7.4. For simplicity of the explanation, we have not considered the processor shutdown during the formulation of the optimization problems in this section, however, the extension is straightforward.

[4]There are standard ways of rewriting the objective in order to use only linear expressions (eliminate the multiplication between variables xs_k, t_{off_k} and t_{idle_k}). We reffer the interested reader to [Wil99].

3.9.1 Voltage Selection on Repeater-Based Buses

Consider an architecture consisting of two voltage-scalable processing elements (CPU1 and CPU2) that communicate via a repeater-based, shared bus (CL1), which also allows voltage selection. CPU1 executes task τ_1 and CPU2 runs τ_2. Task τ_2 can only start after receiving data from τ_1, and it has to finish execution before a deadline of $2ms$. Fig. 3.9(a) shows the schedule for this system, considering an execution at the nominal voltage settings (highest supply voltage and body bias voltage). The diagram shows the energy dissipation (dynamic and leakage) of the

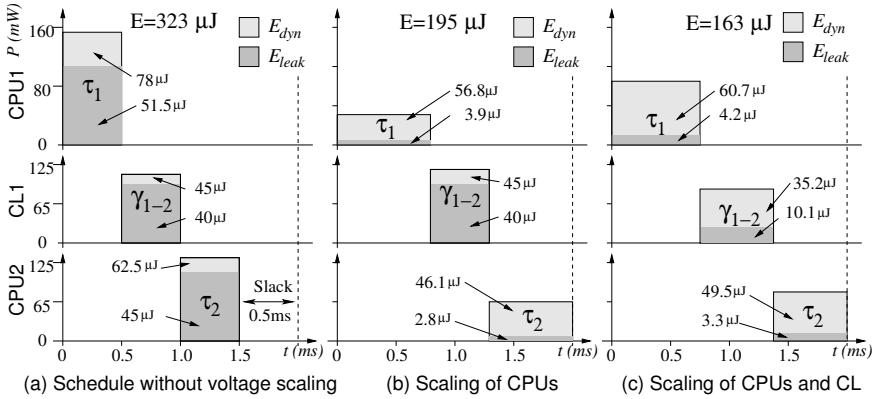


Figure 3.9: Voltage selection on a repeater-based bus

individual components. For clarity we assume in this example that the processors as well as the repeaters of the bus have the same nominal voltage values ($V_{dd} = 1.8V$ and $V_{bs} = 0V$). Furthermore, we assume that the supply voltages and the body bias voltages of all components can be varied continuously in the ranges $[0.6, 1.8]V$ and $[-1, 0]V$, respectively. Given the power consumptions at the nominal voltages, we can compute a total energy consumption of the tasks and communication in the initial schedule as $(156 + 103)mW \cdot 0.5ms + (90 + 80)mW \cdot 0.5ms + (125 + 90)mW \cdot 0.5ms = 323\mu J$. As can be observed, at the nominal voltages the system over-performs, leading to a slack of $0.5ms$.

We can exploit this slack by scaling the voltages of the processing elements. Using the technique described in Section 3.6, the resulting voltages for tasks τ_1 and τ_2 are $(1.43V, -0.42V)$ and $(1.54V, -0.49V)$, respectively. The corresponding, voltage scaled schedule is shown in Fig. 3.9(b). The dynamic and leakage power consumptions of the tasks are reduced to $(72mW, 5mW)$ and $(65mW, 4mW)$; however, the execution times have increased to $0.79ms$ and $0.71ms$ respectively.

With these settings, the system dissipates $195\mu J$, a reduction by 39% compared to the energy at nominal voltages.

To demonstrate the importance of combined voltage selection of the processors and the repeater-based bus, we have produced the schedule in Fig. 3.9(c). The optimal voltage settings can be calculated as $(1.48V, -0.42V)$ for CPU1, $(1.77V, -0.61V)$ for CPU2, and $(1.59V, -0.50V)$ for the bus repeaters. Correspondingly the power dissipations are $(81mW, 5.6mW)$, $(73.8mW, 4.9mW)$ and $(55.8mW, 16mW)$ thereby, reducing the overall system energy dissipation to $163\mu J$. This is a reduction of 49% compared to the nominal energy consumption, which is 10% more than in the case when only the PEs are voltage scaled.

3.9.2 Voltage Swing Selection on Fat Wire Buses

In this example, we illustrate the influence that a dynamic variation of the voltage swing (the voltage on the wire) has on the energy efficiency of the bus. Fig. 3.10 shows the total power consumption of a fat wire bus (including drivers and receivers), depending on the voltage swing at which data is sent. These plots have been generated via SPICE simulations using the Berkeley predictive $70nm$ CMOS technology library. The two plots show the total power consumption on the bus for two different voltage settings of the bus drivers and receivers. For example, if the driver connected to CPU1 and the receiver at CPU2 operate at $1.0V$, the lowest bus power dissipation ($0.55mW$) is achieved by a voltage swing of $0.14V$. Let us assume that the voltages of the driver and receiver are changed during run-time to $1.8V$ due to voltage selection. The bus power/voltage swing relation for this situation is indicated by the dashed line. As we can observe, by keeping the voltage swing at $0.14V$, the power dissipation on the bus will be $4.5mW$. However, inspecting the plot reveals that it is possible to reduce the bus power dissipation by changing the voltage swing from $0.14V$ to $0.6V$. At this voltage swing, the bus dissipates a power of $2.2mW$, i.e., a 51% reduction can be achieved by changing the voltage swing.

Now assume that the driver and receiver voltages are changed back from $1.8V$ to $1.0V$. Keeping the swing at $0.6V$ results in a power of $0.83mW$, which is, compared to the optimal $0.55mW$ at $0.14V$, 33% higher than necessary.

3.9.3 Communication Models

We consider a bus-based communication system as in Fig. 3.11. Whenever the processor CPU_1 sends data to CPU_2 over the bus, V_{dd_1} is converted to the bus voltage V_{dd_3} by the bus adapter of CPU_1. At the destination processor CPU_2, V_{dd_3}

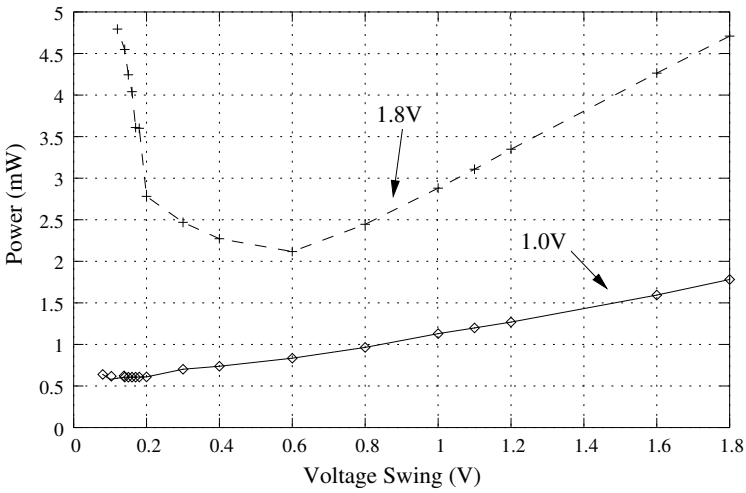


Figure 3.10: Optimum swing on a fat wire bus

is converted to V_{dd_2}. Each voltage conversion in the bus adapter requires an energy overhead, which is:

$$E_{adapter} = C_{adapter} \cdot (V_{dd_{CPU}} - V_{dd_{bus}})^2 \tag{3.45}$$

Thus, the total energy consumed when communicating between two processors CPU_1 and CPU_2 over the bus is:

$$E_{comm} = E_{adapter_1} + E_{bus} + E_{adapter_2} \tag{3.46}$$

Feature size scaling in deep-submicron circuits is responsible for an increasing wire delay of the global interconnects. This is mainly due to higher wire resistances caused by a shrinking cross-sectional area. Two approaches to cope with this problem have been proposed: (a) the usage of repeaters [IF99, KCS02] and (b) the usage of fat wires [Sve01, SK01]. The bus energy E_{bus} in Eq. (3.46) depends on which of these two approaches is used.

Repeater-Based Bus

The wire delay depends quadratically on the wire length, which can be approximated using an *RC* model. In order to reduce this quadratic dependency, it is possible to break the wire into smaller segments by inserting repeaters. The au-

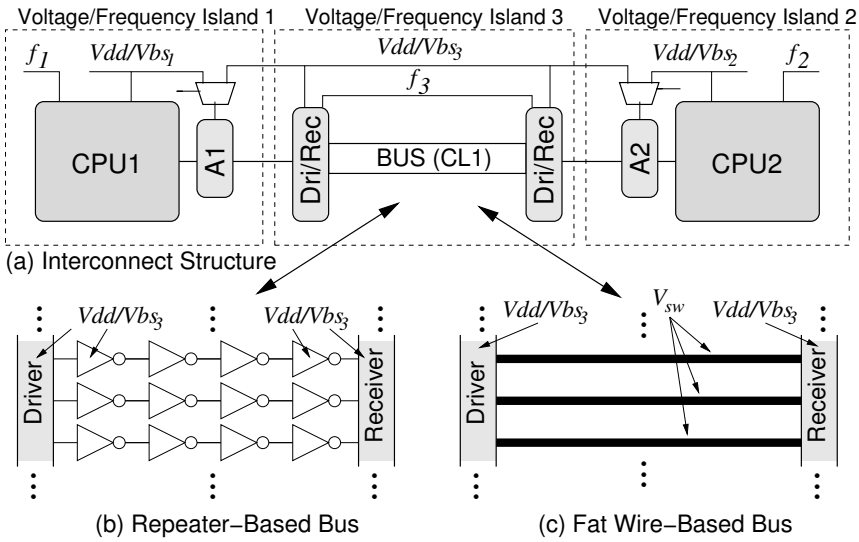


Figure 3.11: Interconnect structures

thors in [SK01] estimate an increasing number of repeaters with technology scaling down. For instance, up to 138 repeaters are used in $50nm$ technology for a corner-to-corner wire with a die size of $750mm^2$. Repeaters are implemented as simple CMOS inverter circuits (Fig. 3.11(b)). In accordance, the power dissipated by a bus implemented with repeaters is given by,

$$P_{rep} = N \cdot (\underbrace{s_\tau \cdot C_{rep} \cdot V_{dd}^2 \cdot f}_{P_{dyn}} + \underbrace{V_{dd} \cdot K_3 \cdot e^{K_4 \cdot V_{dd}} \cdot e^{K_5 \cdot V_{bs}} + |V_{bs}| \cdot I_{Ju}}_{P_{leak}}) \tag{3.47}$$

where N is the number of repeaters, s_τ is the average switching activity caused by communication task $\tau \in \mathcal{K}$, C_{rep} is the load capacity of a repeater (the sum of the output capacity of a repeater C_d, the wire capacity C_w, and the input capacity of the next repeater C_g), and V_{dd}, V_{bs}, and f are the supply voltage, body bias voltage, and the frequency at which the repeaters operate. Further, the constants K_3, K_4, K_5, and I_{Ju} depend on the repeater circuits (see Section 3.3).

The bus speed is constrained by the repeater frequency. Since repeaters are implemented as CMOS inverters, we use Eq. (3.3) to approximate the operational

frequency f of the bus. The execution time of a communication $\tau \in \mathcal{K}$ is given by,

$$t = \left\lceil \frac{NB_\tau}{W_{bus}} \right\rceil \cdot \frac{1}{f} \tag{3.48}$$

where NB_τ denotes the number of bits to be transmitted by communication τ and W_{bus} is the width of the bus (i.e. the number of bits transmitted with each clock cycle). Accordingly to Eq. (3.47) and (3.48), the bus energy dissipation is given by $E_{bus} = P_{rep} \cdot t$. Scaling the supply and body bias voltage of the repeaters requires also an overhead in terms of energy and time, similar to the overheads required by processor voltage selection (see Eq. (3.4) and (3.5)).

Fat Wire-Based Bus

Another approach for reducing the wire delay is to increase the physical dimensions of the wire, instead of scaling them down with technology. The usage of "fat" wires, on the top metal layer, has been proposed in [Sve01]. The main advantage of such wires is their low resistance. Provided that $L \cdot R_w/Z_0 < 2ln2$ (L is the wire length, R_w is the wire resistance per unit length and Z_0 its characteristic impedance), they exhibit a transmission line behavior, as opposed to the RC behavior in the repeater-based architecture. Using fat wires, the transmission speed approaches the physical limits (the speed of light in the particular dielectric). However, only a limited wire length can be accomplished with the available width of the top metal layer. For example, for a $4mm$ long wire in $180nm$ technology, the authors in [CS02] obtained a fat wire width of $2\mu m$ on the top metal layer.

The dynamic power consumption of a fat wire-based bus is mainly due to its large line capacitance. This capacitance is driven by a driver, with the dynamic power consumption:

$$P_{dri_{dyn}} = s_\tau \cdot f \cdot (C_{dri} + C_w) \cdot V_{dd}^2 \tag{3.49}$$

where s_τ is the switching activity caused by communication task $\tau \in \mathcal{K}$, f is the bus frequency, and C_{dri} and C_w represent the capacitance of the driver and the wire, respectively.

One way to limit the dynamic power is to transmit data at a lower voltage swing, V_{sw}, instead of using the higher bus voltage V_{dd}. Correspondingly, the dynamic power consumed by the driver is given by:

$$P_{dri_{dyn}} = s_\tau \cdot f \cdot (C_{dri} + C_w) \cdot V_{sw}^2 \tag{3.50}$$

The driver dissipates a non-negligible leakage power

$$P_{dri_{leak}} = L_g \cdot (V_{dd} \cdot K_3 \cdot e^{K_4 \cdot V_{dd}} \cdot e^{K_5 \cdot V_{bs}} + |V_{bs}| \cdot I_{Ju}) \tag{3.51}$$

Since the lower swing corresponds to lower signal values, a receiver has to restore the "original" signal. This requires an amplification, for which a dynamic and a leakage power consumption can be calculated as:

$$P_{rec_{dyn}} = s_\tau \cdot f \cdot C_{rec} \cdot V_{dd}^2 \tag{3.52}$$

$$P_{rec_{leak}} = L_g \cdot (V_{dd} \cdot K_3 \cdot e^{K_4 \cdot V_{dd}} \cdot e^{K_L \cdot (V_{dd}/2 - V_{sw}/2)} \cdot e^{K_5 \cdot V_{bs}} + |V_{bs}| \cdot I_{Ju}) \tag{3.53}$$

Please note that the leakage power exponentially depends on the difference between the bus voltage V_{dd} and the voltage swing V_{sw} (K_L is a technology dependent parameter), i.e., a lower voltage swing results in a higher static energy (while the dynamic power is reduced, Eq. 3.50). In order to find the most efficient solution we need to find an appropriate voltage swing that minimizes the total bus power $P_{bus} = P_{dri_{dyn}} + P_{dri_{leak}} + P_{rec_{dyn}} + P_{rec_{leak}}$. Using the optimal voltage swing can significantly reduce the power consumption of the bus [CS02, Sve01].

The speed at which the data can be transmitted over the fat wires can be considered to be independent of the voltage swing V_{sw}. Yet, the bus driver and receiver circuits introduce a delay that depends on the voltages V_{dd} and V_{bs}. This delay d and the corresponding operational frequency can be calculated according to Eq. (3.3). In order to lower the power dissipation of the drivers and receivers, it is possible to reduce V_{dd} and/or to increase V_{bs}, which, in turn, necessitates the reduction of the bus speed. However, it is important to note that the optimal voltage swing depends on the V_{dd} and V_{bs} settings of the drivers and receivers (see Fig. 3.10). Since these settings are dynamically changed during run-time via voltage selection, the value of the optimal voltage swing changes as well during run-time, and has to be adapted accordingly.

In addition to the transition overheads in terms of energy and time, which are required when scaling the voltages of the drivers and receivers (see Eq. (3.4) and (3.5)), the dynamic scaling of the voltage swing necessitates additional overheads. For a transition from V_{sw_j} to V_{sw_k} these overheads in energy and time are given by,

$$\varepsilon_{k,j} = C_{wr} \cdot (V_{sw_k} - V_{sw_j})^2 \quad \text{and} \quad \delta_{k,j} = p_{Vsw} \cdot |V_{sw_k} - V_{sw_j}| \tag{3.54}$$

where C_{wr} is the wire power rail capacitance and p_{Vsw} is the time/voltage slope.

3.9.4 Problem Formulation

We assume that all computation tasks and communications have been mapped and scheduled onto the target architecture. For each computation task $\tau_i \in \Pi$ its deadline dl_i, its worst-case number of clock cycles to be executed WNC_i, and the switched capacitance C_{eff_i} are given. Each processor can vary its supply voltage V_{dd} *and* body bias voltage V_{bs} within certain continuous ranges (for the continuous voltage selection problem), or within a set of discrete voltages pairs $m_z = \{(V_{dd_z}, V_{bs_z})\}$ (for the discrete voltage selection problem). A transition between two different performance modes on a processor requires a time and an energy overhead.

For each communication task $\tau_k \in \mathcal{K}$, the number of bytes NB_k is given. Depending on the employed bus implementation style, either using repeaters or fat wires, we have to distinguish between two subproblems:

Repeater Implementation: The communication speed as well as the communication power on bus architectures implemented through repeaters depend on the supply voltage and body bias voltage. Similar to processing elements, these voltages can be varied within a continuous range, or within a set of discrete voltage pairs $m_z = \{(V_{dd_z}, V_{bs_z})\}$, and transitions between different bus performance modes require an energy and time overhead. Furthermore, an energy overhead is required to adapt the bus voltage to the processor voltage.

Fat Wire Implementation: If communication is performed over fat wires, it is necessary to dynamically adapt the voltage swing at which data is transfered. Furthermore, in order to reduce the power dissipated by the bus drivers and receivers, it is possible to dynamically scale the supply and body bias voltage of these components. While the voltage swing can be scaled without an influence on the bus speed, the operational speed of the bus drivers and receivers is affected through voltage selection, i.e., the bus performance has to be adjusted in accordance to the driver/receiver speed. In the case of continuous voltage selection, the value for the voltage swing, the supply voltage, and the body bias voltage can be changed within a continuous range. On the other hand, for the discrete voltage selection case, the components operate across sets of discrete voltages, referred to as modes. For the voltage swing this set is $n_z = \{V_{sw_z}\}$ and for the bus drivers and receiver the set is $m_z = \{(V_{dd_z}, V_{bs_z})\}$. Of course, changing the voltage swing value as well as the supply and body bias voltages requires an energy and time overhead. □

Our overall goal is to find mode assignments for each processing and communication task, such that the individual task deadlines are satisfied and the total energy consumption, including overheads, is minimal.

3.9.5 Voltage Selection with Processors and Communication Links

We introduce a nonlinear programming model of the continuous voltage selection problem formulated in Section 3.9.4 which is optimally solvable in polynomial time, as follows:
Minimize

$$\underbrace{\sum_{k}^{|\Pi|} E_{dyn_k} + E_{leak_k}}_{\text{computation}} + \underbrace{\sum_{k}^{|\mathcal{K}|} E_{dyn_k} + E_{leak_k}}_{\text{communication}} + \underbrace{\sum_{(k,j)\in\mathcal{E}^{\bullet}} \varepsilon_{k,j}}_{\text{overhead}} \tag{3.55}$$

subject to

$$t_k = \begin{cases} WNC_k \cdot \frac{(K_6 \cdot L_d \cdot V_{dd_k})}{((1+K_1)\cdot V_{dd_k} + K_2 \cdot V_{bs_k} - V_{th_1})^{\alpha}} & \text{if } \tau_k \in \Pi \\ \left\lceil \frac{NB_k}{W_{bus}} \right\rceil \cdot \frac{(K_6 \cdot L_d \cdot V_{dd_k})}{((1+K_1)\cdot V_{dd_k} + K_2 \cdot V_{bs_k} - V_{th_1})^{\alpha}} & \text{if } \tau_k \in \mathcal{K} \end{cases} \tag{3.56}$$

$$D_k + t_k \leq D_l \quad \forall (k,l) \in \mathcal{E} \tag{3.57}$$

$$D_k + t_k + \delta_{k,l} \leq D_l \quad \forall (k,l) \in \mathcal{E}^{\bullet} \tag{3.58}$$

$$D_k + t_k \leq dl_k \quad \forall \tau_k \in \Pi \text{ with a deadline} \tag{3.59}$$

$$D_k \geq 0 \tag{3.60}$$

$$V_{dd_{min}} \leq V_{dd_k} \leq V_{dd_{max}} \tag{3.61}$$

$$V_{bs_{min}} \leq V_{bs_k} \leq V_{bs_{max}} \tag{3.62}$$

$$V_{sw_{min}} \leq V_{sw_k} \leq V_{sw_{max}} \tag{3.63}$$

The variables that need to be determined are the task and communication execution times t_k, the start times D_k, as well as the voltages V_{dd_k}, V_{bs_k}, and V_{sw_k}. The whole formulation can be explained as follows. The total energy consumption (Eq. (3.55)), with its three contributors (energy consumption of tasks, communication, and voltage transitions) has to be minimized. For all these energies both their dynamic and active leakage components are considered. The dynamic energy of tasks and communications is given by the following equations (derived from the equations discussed in Section 3.3):

$$E_{dyn_k} = \begin{cases} WNC_k \cdot s_k \cdot C_{eff_k} \cdot V_{dd_k}^2 & \text{if } \tau_k \in \Pi \\ \sum^{N} \left\lceil \frac{NB_k}{W_{bus}} \right\rceil \cdot s_k \cdot C_{rep} \cdot V_{dd_k}^2 & \text{if } \tau_k \in \mathcal{K} \text{ on repeaters} \\ \left\lceil \frac{NB_k}{W_{bus}} \right\rceil \cdot s_k \cdot C_{fat} \cdot V_{sw_k}^2 & \text{if } \tau_k \in \mathcal{K} \text{ on fat wires} \end{cases} \tag{3.64}$$

where $C_{rep} = C_d + C_w + C_g$ and $C_{fat} = C_{dri} + C_w + C_{rec}$ are the total capacitances that have to be charged by bus implementation either repeater-based or fat wire-based, respectively.

The leakage power dissipation of processors and repeater-based buses is:

$$E_{leak_k} = L_g(K_3 \cdot V_{dd_k} \cdot e^{K_4 \cdot V_{dd_k}} \cdot e^{K_5 \cdot V_{bs_k}} + I_{Ju} \cdot |V_{bs_k}|) \cdot t_k \tag{3.65}$$

For fat wire-based buses we need to additionally account for the leakage in the receiver (see Eq. (3.51) and (3.53)), given by,

$$E_{leak_k} = (P_{dri_{leak}} + P_{rec_{leak}}) \cdot t_k \tag{3.66}$$

The energy overhead due to voltage transitions is given by Eq. (3.4) and (3.54).

The constraints are similar to the ones in Section 3.6, expressing the execution order imposed by the scheduling and task graph dependencies, as well as the time constraints.

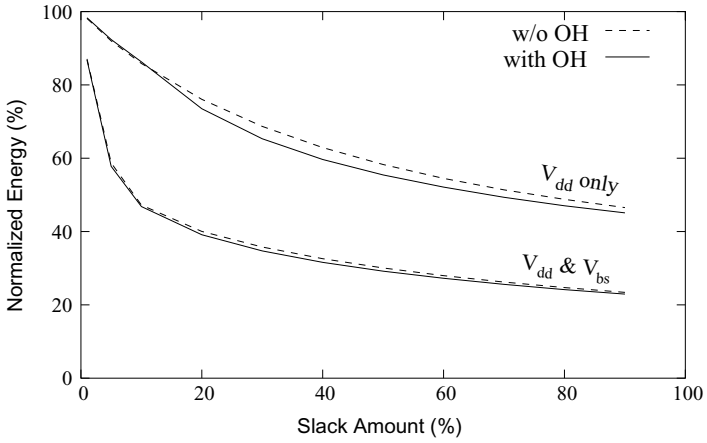
We use a heuristic similar to the one presented in Section 3.7.4 in order to translate the computed continuous voltages into the discrete ones available for the processors and buses.

3.10 Experimental Results

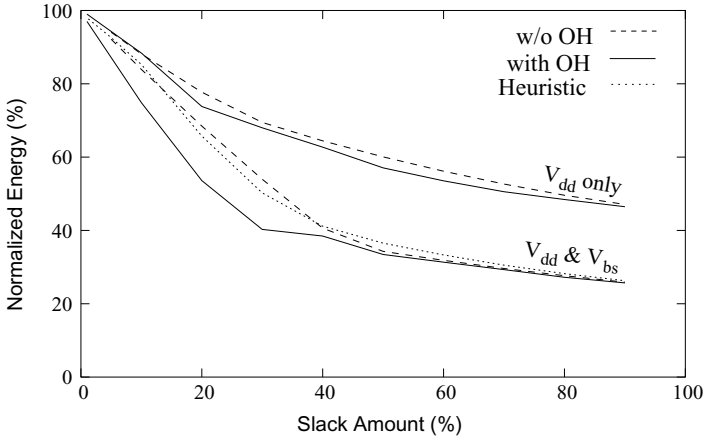
We have conducted several experiments using numerous generated benchmarks as well as two real-life examples, in order to demonstrate the efficiency of the presented approaches.

3.10.1 V_{dd} and V_{bs} Selection on the Processors

The first set of experiments was conducted in order to demonstrate the achievable energy savings when comparing the classic V_{dd} selection with simultaneous V_{dd} and V_{bs} selection. The automatically generated benchmarks consist of 100 task graphs containing between 50 and 150 tasks, which are mapped and scheduled onto architectures composed of 2 to 3 processors (we have considered that all processors are Crusoe TM5600). The technology dependent parameters of these processors were considered to correspond to a CMOS fabrication in $65nm$, for which the leakage power represents 50% of the total power consumed, [MFMB02]. For experimental purpose the amount of deadline slack in each benchmark was varied over a range 0 to 90%, using a 10% increment, resulting in 900 performed evaluations. The continuous voltage ranges were set to $0.6V \leq V_{dd} \leq 1.8V$ and $-1V \leq V_{bs} \leq 0$. The values for C_r, C_s, p_{Vdd}, and p_{Vbs} were set to $10\mu F$, $40\mu F$, $100\mu s/V$, and



(a) Continuous Voltage Selection



(b) Discrete Voltage Selection

Figure 3.12: Optimization Results for Processor DVS & ABB

$100\mu s/V$, respectively. Fig. 3.12(a) shows the outcomes for the continuous voltage selection with and without the consideration of transition overheads. The figure shows the percentage of total energy consumed (relative to the baseline energy) as a function of the available slack within the application. As a baseline we consider the energy consumption at the nominal (highest) voltage for V_{dd} and V_{bs}. It is easy to observe the advantage of the combined voltage selection scheme over the classical voltage selection, with a difference of up to 40%. These observations hold with and without the consideration of overheads. Regarding the influence of the overhead on the overall energy consumption, we can see that the savings are around 1% for the combined scheme and 2% for the Vdd-only selection. These moderate amounts of additional savings have a straightforward explanation: Within the continuous scheme (which from a practical point of view is unrealistic), the voltage differences between tasks are likely to be small, i.e., large overheads are avoided (see Eq. (3.4) and Eq. (3.5)).

We have further evaluated the discrete voltage selection scheme. Here the processors could switch between three different voltage settings $(1.8, 0)$, $(1.5, -0.4)$, and $(1.2, -0.6)$ for the combined scheme, and 1.8, 1.5, and 1.2 for the classical V_{dd} selection. The results are given in Fig 3.12(b). As in the continuous case, we can observe the difference between the classical supply voltage selection and the more efficient combined selection scheme. For low amounts of slack (around 10%), the savings for the combined selection are significantly lower than in the continuous case. The reason for this is that, due to the small slack available, the processors have to run in the highest voltage mode, which does not reduce leakage power. Further, we can see that with increasing slack, the overall energy approaches the theoretical minimum given by the continuous case, since more time is spent in the energy-efficient mode m_3. It is interesting to observe the influence of the transition overheads, in particular when not much system slack is available. In this situation the unnecessary switching between voltages to exploit the "small" amounts of slack causes an increased energy overhead. Compare, for instance, the cases where the combined V_{dd} and V_{bs} selection has been optimized with and without considering the overheads. Between 10% to 40% of slack, the consideration of transition overheads results in solutions with up to 12% higher savings. Of course, with an increasing amount of slack, the number of tasks executed at the lowest voltage setting increases, and hence the number of transitions is decreased. As a result, the influence of the transition overheads reduces.

It should be noted that the reported results for the discrete scheme have been evaluated using graphs with at most 80 tasks (without overhead, DNOH) and 40 tasks (with overhead, DOH), since the required optimization times become intractable, as a result of the NP-hardness of the problem (Section 3.7.1). To overcome this problem we have additionally investigated the voltage selection heuristic

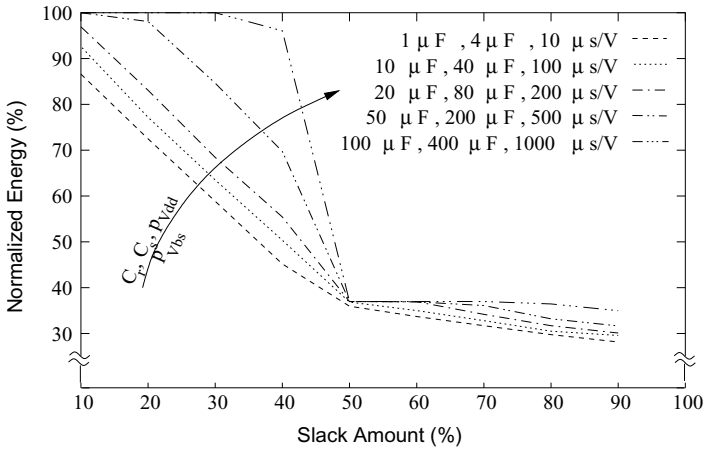


Figure 3.13: Influence of voltage selection overheads

proposed in Section 3.7.4. The results of the heuristic are shown by the dotted line in Fig 3.12(b) and, as can be seen, they are close to the optimal (maximum 8% deviation) solution. Moreover, due to its relatively reduced polynomial time complexity, it can be applied to large instances of the problem. At this point it is interesting to note that the optimization times for individual applications with up to 300 tasks using continuous voltage selection were below 1 minute, using the MOSEK solver [MOS] on a 2GHz AMD Athlon PC. Typically, for task graphs with less then 100 tasks, the optimization time is below 15 seconds. The discrete voltage selection without the consideration of the transition overheads, runs between 5 and 20 minutes, for tasks graphs with less then 90 tasks. When considering the overheads during the discrete optimization, an important parameter that affects the optimization time, besides the number of tasks, is the number of execution modes. We were not able to solve optimally task graphs with more then 30 tasks, considering 3 or more execution modes. Even for such a small number of tasks, the optimization time is around 1 hour. The proposed heuristic for discrete voltages, however, has a runtime comparable to the continuous voltage optimization, making it suitable for large applications.

3.10.2 Significance of Transition Overheads

In order to further investigate the influence of transition overheads, we have carried out an additional set of experiments in which the amount of the processors' overheads in terms of energy and delay were varied by adjusting the values for C_r,

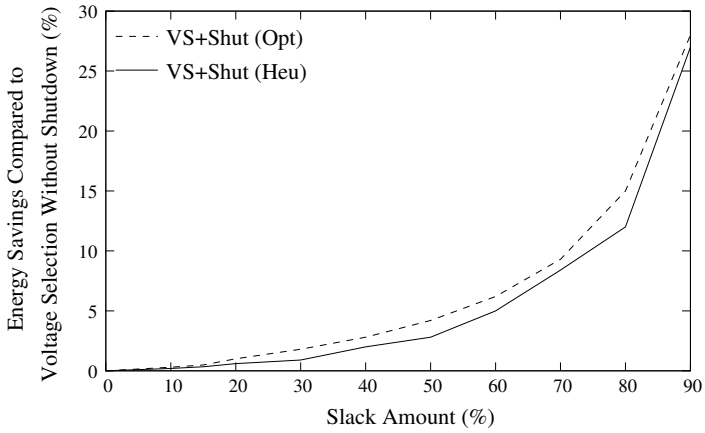


Figure 3.14: Voltage Selection with Shutdown

C_s, p_{Vdd}, and p_{Vbs} (see Section 3.3). In accordance, we use the discrete voltage selection with consideration of overheads. The results are given in Fig 3.13. As expected, the energy dissipation increases for higher values of the overhead determining parameters. For instance, while a processor which requires $C_r = 1\mu F$, $C_s = 4\mu F$, $p_{Vdd} = 10\mu s/V$, and $p_{Vbs} = 10\mu s/V$ transition overheads can reduce the energy consumption by 56% if 40% of slack is available, another processor with $C_r = 20\mu F$, $C_s = 80\mu F$, $p_{Vdd} = 200\mu s/V$, and $p_{Vbs} = 200\mu s/V$ achieves only 42%. This highlights the importance to carefully consider the influence of transition overheads.

3.10.3 Voltage Selection with Processor Shutdown

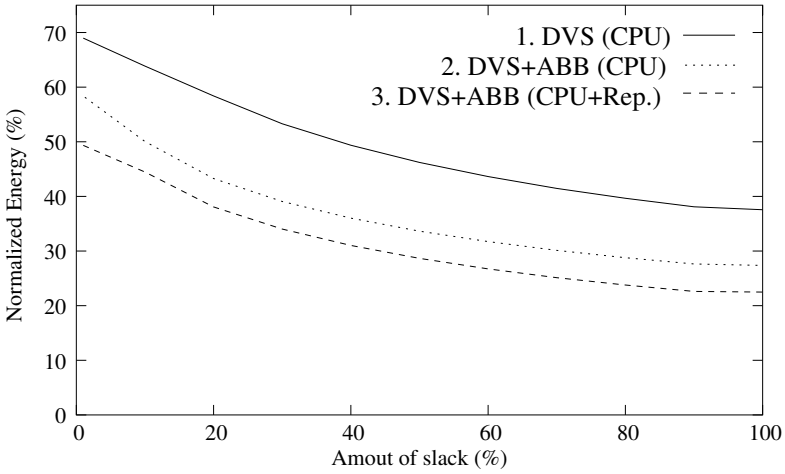
Using the same setup as in the previous experiments, we have studied the achievable energy savings that can be obtained by using the proposed voltage selection with shutdown, presented in Section 3.8.2. We have assumed that the overheads for a shutdown operation are $E_{soh} = 300\mu J$ and $t_{soh} = 1ms$, as in [RJ05]. The results are presented in Fig. 3.14. On the X axis, we have varied the amount of available deadline slack. We plotted with the continuous line the energy savings achievable by the combined voltage selection and shutdown heuristic presented in Section 3.8.2, relative to a system that is optimized using solely DVS and ABB, without shutdown. In order the measure the quality of the heuristic, we have also represented with a dotted line the results obtained by an optimal solution. As we

can observe from Fig. 3.14, if the amount of slack is low, shutting down does not yield additional energy savings. However, the additional benefit of the shutdown is significant for larger amounts of slack. For example, for systems having 30% slack, the additional savings obtained with shutdown, relative to DVS and ABB are only 2%. When the available slack is above 60%, the savings due to shutdown range from 10% to almost 30%. It is interesting to note that the proposed heuristic yields results that are close to the optimal solution.

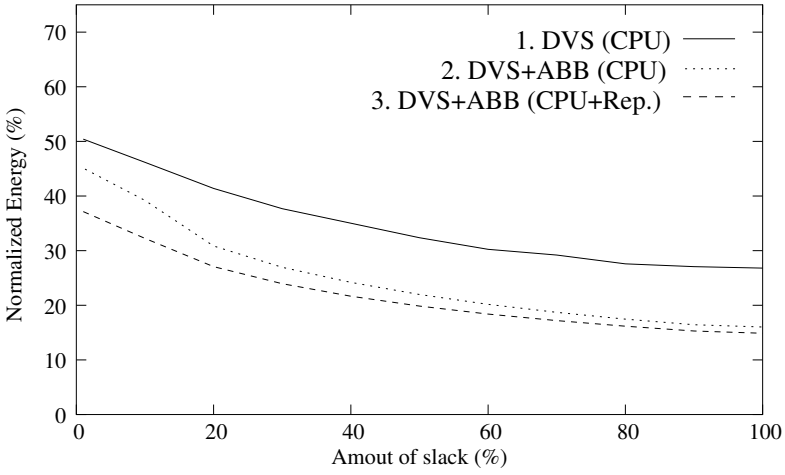
3.10.4 Combined Voltage Selection for Processors and Communication

We have conducted a set of experiments in order to validate the presented techniques for combined processor and bus voltage selection. The automatically generated benchmarks consist of 120 task graphs containing between 50 and 300 tasks, which are mapped and scheduled onto architectures composed of 2 to 5 processors, interconnected via 1 to 4 buses either implemented repeater-based or fat wire-based. The continuous voltage ranges were set to $0.6V \leq V_{dd} \leq 1.8V$ and $-1V \leq V_{bs} \leq 0$, while the discrete voltage levels are $m_z = \{(1.8,0),(1.4,-0.2),(0.8,-0.6),(0.6,-1)\}$. The voltage ranges for repeater-based systems are identical to the possible processor voltage settings. For the fat wire-based buses the continuous voltage swing can be set between 0.2 and $1V$, and for the discrete case it can be adjusted to $m_z = 0.2, 0.3, 0.4, 0.6, 1V$. The amount of deadline slack in each benchmark was varied over a range 0 to 100%, using a 10% increment. Furthermore, the amount of communication within the generated benchmarks was varied between 10 to 50% of the total execution time, with an increment of 10%. Overall, these experiments resulted in 2400 performed evaluations.

The first set of experiments was conducted with the aim to investigate the energy savings that are achievable when dynamically scaling the supply voltage as well as body bias voltage of bus repeaters. The 32bit-wide bus architecture under consideration consisted of 27 repeaters per bit-line of which each has a total length of $27.4mm$. The capacitance of a single wire including the repeaters was estimated as $7.2pF$, using the power optimized data from [BM02]. Fig. 3.15(a) shows the outcomes of three system configurations for different amounts of system slack. All plots have been normalized against the energy dissipation at nominal (highest) voltages. The first plot gives the energy consumption for systems in which the repeaters' voltages are kept fixed, while the supply voltage (but not the body-bias voltage) of the processors is dynamically scaled. The second plot represents a system in which the repeater settings are still kept fixed, while combined V_{dd} and V_{bs} scaling is applied to the processors. The third plot indicates the systems in which



(a) Energy dissipation of a reapter–based system



(b) Energy dissipation of a fat wire–based system

Figure 3.15: Optimization Results for Different Bus Implementations

the repeater-based bus as well as the processors are scaled by changing V_{dd} and V_{bs}. Please note that Fig. 3.15(a) gives the energy values for systems with a communi-

cation amount of 30%, compared to the total execution time. Inspecting the graphs reveals that the highest energy savings are achieved by considering the combined V_{dd} and V_{bs} continuous voltage selection scheme on the buses as well as on the processors (plot 3). We can also observe that the energy efficiency is increased by approx. 12% if combined voltage selection is applied on the bus (difference between plot 2 and 3). Generally, the combined V_{dd} and V_{bs} scaling yields higher energy saving (around 30%) than the V_{dd}-only scaling (difference between plot 1 and 2). Since all plots in Fig. 3.15(a) represent the results for continuous voltage selection, it is interesting to note that the proposed heuristic for discrete voltage selection (Section 3.7.4) achieves results that are within 4% of the values obtained at continuous voltage levels. It is important to note that the efficiency difference of about 12% on average, between implementations with and without bus voltage selection is preserved also when discrete voltage levels are used.

In the second set of experiments, shown in Fig. 3.15(b), we investigate the achievable energy savings on a fat wire-based bus system, assuming the same bus-width as in the previous experiment. Since fat wires are considered to be suitable only for short distance connections, we consider a length of $4mm$ with a single line capacitance of $609fF$. Similarly to the previous experiments, the plots 1 and 2 represent systems in which only the processing elements are scaled (V_{dd} only for plot 1 and combined V_{dd} and V_{bs} for plot 2), while the third plot indicates systems in which the processors and buses are voltage scaled in terms of V_{dd}, V_{bs}, and V_{sw}. As expected, the fully voltage scalable systems, achieve the best energy savings, with reductions between 4% to 18% compared to systems with fixed bus voltages. Again, applying the heuristic for discrete voltage selection shows that results comparable to the continuous case (within 4%) can be achieved.

Please note that we do not advocate here repeater-based or fat wire-based approaches and do not try to show that one is better than the other. What we do show is that energy savings can be achieved if voltage selection is applied on the communication links and that the communication energy models are highly dependent on the actual technique used to implement the communication lines. The experiments have also shown that with an increasing amount of communication data, the bus voltage selection approach achieves increasingly higher energy reductions. If, for example, the time spent for communications is around 15% of the total execution time, the energy savings due to bus voltage scaling are around 10%. With communication time around 30%, the energy savings become around 16%.

3.10.5 Real-Life Examples

We have conducted experiments on two real-life applications: a GSM voice codec and a generic multimedia system (MMS), that includes a H263 video encoder and

decoder and MP3 audio encoder and decoder . Details regarding these applications can be found in [SAHE04] and [HM03]. The GSM voice codec consists of 87 tasks and is considered to run on an architecture composed of 3 processing elements with two voltage modes ($(1.8V, -0.1V)$ and $(1.0V, -0.6)$). At the highest voltage mode, the application reveals a deadline slack close to 10%. Switching overheads are characterized by $C_r = 1\mu F$, $C_s = 4\mu F$, $p_{Vdd} = 10\mu s/V$, and $p_{Vbs} = 10\mu s/V$. Tab. 3.1 shows the results in terms of dynamic E_{dyn}, leakage E_{leak}, overhead ε, and total energy E_{active} (Columns 2–5). Each line represents a different

Approach	E_{dyn} (mJ)	E_{leak} (mJ)	ε (mJ)	E_{active} (mJ)	*Reduction* (%)
Nominal	1.342	0.620	non	1.962	—
DVDDNOH	1.185	0.560	0.047	1.792	8.7
DVDDOH	1.190	0.560	0.003	1.753	10.7
DNOH	1.253	0.230	0.048	1.531	22.0
DOH	1.255	0.230	0.002	1.487	24.3
Heuristic	1.271	0.250	0.008	1.529	22.1

Table 3.1: Optimization results for the GSM codec

voltage selection approach. Line 2 (Nominal) is used as a baseline and corresponds to an execution at the nominal voltages. Lines 3 and 4 give the results for the classical V_{dd} selection, without (DVDDNOH) and with (DVDDOH) the consideration of overheads. As we can see, the consideration of overheads achieves higher energy saving (10.7%) than the overhead neglecting optimization (8.7%). The results given in lines 5 and 6 correspond to the combined V_{dd} and V_{bs} selection schemes. Again we distinguish between overheads neglecting (DNOH) and overhead considering (DOH) approaches. If the overheads are neglected, the energy consumption can be reduced by 22%, yet taking the overheads into account results in a reduction of 24.3%, solely achieved by decreasing the transition overheads. Compared to the classical voltage selection scheme, the combined selection achieved a further reduction of 14%. The last line shows the results of the proposed heuristic approach. It should be noted that, since the problem is NP hard, such heuristic techniques are needed when dealing with larger cases (increased number of voltage modes and tasks). In the GSM application, although the number of tasks is relatively large, we considered only two voltage modes. Therefore the optimal solutions could be obtained for the DOH problem.

We have performed the same set of experiments on the MMS system consisting of 38 tasks that is considered to run on an architecture composed of 4 processors with four voltage modes ($(1.8V, 0.0V)$, $(1.6V, -0.8)$, $(1.3V, -0.9)$ and

$(1.0V, -0.9)$). At the highest voltage mode, the application reveals a deadline slack close to 40%. Tab. 3.2 shows the results in terms of dynamic E_{dyn}, leakage E_{leak}, overhead ε, and total E_{active} energy (Columns 2–5). As with the GSM, the con-

Approach	E_{dyn} (mJ)	E_{leak} (mJ)	ε (mJ)	E_{active} (mJ)	*Reduction* (%)
Nominal	14.88	12.05	non	26.93	—
DVDDNOH	11.33	9.45	0.68	21.46	20.4
DVDDOH	11.31	9.46	0.0001	20.77	22.9
DNOH	11.40	7.18	0.89	19.47	27.7
DOH	11.41	7.18	0.01	18.60	31.0
Heuristic	11.62	7.30	0.40	19.32	29.3

Table 3.2: Optimization results for the MMS system

sideration of overheads achieves higher energy savings (22.9% for the Vdd-only selection and respectively 31.0% for the combined approach) than the overhead neglecting optimization (20.4 and respectively 27.7%). Compared to the classical voltage selection scheme (22.9% savings), the combined selection achieved a further reduction of 8.1%.

We have performed a set of experiments on each of the two real-life applications in order to show the efficiency of the proposed voltage selection with processor shutdown technique. The voltage modes are the same for GSM codec and respectively for the MMS system as the ones used in the previous experiments. The results are presented in tables 3.3 and 3.4. Each line represents a different approach. The first line (Nominal) is the baseline and represents an execution at the highest voltages, without any processor shutdown. The remaining four lines represent the resulting energy consumptions for supply voltage selection without (DVddNoSH) and with shutdown (DVddSH) and respectively the supply and body bias selection without (DVddVbsNoSH) and with shutdown (DVddVbsSH). For each approach we list the active (E_{active}), idle and total energy (E_{idle}) consumption. The overheads for a shutdown operation are estimated in [RJ05] as $E_{soh} = 300\mu J$ and $t_{soh} = 1ms$. If we use these values for the GSM voice codec, we cannot perform any shutdown, due to the little amount of slack available after voltage selection. If we consider lower shutdown overheads ($E_{soh} = 90\mu J$ and $t_{soh} = 0.3ms$), we obtain the results presented in table 3.3. As we can see, even considering a reduced overhead, the energy can be improved via shutdown by only 4%. It is interesting to compare the active and idle energy values resulted after performing voltage selection without and with processor shutdown from the lines 4 and 5 in table 3.3. As we can see, the active energy is slightly increased when we perform the shut-

down (from 1.48mJ to 1.50mJ), while the idle energy is reduced (from 0.93mJ to 0.70mJ). This means that a situation similar to the one described in Fig. 3.7 is encountered during the optimization (the voltages for a task are increased in order to allow the merging of several idle intervals into one big shutdown period). The difference between the total energy (E_{total}) and the sum of active (E_{active}) and idle (E_{idle}) energies represents the energy corresponding to the shutdown overheads plus the low energy consumed in the shutdown state. A simple calculation shows that only one shutdown is perfomed in case of the GSM voice codec.

A similar experiment was performed for the MMS. We have used the shutdown overheads estimated in [RJ05] ($E_{soh} = 300\mu J$ and $t_{soh} = 1ms$). The results are presented in table 3.4. It is interesting to note that performing shutdown in conjunction with only supply voltage selection provides a reduction of 9%, compared to a reduction of 5% obtained by the shutdown with the combined V_{dd} and V_{bs} selection. This is due to the fact that the combined supply and body bias voltage selection exploits more slack than the supply-only voltage selection, thus leaving less idle time for potential shutdown operations. As opposed to the GSM voice codec, the optimization determines 5 shutdowns for the MMS.

Approach	E_{active} (mJ)	E_{idle} (mJ)	E_{total} (mJ)	*Reduction* (%)
Nominal	1.96	1.02	2.98	—
DVddNoSH	1.74	0.93	2.68	10
DVddSH	1.75	0.62	2.56	14
DVddVbsNoSH	1.48	0.93	2.41	19
DVddVbsSH	1.50	0.70	2.30	23

Table 3.3: Results for the GSM codec with shutdown

Approach	E_{active} (mJ)	E_{idle} (mJ)	E_{total} (mJ)	*Reduction* (%)
Nominal	26.93	6.94	33.87	—
DVddNoSH	20.78	4.83	25.61	25
DVddSH	20.83	0.20	22.53	34
DVddVbsNoSH	18.55	4.78	23.33	32
DVddVbsSH	19.85	0.20	21.56	37

Table 3.4: Results for the MMS system with shutdown

In the previous experiments, communication energy has been ignored. Another set of experiments was performed on the two benchmarks in order to high-

light the importance of combined processor and communication links' scaling. The GSM codec is considered to run on an architecture composed of 3 processors (with two voltage modes ($(1.8V, -0.1V)$ and $(1.0V, -0.6V)$)), communicating over a repeater-based shared bus. At the nomimal voltages, the communication accounts for 15% of the total energy consumption. Tab. 3.5 shows the resulting total energy consumptions for six different situations. The first column denotes the used

Approach	VS type	E_{tot} (mJ)	Reduc. (%)
Nominal	—	2.273	—
CPU (V_{dd})	cont.	2.091	9
CPU (V_{dd},V_{bs})	cont.	1.831	20
Heu.CPU (V_{dd},V_{bs})	disc.	1.887	17
CPU+BUS (V_{dd},V_{bs})	cont.	1.665	27
Heu.CPU+BUS(V_{dd},V_{bs})	disc.	1.723	24

Table 3.5: Results for the GSM codec considering the communication

voltage selection technique and the second indicates if continuous or discrete voltages were considered. The third and fourth column give the energy consumption and achieved reduction in percentage for each scaling approach. For instance, according to the second row, the system dissipates an energy of $2.273\mu J$ at nominal voltage settings, i.e., without any voltage selection. This value serves as a baseline for the reductions indicated in the fourth column. The third and fourths row present the results of systems in which the bus remains unscaled while the processors are either V_{dd} or V_{dd} and V_{bs} scaled over a continuous range. As we can observe, savings of 9 and 20% are achieved. In order to adapt the continuous selected voltages towards the two discrete voltage settings at which the processor can possibly run, we apply our heuristic outlined in Section 3.7.4. The achieved reduction in the discrete case is 17% (row 5). Nevertheless, as shown by the values given in row 6, it is possible to further reduce the energy by scaling the repeater-based bus. Compared to the baseline, a saving of 27% is achieved. Using the discrete voltage heuristic, the final energy dissipation results in $1.723\mu J$, which is 24% below the unscaled system. The MMS system is mapped on 4 processors that communicate over two repeater-based buses. At the nomimal voltages, the communication accounts for 25% of the total energy consumption. The results are presented in table 3.6.

In this chapter we have focused on the voltage selection problem. The solutions presented and the heuristics proposed can be included in design space exploration frameworks that also perform other system level optimizations, such as task mapping and scheduling. This has been demonstrated by integrating our work in the

Approach	VS type	E_{tot} (mJ)	Reduc. (%)
Nominal	—	35.01	—
CPU (V_{dd})	cont.	28.99	18
CPU (V_{dd},V_{bs})	cont.	26.05	26
Heu.CPU (V_{dd},V_{bs})	disc.	26.82	24
CPU+BUS (V_{dd},V_{bs})	cont.	22.94	35
Heu.CPU+BUS(V_{dd},V_{bs})	disc.	23.48	33

Table 3.6: Results for the MMS system considering the communication

frameworks proposed in [RGA+06, SHE05]. We will discuss this aspect in Chapter 4.

Chapter 4

Mapping, Scheduling and Voltage Selection

Multiprocessor Systems-on-Chip (MPSoCs) represent today the main trend for architectural designs, since they are able to provide scalable computation power while still retaining the flexibility to support different task mixes [Wol05]. In this chapter we present two system level energy optimization frameworks that integrate task mapping, scheduling and voltage selection. The results are validated through the optimization of a GSM voice codec that is implemented on a cycle accurate simulation platform. The energies predicted by the optimization flow match the ones measured on the simulator. This chapter is structured as follows: we first describe previous work in the field. The target architecture and the virtual platform environment are presented in Section 4.2. Section 4.3 gives the problem formulation, followed by an optimal solution in Section 4.4 and a genetic-based heuristic in Section 4.5. Discussions on computational efficiency, validation and experimental results conclude the chapter.

The main goal of this chapter is to demonstrate how voltage selection techniques can be integrated in a broader system-level design flow. This work has been done together with Martino Ruggiero, Pari Gioia, Guerri Alessio, Luca Benini , Michela Milano and Davide Bertozzi from Bologna University, Italy [RGA+06], and, with Marcus Schmitz and Bashir M. Al-Hashimi from University of Southampton [AEP+07b].

4.1 Introduction and Related Work

Task mapping and scheduling are combinatorial optimization problems and have been shown to be NP complete [GJ79]. Traditionally, there are two main approaches to these problems:

- Using optimal algorithms [PP92, RGA+06, BGM+06, LTK04], such as Integer Linear Programming (ILP) or Constraint Programming (CP) formulations. Due to the problem complexity, such approaches must be carefully deployed. For example, it is known that scheduling problems are not efficiently tackled by ILP approaches. This is due to extra complexity introduced in the ILP model in order to be able to capture precedence constraints.

- Deployment of heuristic methods [LJ07, VM03, HM03, SHE05, SAHE04, SAHE02, DJ98, BTT98, DJ99, ACD74, WG90, OH96, SL93, KA99, BJM97, EDPP00] to provide good (even if not optimal) solutions. However, heuristic algorithms can still impose significant computational requirements without guarantees on the quality of final solutions. Well-known heuristic techniques include genetic algorithms, simulated annealing [OvG89, Ree93] and tabu search [Glo89, Glo90].

Assuming the mapping of the tasks on the processors is given as input, the authors from [GK01, GK03, Gru01] present a scheduling technique that maximizes the available slack, which is then used to reduce the energy via voltage scaling. [LJ07] proposes a scheduling algorithm based on simulated annealing and a heuristic based on energy gradients for voltage scaling. The allocation of the tasks on the processors (mapping) has a great influence on the energy consumption. [SHE05, SAHE02, SAHE04] present a heuristic approach for mapping, scheduling and voltage scaling on multiprocessor architectures. In the context of a network-on-chip platform, [HM03] presented a mapping and scheduling algorithm for tasks and communications with the objective of minimizing the energy. They use a sub-optimal heuristic and do not consider voltage-scalable cores. The closest approach to the work presented in this chapter is the one from [LTK04]. They propose a mixed integer linear programming (MILP) formulation for mapping, scheduling and continuous voltage selection.

We present in this chapter two approaches for the mapping, scheduling and voltage selection problem (MSDVS): one that is based on exact algorithms [RGA+06] and one based on a genetic heuristic [AEP+07b]. In both approaches, the voltage selection techniques introduced in Section 3 are integrated in the system level optimization framework.

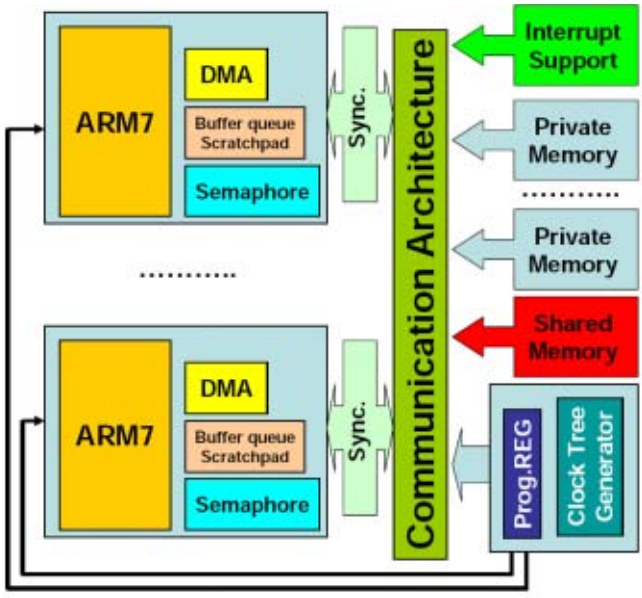


Figure 4.1: Target Hardware Architecture

4.2 Hardware Architecture Model

The target architecture illustrated in Fig. 4.1 is a general platform for a distributed MPSoC. It consists of processor cores, an AMBA AHB-compliant shared bus and a shared memory for inter-core communication. The processors are homogeneous and consist of ARM7 cores with instruction and data caches and of tightly coupled software-controlled scratch-pad memories. The MPARM virtual platform [BBB+03, LPB04, And06] is used as a cycle-accurate simulation environment for this hardware architecture. Applications compiled with a cross-compiler can be executed on the virtual platform. After the simulation, certain statistics can be collected, such as, execution times, power and energy values for the hardware components, as well as statistics regarding the memory accesses or the performance of the bus.

Messages can be exchanged by tasks through communication queues, that are allocated at design time either in scratchpad memories or in the shared memory, depending on whether tasks are mapped onto the same processor or not. Synchronization between tasks is implemented by means of two hardware semaphores. When a producer generates a message, it locally checks an integer semaphore which con-

tains the number of free messages in the queue. If a space is available, it decrements the semaphore and starts writing the message. When the message is ready, it signals this to the consumer by incrementing the consumer pointer. Distributed semaphores were implemented to avoid bus transaction overheads associated with checking centralized hardware semaphores connected to the bus.

The software support is provided by a small executive and by a set of high-level APIs to support message passing on the underlying hardware architecture. The communication and synchronization library abstracts low level architectural details to the programmers, such as memory maps or explicit management of hardware semaphores and shared memory [FA05, And06]. The virtual platform supports a set of frequencies and voltages for each processor core. For this purpose, additional modules were integrated into the platform, namely a variable clock tree generator, programmable registers and a synchronization module. The clock tree generator feeds the hardware modules of the platform with independent and frequency scaled clock trees. A set of programmable registers has been connected to the system bus: each one of these registers contains the integer divider of the baseline frequency for each processor. Finally, a synchronization module consisting of dual-clock FIFOs was designed to interface each processor (which can be frequency-scaled) to the bus, which is assumed to operate at the maximum frequency. The maximum AMBA AHB frequency of 200MHz was kept as the maximum processing core frequency, to which frequency dividers were applied.

The virtual platform environment provides power statistics for ARM cores, caches, on-chip memories and AMBA AHB bus, leveraging technology homogeneous power models for a 0.13 nm technology provided by ST Microelectronics. When all tasks mapped on a processor core are suspended, then the core enters power save mode, where the power consumption is assumed to be negligible.

4.3 Problem Formulation

MSDVSP is the problem of determining the number of processors, mapping tasks to processors, selecting the voltage/frequency mode for each task and scheduling each of them such that the resulting energy is minimized and the timing constraints are satisfied.

As input, we consider a set of tasks $\Pi = \{\tau_i\}$ with dependencies captured by a task graph $G(\Pi, \Gamma)$. Each task $\tau_i \in \Pi$ has a deadline dl_i. Edges $\gamma \in \Gamma$ indicate the dependencies between these tasks (communications). The hardware architecture consists of a set of available processors $\mathcal{P} = \{p_k\}$. For each task τ_i, the deadline dl_i is given. For each processor $p_k \in \mathcal{P}$, the worst-case number of clock cycles WNC_i^p to be executed by task τ_i is also given. In particular, without any loss in generality,

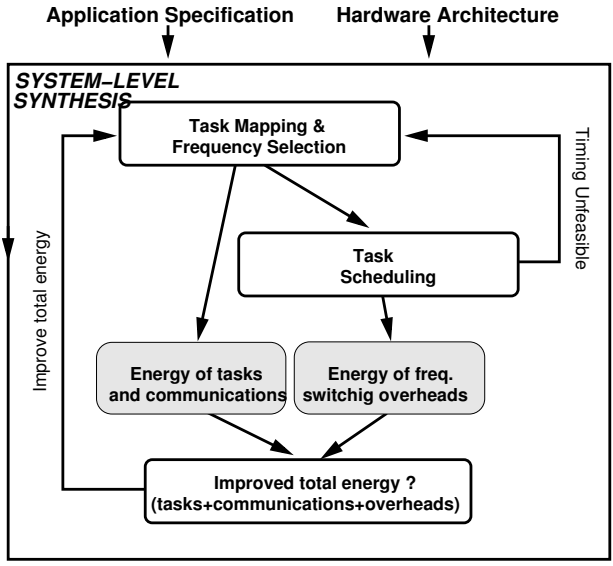


Figure 4.2: Optimal Mapping & Scheduling & Frequency Selection

in this chapter we consider a homogeneous architecture, and thus the number of clock cycles for each task does not depend on the processor. Each processor can vary its frequency f and voltages within a set of discrete set of performance modes $\mathcal{M}$. The power dissipation of each task depends on the mode $m \in \mathcal{M}$ used to execute it. Tasks are executed cycle by cycle. As opposed to Chapter 3, we assume that each task is executed using one single mode. The goal is to find a mapping, schedule and frequency/voltage assignment for each task such that the individual task deadlines are met and the total energy consumption is minimal.

4.4 Optimal Mapping, Scheduling and Dynamic Voltage Selection

Many optimization problems can be decomposed into well known, structured and widely studied sub-problems such as scheduling, packing, matching and resource allocation. These applications have been considered and solved separately by both the Operations Research (OR) and the Artificial Intelligence (AI) communities. It is widely acknowledged that exploiting the structure of these problems improves the performances of the corresponding algorithms. For example, solving schedul-

ing problems with ILP techniques is inefficient, while CP is extremely suitable. On the other hand problems like resource allocation are better dealt using ILP solvers. In general, merging different algorithmic aspects leads to an efficient solving process and may determine significant performance speed ups in finding the optimal solution. As a result, many practical problem configurations, traditionally tackled by means of heuristic methods, become now tractable by complete approaches that provide the optimal solution in reasonable time. We will present such an approach in the following.

The proposed solution is based on the principle of logic-based Benders decomposition [HO03]. The problem is decoupled in two parts: task mapping with performance mode assignment (Master Problem) and scheduling (Subproblem). The energy dissipation is minimized during the task mapping step, while mode switching overhead is minimized during the scheduling process. The scheduling part is also responsible for finding an execution order for the tasks that meets the deadlines. The optimization approach is presented in Fig. 4.2. As the picture shows, there are several mapping and scheduling iterations. The Master problem produces candidate mappings and task frequency/voltage assignments with the objective of minimizing the system energy. As there is no timing information during the mapping process, each candidate mapping has to be validated by scheduling the tasks. Moreover, the energy overhead due to mode switching can only be computed during scheduling. If several schedules are feasible for a given mapping, the one with the lowest switching energy is reported. Successive candidate mappings will have the tasks and communications energy in increasing order. If the energy overhead due to mode switching is neglected as in [BGM+06], the optimization stops when the first mapping is schedulable. However, if the energy overhead is considered, the optimization stops when a mapping with an energy consumed by the tasks and communications higher then the minimum total energy (tasks+communications+overheads) found until that point is produced.

We will present in the following the mapping and the scheduling formulations.

4.4.1 The Master Problem Model

The Master problem is formulated as an ILP. The mapping of a certain task $\tau_t \in \Pi$ on the processor $p_p \in \mathcal{P}$ is modelled using the binary variables $X_{p,t,m}$. $X_{p,t,m} = 1$ if task τ_t is mapped on the processor p_p and runs in mode m (with frequency f_m) and $X_{p,t,m} = 0$ otherwise.

The communication between tasks is modelled as follows. We assume that two tasks running on the same processor communicate over the scratchpad and thus do not require the bus (intra-processor communications). Tasks that are mapped on different processors communicate via the shared memory over the bus (inter-

processor communications). As opposed to Chapter 3, where a communication is modeled by a single message exchanged via the bus, we now decouple it in two parts: the sending task writes data to the shared memory and the receiving task reads the data from the shared memory. This difference is due to the particular MPSoC architecture used in this chapter to validate the resulting optimized system.

In order to capture the communication, the following binary variables are used:

- $InterR_{src,dst,m} = \begin{cases} 1 & \text{if } (\tau_{src}, \tau_{dst}) \in E \text{ and } \tau_{dst} \text{ runs in mode } m \\ 0 & \text{otherwise} \end{cases}$

  This variable models the fact that the tasks τ_{src} and τ_{dst} are mapped on different processors. Furthermore, the performance mode used during the reading part of the inter-processor communication is also captured. For example, if the communicating tasks τ_i and τ_j are mapped on different processors and τ_j is executed with the frequency f_k, then $InterR_{i,j,k} = 1$ and $InterR_{i,j,l} = 0, \forall l \neq k$.

- $InterW_{src,dst,m} = \begin{cases} 1 & \text{if } (\tau_{src}, \tau_{dst}) \in \Gamma \text{ and } \tau_{src} \text{ runs in mode } m \\ 0 & \text{otherwise} \end{cases}$

  This variable models the fact that the tasks τ_{src} and τ_{dst} are mapped on different processors. Furthermore, the performance mode used during the reading part of the inter-processor communication is also captured. For example, if the communicating tasks τ_i and τ_j are mapped on different processors and τ_i is executed with the frequency f_k, then $InterW_{i,j,k} = 1$ and $InterR_{i,j,l} = 0, \forall l \neq k$.

- $IntraR_{src,dst,m} = \begin{cases} 1 & \text{if } (\tau_{src}, \tau_{dst}) \in \Gamma \text{ and } \tau_{dst} \text{ runs in mode } m \\ 0 & \text{otherwise} \end{cases}$

  This variable models the fact that the tasks τ_{src} and τ_{dst} are mapped on the same processor. The frequency of the reading part of the intra-processor communication is also captured, similar to the $InterR$ variable.

- $IntraW_{src,dst,m} = \begin{cases} 1 & \text{if } (\tau_{src}, \tau_{dst}) \in \Gamma \text{ and } \tau_{src} \text{ runs in mode } m \\ 0 & \text{otherwise} \end{cases}$

  This variable models the fact that the tasks τ_{src} and τ_{dst} are mapped on the same processor. The frequency of the writing part of the intra-processor communication is also captured, similar to the $InterW$ variable.

Additionally, as an input parameter, the binary matrix $Dependency_{i,j}$ specifies, according to the input task graph, if two tasks τ_i and τ_j communicate (inter- or intra-processor).

Depending on the type of communication used, the amount of time and energy differs. The sending task τ_{src} spends time and energy to write the data to the scratchpad in case of intra-processor or to the shared memory via the bus in case of inter-processor communication. Similarly, the receiving task τ_{dst} needs time and energy to read the data from the scratchpad or from the shared memory. Both the read and write activities are performed at the same speed of the corresponding task that performs the operation. During all the transactions, the bus works at the maximum speed.

Consequently, another set of input parameters is constituted by the worst-case number of clock cycles for each read and write operation between any pair of communicating tasks (τ_{src}, τ_{dst}):

$$(WNCIntraW_{src,dst}, WNCIntraR_{src,dst}, WNCInterW_{src,dst}, WNCInterR_{src,dst})$$

In the following we present the ILP formulation:
Minimize:

$$Energy = \sum_{\tau_k} E_{task_k} + \sum_{\tau_{src},\tau_{dst}} E_{inter\_comm_{src,dst}} + \sum_{\tau_{src},\tau_{dst}} E_{intra\_comm_{src,dst}} + E_{oh} \quad (4.1)$$

Such that:

$$P_{k,m} = P_{dnom_m} \cdot Ceff_k + P_{leak_m} \forall \tau_k \in \Pi, m \in \mathcal{M} \quad (4.2)$$

$$E_{task_k} = \sum_{p=1}^{|\mathcal{P}|} \sum_{m=1}^{|\mathcal{M}|} X_{p,k,m} \cdot WNC_k \cdot P_{k,m} \forall \tau_k \in \Pi \quad (4.3)$$

$$E_{inter\_comm_{src,dst}} = \sum_{m=1}^{|\mathcal{M}|} InterW_{src,dst,m} \cdot (P_{src,m} + P_{bus}) \cdot \frac{WNCInterW_{src,dst}}{f_m}) + \quad (4.4)$$
$$+ \sum_{m=1}^{|\mathcal{M}|} InterR_{src,dst,m} \cdot (P_{dst,m} + P_{bus}) \cdot \frac{WNCInterR_{src,dst}}{f_m})$$

$$E_{intra\_comm_{src,dst}} = \sum_{m=1}^{|\mathcal{M}|} IntraW_{src,dst,m} \cdot (P_{src,m}) \cdot \frac{WNCIntraW_{src,dst}}{f_m}) + \quad (4.5)$$
$$+ \sum_{m=1}^{|\mathcal{M}|} IntraR_{src,dst,m} \cdot (P_{dst,m}) \cdot \frac{WNCIntraR_{src,dst}}{f_m})$$

$$\sum_{p=1}^{|\mathcal{P}|}\sum_{m=1}^{|\mathcal{M}|} X_{p,k,m} = 1 \;\; \forall \tau_k \in \Pi \tag{4.6}$$

$$\sum_{m=1}^{|\mathcal{M}|} InterR_{src,dst,m} = \frac{1}{2}\cdot \sum_{p=1}^{|\mathcal{P}|} \mid \sum_{m=1}^{|\mathcal{M}|} X_{p,src,m} - \sum_{m=1}^{|\mathcal{M}|} X_{p,dst,m} \mid (\tau_{src},\tau_{dst}) \in \mathcal{E} \tag{4.7}$$

$$\sum_{m=1}^{|\mathcal{M}|} InterW_{src,dst,m} = \frac{1}{2}\cdot \sum_{p=1}^{|\mathcal{P}|} \mid \sum_{m=1}^{|\mathcal{M}|} X_{p,src,m} - \sum_{m=1}^{|\mathcal{M}|} X_{p,dst,m} \mid (\tau_{src},\tau_{dst}) \in \mathcal{E} \tag{4.8}$$

$$InterR_{src,dst,m} \geq \sum_{p=1}^{\mathcal{P}} X_{p,dst,m} \;\; \tau_{src},\tau_{dst} \in \mathcal{E}, m \in \mathcal{M} \tag{4.9}$$

$$InterW_{src,dst,m} \geq \sum_{p=1}^{\mathcal{P}} X_{p,src,m} \;\; \tau_{src},\tau_{dst} \in \mathcal{E}, m \in \mathcal{M} \tag{4.10}$$

$$\sum_{m=1}^{|\mathcal{M}|} IntraR_{src,dst,m} = Dependency_{src,dst} - \sum_{m=1}^{|\mathcal{M}|} InterR_{src,dst,m} \tag{4.11}$$

$$IntraR_{src,dst,m} \geq \sum_{p=1}^{\mathcal{P}} X_{p,dst,m} \tag{4.12}$$

$$\sum_{m=1}^{|\mathcal{M}|} IntraW_{src,dst,m} = Dependency_{src,dst} - \sum_{m=1}^{|\mathcal{M}|} InterW_{src,dst,m} \tag{4.13}$$

$$IntraW_{src,dst,m} \geq \sum_{p=1}^{\mathcal{P}} X_{p,src,m} \tag{4.14}$$

We start the explanation of the ILP model with the constraints. Eq. 4.6 restricts each tasks' assignment to one single processor and one single performance mode. Using this information captured by the $X_{p,k,m}$ variables, we can calculate in Eq.4.3 the energy consumed by each task τ_k mapped on processor p_p and running at freqency f_m. For the clarity of mathematical expressions, the total power consumed by task τ_k if executed in mode m, $P_{k,m}$, is used. $P_{k,m}$ is calculated as the sum of the dynamic power and leakage power, Eq.4.2.

Eq. 4.7, 4.9, 4.8, 4.10, 4.11, 4.12, 4.13, 4.14 set the variables that capture the communication. For each pair of communicating tasks (τ_{src},τ_{dst}) mapped on different processors, the variables $InterR_{src,dst,m}$ and $InterW_{src,dst,m}$ corresponding to the frequency of the source (for the write operation) or destination (for the read operation) task are forced to 1. For the other frequencies of the same task pair, for

tasks that are mapped on the same processor and for tasks that do not communicate, the variables $InterR_{src,dst,m}$ and $InterW_{src,dst,m}$ are set to 0 (Eq.4.7,4.9, 4.8, 4.10).

Tasks (τ_{src}, τ_{dst}) that communicate ($Dependency_{src,dst} = 1$) but are mapped on the same processor ($InterW_{src,dst,m} = InterR_{src,dst,m} = 0$) will have $IntraW_{src,dst,m} = IntraR_{src,dst,m} = 1$.

Consequently, when all the variables $InterR_{src,dst,m}$, $InterW_{src,dst,m}$, $IntraR_{src,dst,m}$, $IntraW_{src,dst,m}$ are set, the energy for inter-processor (Eq.4.4) and intra-processor (Eq. 4.5) communications can be computed.

The energy consumption of a certain mapping is evaluated through the objective function from Eq.4.1. Eq.4.1 captures the energy consumed by the tasks, inter-processor and intra-processor communications, as well as the energy consumed due to the mode switching overhead E_{oh}. At this point it is interesting to note that the energy of the tasks and communications depends only on master problem variables. However, the value of the switching overheads can be computed only during scheduling. They are constrained by Benders Cuts, after the first iteration.

Several improvements can be introduced in the master problem model. In particular, the optimization time is improved if the symmetries leading the solver to explore the same configurations several times are removed. Consequently, a constraint imposing that each task t_i should be allocated on a processor p_j only if $i \leq j$ can be added. Moreover, in order to prevent initial mappings with tasks that are potentially running with low frequencies on the same processor (thus avoiding communication), the load on each processor is constrained by a safe upper bound that does not prevent the algorithm from finding the optimal solution.

4.4.2 The Sub-Problem model

The scheduling phase is modeled using Constraint Programming (CP). Each task τ_i has associated a variable $Start_i$ representing the starting time. The task execution time $t_{exe_i} = \frac{WNC_i}{f_i}$ was decided during mapping (where both the processor where the task is mapped and the frequency were calculated). The amount of time required by each task to read/write from/to the scratchpad is added to the task's worst-case execution time. If two communicating tasks τ_{src} and τ_{dst} are mapped on different processors, two additional activities (one for writing data on the shared memory and one for reading data from the shared memory) are introduced. We model the starting time of these activities with variables $StartWrite_{src,dst}$ and $StartRead_{src,dst}$. Reading and writing from/to the shared memory are performed at the same frequency as the corresponding task. If τ_{src} writes and τ_{dst} reads data , the writing activity is performed at the same frequency of τ_{src} and its duration $dWrite_{src,dst}$ depends on the frequency and on the amount of data τ_{src} writes, i.e., $WNC_{Wsrc,dst}/f_{src}$. Analogously, the reading activity is performed at the same frequency of τ_{dst} and

its duration $dRead_{src,dst}$ depends on the frequency and on the amount of data τ_{dst} reads, i.e., $WNC_{R_{src,dst}}/f_{dst}$. Clearly the read and write activities are linked to the corresponding task:

$$Start_{src} + duration_{src} \leq StartWrite_{src,dst} \quad \forall(\tau_{src}, \tau_{dst})$$

$$StartRead_{src,dst} + dRead_{src,dst} \leq Start_{dst} \quad \forall(\tau_{src}, \tau_{dst})$$

In the subproblem, we model precedence constraints in the following way: if tasks τ_i precedes task τ_j and they run on the same processor at the same frequency the precedence constraint is simply:

$$Start_i + Duration_i \leq Start_j$$

If instead the two tasks run on the same processor using different frequencies, we should add the time T_i for switching between the two frequencies.

$$Start_i + Duration_i + T_{oh_i} \leq Start_j$$

If the two tasks that are mapped on different processors communicate we add the time for the communication.

$$Start_i + Duration_i + dWrite_{ij} + dRead_{ij} \leq Start_j$$

Tasks can be executed on each processor sequentially, one task at a time. Of course, several tasks can be executed in parallel, each one on a different processor, according to the given precedence constraints (dependencies in the task graph). This is modeled in CP using the *cumulative* operator. This operator acts on each processor, modeled as a resource, restricting the number of tasks that can be executed at any given time on it to 1:

$$cumulative(TaskList_p, DurationList_p, [1], 1) \quad \forall \text{ processors } p_p$$

The parameters of the *cumulative* operator are the list of tasks $TaskList_p$ mapped on processor p_p, the list of their execution times $DurationList_p$, their resource consumption (which is a list of 1), and the capacity of the processor (1). In other words, the processor is modeled as a unary resource.

Capturing the bus performance is not easy on hardware platforms similar to one used in this chapter. The difficulty comes from the various types of traffic existing on the bus. First, the communication messages exchanged between tasks mapped on different processors are transferred via the bus. Traditionally [EDPP00, ASE+04b, PPE+06, SIE06], these are modeled as individual activities

that are scheduled on the bus. Second, for each task, code and the data are stored in a private memory, associated to the processor where the task is mapped. During the execution of the tasks, the corresponding instructions and data are fetched from the private memory to the processor via the bus. Even if caches are used, this results in a certain amount of traffic on the bus. However, modeling such individual memory accesses would result in a large number of optimization variables. Furthermore, the size of the memory necessary for storing the schedule for the bus makes such an approach impossible. We will address these issues in Chapter 6. In this chapter, however, the bus is captured through an additive model that was presented in [BGM+06]. An activity is associated to a write or read operation to/from the shared memory, performed by a pair of two communicating tasks mapped on different processors. The additive model measures the efficiency of the bus as its ability to provide the bandwidth required by read or write activities. Assuming that congestion effects can be neglected, the bus must be able to provide a bandwidth equal to the sum of the simultaneous communications. In [BGM+06], this model was demonstrated experimentally to be correct as long as the bus load is below 60%.

The objective function that has to be minimized in the scheduling problem is the mode switching overhead energy.

4.5 Genetic-Based Optimization Heuristic

This section introduces a genetic-based approach that performs task mapping and scheduling, using voltage scaling inside the inner energy optimization loop. The approach is described in detail in [AEP+07b, SAHE04]. The optimization flow, illustrated in Fig.4.3, is split into three parts:

- Genetic task mapping optimization
- Genetic schedule optimization
- Optimal voltage selection

In the genetic task mapping approach, solution candidates (potential mappings) are encoded into mapping strings, as shown in Fig. 4.4. Each gene in these strings describes a candidate mapping of a task to a processor. For instance, task τ_4 in Fig. 4.4 is mapped to CPU0. As typical in all genetic algorithms, ranking, selection, crossover, mutation and offspring insertion are applied in order to evolve an initial solution pool [Gol89, BTT98, ETZ00, SAHE04]. The key feature of this algorithm, is the invocation of the genetic list scheduling for each mapping candidate, in order to calculate the fitness function that guides the optimization.

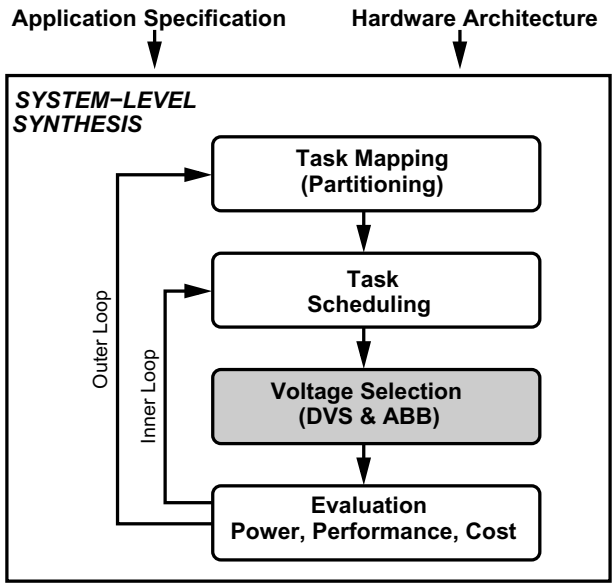


Figure 4.3: Genetic Optimization Flow

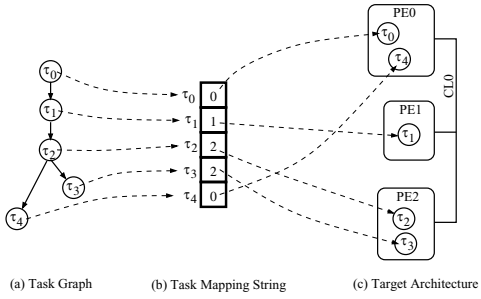


Figure 4.4: Task mapping string describing the mapping of five tasks to an architecture

The genetic scheduling algorithm finds for a given mapping, an energy efficient schedule that respects all the task deadlines. One of the most widely used heuristics approaches is list scheduling (LS). List scheduling algorithms take scheduling decisions based on task priorities [CG72]. They maintain, for each processor, a ready-list that contains the tasks that are ready to be scheduled. A task is consid-

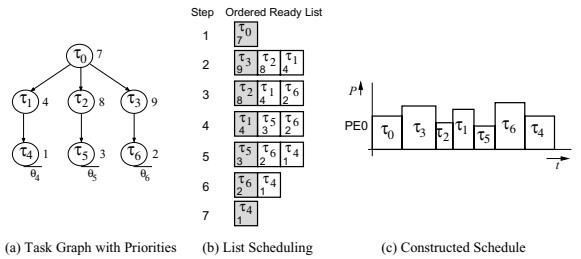


Figure 4.5: List scheduling

ered to be ready, if all its predecessors (given by the task graph) have finished their execution. The static schedule is constructed by scheduling the ready task with the highest priority as soon as the eligible processor becomes available. Thereby, the assignment of priorities defines the task execution order.

The basic idea behind list scheduling is shown in Fig. 4.5, which outlines the construction of a schedule for a single processor system. Consider the task graph with annotated priorities from Fig. 4.5(a). In the initial scheduling step, all tasks with no incoming edges are placed into a ready list, as shown in Fig. 4.5(b), Step 1. For this particular example, in the first step task τ_0 is added to the ready list. Being the only task in the ready list, task τ_0 is scheduled. After its execution has finished, the tasks τ_1, τ_2, and τ_3 become eligible for scheduling (due to their data dependency on τ_0); hence, they are placed into the ready list in decreasing order of their priorities (Scheduling Step 2). At this point τ_3 represents the ready task with the highest priority (9), so it is scheduled in Step 2. Having scheduled task τ_3, task τ_5 becomes ready and thus it is placed into the ready list, according to its priority. This scheduling procedure is repeated until no tasks are left in the ready list. Since each scheduling step schedules one task, seven iterations are necessary. The final schedule is shown in Fig. 4.5(c).

Clearly, different assignments of priorities result in different schedules. The task priorities are encoded into a priority string. The genetic algorithm aims to find an assignment of priorities that leads to a schedule of high quality in terms of timing behavior and exploitable slack time. Both crossover and mutation are applied during the iterative execution of the genetic algorithm. The algorithm terminates after a stop criterion is fulfilled (for example, a bound of the number of consecutive generations that did not improve significantly the solution).

A fitness function is used for evaluating the quality of a schedule. The fitness function captures the energy of a certain schedule. After the list scheduling has constructed a schedule for a given set of priorities, the algorithm proceeds by pass-

ing this schedule to a voltage selection algorithm that identifies the task voltages that minimize the energy dissipation. A penalty is applied for schedules that are not feasible.

As we have seen, the voltage selection is the core of the global energy optimization. During the genetic scheduling step, the voltage scaling heuristic presented in Chapter 3 is used. Once the genetic algorithms are finished, the optimal discrete voltage scaling algorithm presented in Chapter 3, restricted to select one single mode for each task, is performed.

4.6 Experimental Results

We conclude this chapter by presenting some experimental results. The hardware architecture considered during these experiments is described in Section 4.2. The frequency on each processor can be scaled dynamically, within a set of 4 performance modes: m_1=($f_1 = 200MHz, V_{dd} = 2.2V$), m_2=($f_2 = 100MHz, V_{dd} = 1.6V$), m_3=($f_3 = 66MHz, V_{dd} = 1.4$), m_4=($f_4 = 50MHz, V_{dd} = 1.3V$). For this experiment we assume that the power consumed by all tasks executing in a certain mode is equal. Consequently, the corresponding power consumptions are: $P_1 = 10.070mW$, $P_2 = 1.71mW$, $P_3 = 1.010mW$, $P_4 = 0.76mW$. Every frequency switching has an energy and a time penalty. The energy and time penalties for switching from mode i to mode j are given in the follwing matrices:

$$E_{oh_{i,j}} = \begin{pmatrix} 0 & 504 & 504 & 504 \\ 171 & 0 & 171 & 171 \\ 153 & 153 & 0 & 153 \\ 152 & 152 & 152 & 0 \end{pmatrix}$$

$$t_{oh_{i,j}} = \begin{pmatrix} 0 & 205 & 305 & 400 \\ 300 & 0 & 105 & 200 \\ 303 & 104 & 0 & 80 \\ 404 & 202 & 50 & 0 \end{pmatrix}$$

The first experiment is aimed to perform the energy optimization of a GSM encoder. The software application was partitioned into 6 tasks. Using the MPARM cycle accurate simulator [BBB+03, LPB04, And06, BGM+06, KBP+06], for each task the number of clock cycles required for execution on the processors and the number of clock cycles required for the communications were extracted. Both the optimal approach and the genetic algorithm have been used for the optimization. The results obtained by the two methods were identical. Table 4.1 shows these results. The optimization was performed considering several deadlines for the GSM

Deadline (ns)	Number of processors	# task allocated on core	Task frequency divider	Energy (nJ)
6000	1	1,1,1,1,1,1	3,3,3,3,3,3	5840
5500	2	2,1,1,1,1,1	3,3,3,3,3,3	5910
5000	2	1,1,1,1,1,2	3,3,3,3,3,3	5938
4500	2	1,1,1,1,2,2	3,3,3,3,3,3	5938
4000	2	1,1,1,2,2,2	3,3,3,3,3,3	5938
3500	2	1,1,1,2,2,2	3,3,3,3,3,3	5938
3000	3	1,2,2,3,3,3	3,3,3,3,3,3	6008
2500	3	1,1,2,2,3,3	3,3,3,3,3,3	6039
2000	4	1,2,3,3,4,4	3,3,3,3,3,3	6109
1500	6	1,2,3,4,5,6	3,3,3,3,3,3	6304
1000	6	1,2,3,4,5,6	3,2,2,2,3,2	6807
900	6	1,2,3,4,5,6	3,1,2,2,2,2	9834
750	6	1,2,3,4,5,6	2,1,2,2,2,2	9934
730	6	1,2,3,4,5,6	2,1,1,2,2,2	12102
710	6	1,2,3,4,5,6	2,1,1,1,2,2	14193

Table 4.1: Optimization results for the GSM encoder

encoder, starting from a loose deadline of 6000 ns down to a tight one of 710 ns. Table 4.1 shows in the second column the number of processors determined by both algorithms. The third column shows for each of the 6 tasks, the id of the processor where the task was mapped. The fourth column shows for each task the frequency divider. The actual frequency can by calculated by dividing the maximum frequency of 200MHz to the divider. The achieved energy is reported in the last column. If we examine the table, we notice the relation between the deadline and the energy consumption. As expected, if the deadline is large, the resulting energy is small. When the deadline decreases, the energy increases.

Another experiment was performed in order to compare the energies achieved by the two approaches presented in this chapter. A task graph consisting of 11 tasks was used for this purpose. Several instances of this task graph, with different deadlines were used. For each deadline, both optimization approaches were used. The parameters used for the genetic mapping algorithm are the following:
1) The genetic mappings stops after trying 10000 consecutive mappings that did not result in any energy improvement
2) The mutation probability is 0.29 while the crossover probability is 0.71.
3) The population size is 100.

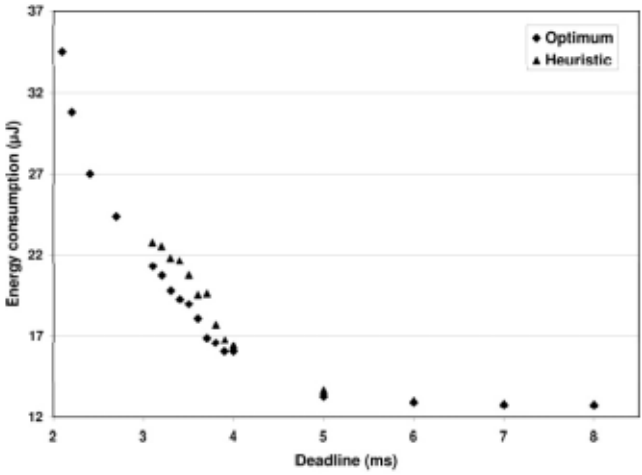


Figure 4.6: Optimal vs. Genetic-based Optimization

The parameters used for the genetic scheduling algorithm are the following:
1) The genetic scheduling stops after trying 100 consecutive schedules that did not result in any energy improvement
2) The mutation probability is 0.10 while the crossover probability is 0.90.
3) The population size is 50.

The results are presented in Fig.4.6. It can be noticed that for loose deadlines (5ms-8ms), both methods lead to the optimal energy or are very close. When the deadlines are very tight (less than 2.7ms), the genetic algorithm is unable to find allocations and schedules that meet the timing constraints. For deadlines between 3.9ms and 3.1ms, the optimal approach provides solutions that are around 7% better than the genetic algorithm. A notable exception is for a deadline of 3.7ms when the genetic algorithm produced a solution that is 25% worse then the optimal one.

In order to further validate the optimization flow in general and voltage selection model in particular, the MPARM virtual simulation platform [BBB$^+$03, LPB04] was deployed to implement the mappings, schedules and voltage/frequency assignments calculated using the optimal approach, for 200 problem instances. For each problem instance the energy predicted by the optimal algorithm and the energy reported by the simulation platform were compared. Fig.4.7 shows the distribution of energy deviations. The average difference between measured and predicted energy values is 2.9%, with a standard deviation of 1.72. This demonstrates the real-world applicability of such system level energy optimization frameworks.

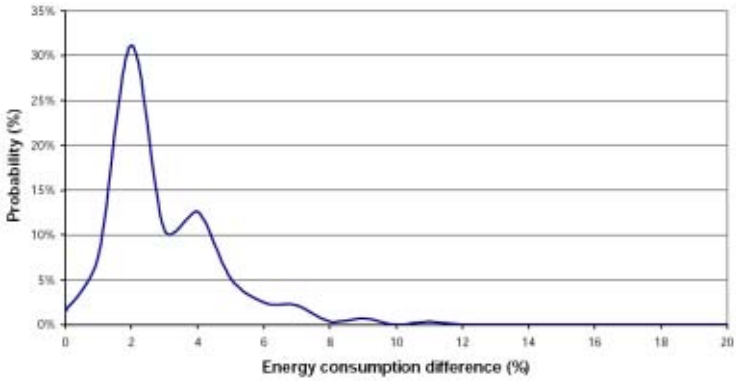


Figure 4.7: Energy Deviation

Chapter 5

Quasi-Static Voltage Selection

Supply voltage scaling and adaptive body-biasing are important techniques that help to reduce the energy dissipation of embedded systems. This is achieved by dynamically adjusting the voltage and performance settings according to the application needs. In order to take full advantage of slack that arises from variations in the execution time, it is important to recalculate the voltage (performance) settings during run-time, i.e., online. However, optimal voltage scaling algorithms are computationally expensive, and thus, if used online, significantly hamper the possible energy savings. To overcome the online complexity, we propose a quasi-static voltage scaling scheme, with a constant online time complexity $O(1)$. This allows to increase the exploitable slack as well as to avoid the energy dissipated due to online recalculation of the voltage settings.

5.1 Introduction and Related Work

Offline techniques calculate all voltage settings at compile time (before the actual execution), i.e., the voltage settings for each task in the system are not changed at run-time. On the other hand, online techniques recompute the voltage settings during run-time. Both approaches have their advantages and disadvantages. Offline voltage selection approaches avoid the computational overhead in terms of time and energy associated with the calculation of the voltage settings. However, to guarantee the fulfillment of deadline constraints, worst-case execution times (WCET) have to be considered during the voltage calculation. In reality, nevertheless, the actual execution time of the tasks, for most of their activations, is shorter than their WCET, with variations of up to 10 times [RE97]. Thus, an offline op-

timization based on the worst case is too pessimistic and hampers the achievable energy savings. In order to take advantage of the dynamic slack that arises from variations in the execution times, it is useful to dynamically recalculate the voltage settings during application run-time, i.e., *online*.

Dynamic approaches, however, suffer from the significant overhead in terms of execution time and power consumption caused by the online voltage calculation. As we will show, this overhead is intolerably large even if low complexity ($O(n)$) online heuristics are used instead of higher complexity optimal algorithms. Unfortunately, researchers have neglected this overhead when reporting high quality results obtained with dynamic approaches [YDS95, IHS98, AMMMA01, ZM04].

[IHS98] developed an online preemptive scheduling algorithm for sporadic and periodic tasks. The authors propose a linear complexity voltage scaling heuristic which uniformly distributes the available slack. An acceptance test is performed online, whenever a new sporadic task arrives. If the task can be executed without deadline violations, a new set of voltages for the ready tasks is computed.

In [AMMMA01], a power-aware hard real-time scheduling algorithm that considers the possibility of early completion of tasks is proposed. The proposed solution consists of three parts: (1) an off-line part where optimal voltages are computed based on the WCET, (2) an online part where slack from earlier finished tasks is redistributed to the remaining tasks, and (3) an online speculative speed adjustment to anticipate early completions of future executions. Assuming that tasks can possibly finish before their worst case execution time, an aggressive scaling policy is proposed. Tasks are run at a lower speed than the one computed assuming the worst case, as long as deadlines can still be met by speeding up the next tasks in case the effective execution time was higher than expected. As the authors do not assume any knowledge of the expected execution time, they experiment several levels of aggressiveness.

[ZM04, ZM05] introduced a feedback EDF scheduling algorithm with dynamic voltage scaling for hard real-time systems with dynamic workloads. Each task is divided in two parts, representing: (a) the expected execution time, and (b) the difference between the worst case and the expected execution time. A PID feedback controller selects the voltage for the first portion and guarantees hard deadline satisfaction for the overall task. The second part is always executed with the highest speed, while for the first part dynamic voltage scaling is used. Online, each time a task finishes, the feedback controller adapts the expected execution time for the future instances of that task. A linear complexity voltage scaling heuristic is employed for the computation of the new voltages. On a system with dynamic workloads, their approach yields higher energy savings then an off-line dynamic voltage scaling schedule.

The techniques presented in [Gru01, ZLL+05, LS04, LZS+06] use a stochastic approach to minimize the average-case energy consumption in hard real-time systems. The execution pattern is given as a probability distribution, reflecting the chance that a task execution can finish after a certain number of clock cycles. [Gru01, LS04, LZS+06] propose solutions that can be applied to single task systems. [ZLL+05] extends the problem formulation to multiple tasks, but assumes that continuous voltages are available on the processors.

Despite their potential to achieve energy reductions, all above mentioned online approaches greatly neglect the computational overhead required for the voltage scaling. We will come back at this important aspect in section 5.2.1.

In [YC03] an approach is outlined in which the online scheduler is executed at each activation of the application. The decision taken by the scheduler is based on a set of precalculated supply voltage settings. The approach assumes that at each activation it is known in advance which subgraphs of the whole application graph will be executed. For each such subgraph worst case execution times are assumed and, thus, no dynamic slack can be exploited.

Noticeable exceptions from this broad off-line/online classification are the intra-task voltage selection approaches presented in [SKL01, SKC04, SKD05, SKL06]. The basic idea of these approaches is to perform an off-line execution path analysis, and to calculate for each of the possible paths the voltage settings in advance. The resulting voltage settings are stored within the application program. During run-time the voltage settings along the activated path are selected. The main advantage of these approaches is the fact that online overheads associated with the voltage calculation are avoided and the execution time variation among different execution paths can be exploited. Despite their energy efficiency these approaches are most suitable for single task systems, since the number of execution paths $p = z^n$ in multi-task applications grows exponentially with the number of tasks n and depends also on the number of execution paths in a single task z. Therefore, the off-line optimization times required for the voltage calculation can become intractable for realistic systems with an increased number of tasks and execution paths. It is also important to note that the approaches described in [SKL01, SKC04, SKD05, SKL06] are restricted to take advantage of slack that arises from different execution times along different execution paths, while assuming a worst-case execution time for each atomic instruction in order to guarantee the satisfaction of deadline constraints. In reality, however, execution time variations are also caused by pipeline stalls, cache hit/miss rate, and different cycles required for the same instruction — all of which are input data dependent.

In this chapter we propose a quasi-static voltage scaling technique for energy minimization of multi-task real-time systems. This technique is able to exploit the dynamic slack and, at the same time, keeps the online overhead (required to

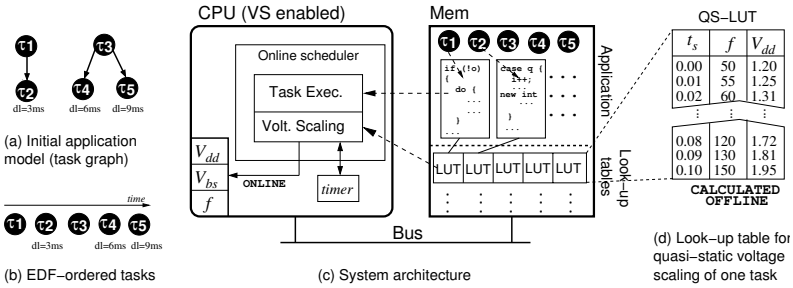


Figure 5.1: System architecture

readjust the voltage settings at run-time) extremely low. The obtained performance is superior to any of the previously proposed dynamic approaches. Henceforth, we will refer to the proposed voltage scaling technique as quasi-static voltage scaling (QSVS).

The chapter is organized as follows: Preliminaries and motivations are given in Section 5.2, as well as the key ideas behind the presented work. An exact problem formulation for quasi-static voltage scaling is given in Section 5.3. Our algorithms to solve this problem are described in Sections 5.4, 5.5, 5.6. Extensive experimental results, including a real-life example, are presented in Section 5.9.

5.2 Application and Architecture Model

In this work, we consider applications that are modeled as task graphs, i.e., several tasks with possible data dependencies among them, as in Fig. 5.1(a). Each task is characterized by several parameters (see also section 5.3), such as a deadline, the effectively switched capacitance, and the number of clock cycles required in the best-case (BNC), expected-case (ENC), and worst-case (WNC). Once activated, tasks are running without being preempted until their completion. The tasks are executed on an embedded architecture that consists of a voltage-scalable processor (scalable in terms of *supply* and *body-bias* voltage). The power and delay model of the processor is described in section 3.3. The processor is connected to a memory that stores the application and a set of look-up tables (LUT), one for each task, required for QSVS. This architectural setup is shown in Fig. 5.1(c). During execution, the scheduler has to adjust the processor's performance to the appropriate level via voltage scaling, i.e., the scheduler writes the settings for the operational frequency f, the supply voltage V_{dd}, and the body-bias voltage V_{bs} into special processor registers before the task execution starts. An appropriate performance level

allows the tasks to meet their deadlines while maximizing the energy savings. In order to exploit slack that arises from variations in the execution time of tasks, it is unavoidable to dynamically re-calculate the performance levels. Nevertheless, calculating appropriate voltage levels (the means by which performance levels are calculated) is a computationally expensive task, i.e., it requires precious CPU time, which, if avoided, would allow to lower the CPU performance and, consequently, the energy consumption.

The approach presented in this chapter aims to reduce this online overhead by performing the necessary voltage selection computations offline (at compile time) and storing a limited amount of information as look-up tables (LUTs) within memory. This information is then used during application run-time (i.e., online) to calculate the voltage and performance settings extremely fast (constant time $O(1)$), see Fig. 5.1(d).

5.2.1 Motivation

This section motivates the proposed quasi-static voltage scaling technique and outlines its basic idea.

Online Overhead Evaluation

As we have mentioned earlier, to fully take advantage of variations in the execution time of tasks, with the aim to reduce the energy dissipation, it is unavoidable to recompute the voltage settings online according to the actual task execution times. This is illustrated in Fig. 5.2, where we consider an application consisting of $n = 4$ tasks. The voltage level pairs (V_{dd}, V_{bs}) used for each task are also included in the figure. Only after task τ_1 has terminated, we know its actual finishing time and, accordingly, the amount of dynamic slack that can be distributed to the remaining tasks (τ_2,τ_3,τ_4). Ideally, in order to optimally distribute the slack among these tasks (τ_2,τ_3, and τ_4), it is necessary to run a voltage scaling algorithm (in Fig. 5.2 indicated as VS1) before starting the execution of task τ_2. A straightforward implementation of an ideal online voltage scaling algorithm is to perform a "complete" recalculation of the voltage settings each time a task finishes, using for example the approaches described in [SAH01, YLJ03]. However, such an implementation would be only feasible if the computational overhead associated with the voltage scaling algorithm is very low, which is not the case in practice. The computational complexity of such optimal voltage scaling algorithms for monoprocessor systems is $O(m \cdot n)$ [SAH01, YLJ03] (with m specifying the accuracy-a usual value of 100 and n being the number of tasks). That is, a substantial amount of CPU cycles are spent calculating the voltage/frequency settings each time a task finishes—during

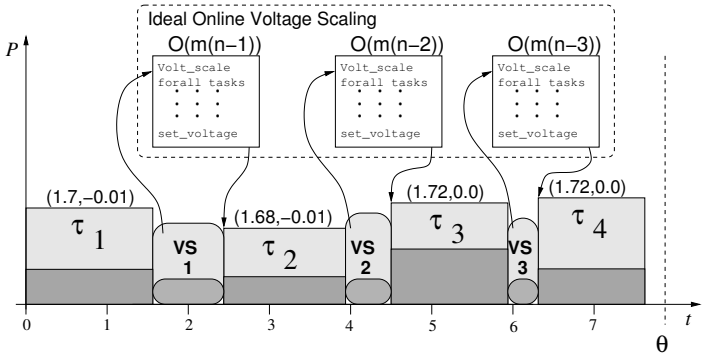


Figure 5.2: Ideal online voltage scaling approach

these cycles the CPU uses precious energy and reduces the amount of exploitable slack.

To get insight into the computational requirements of voltage scaling algorithms and how this overhead compares to the amount of computations performed by actual applications, we have simulated and profiled several applications and voltage scaling techniques, using two cycle accurate simulators: StrongARM (SA-1100) [QM03] and PowerPC(MPC750)[Gro, PMT04]. We have also performed measurements on actual implementations using an AMD platform (AMD Athlon 2400XP). Tab. 5.1 shows these results for two applications that can be commonly found in hand-held devices: a GSM voice codec and an MPEG video encoder. Results are shown for AMD, SA-1100 and MPC750 and are given in terms of best-

Bench-mark type	AMD Athlon			SA1100			MPC750		
	BNC (k)	WNC (k)	Var. (%)	BNC (k)	WNC (k)	Var. (%)	BNC (k)	WNC (k)	Var. (%)
GSM	140	155	10	367	394	7	159	181	13
MPEG	731	1,700	43	4,458	8,043	45	3,869	6,439	40

Table 5.1: Simulation results of different applications

case (BNC) and worst-case number (WNC) of thousands of clock cycles needed for the execution of one period of the considered applications (20 ms for the GSM codec and 40 ms for the MPEG encoder). [1] For instance, on the SA-1100 processor one iteration of the MPEG encoder requires in the best-case 4.458 kcycles and in the worst-case 8.043 kcycles, that is a variation of 45%. Similarly, Tab. 5.2

[1] Note that the numbers for BNC and WNC are lower and upper bounds observed during the profiling. They have not been analytically derived.

Voltage scaling algorithm	AMD NC (k)	SA-1100 NC (k)	MPC750 NC (k)
OptimalVS(Vdd+Vbs, 20 tasks) [YLJ03]	8,410	1,232,552	136,950
OptimalVS(Vdd, 20 tasks) [YLJ03]	210	32,320	3,513
MTS Heuristic(Vdd, 20 tasks) [Gru02]	8	84	12
MTS Heuristic(Vdd+Vbs, 20 tasks)	40	623	73
Greedy Heuristic(Vdd) [AMMMA01]	0.9	10	1.0
Greedy Heuristic(Vdd+Vbs)	4.9	34	3.8
Quasi-Static(Vdd+Vbs) (proposed)	0.9	1.0	1.0

Table 5.2: Simulation results: Voltage scaling algorithms

presents the simulation outcomes for different voltage scaling algorithms. As an example, performing one single time the optimal online voltage scaling using the algorithm from [YLJ03] for 20 remaining tasks (just like VS1 is performed for the three remaining tasks τ_2, τ_3, and τ_4 in Fig. 5.2) requires 8,410 kcycles on the AMD processor, 136,950 kcycles on the MPC750 processor, while on SA-1100 it requires even 1,232,552 kcycles. Using the same algorithm for V_{dd}-only scaling (no V_{bs} scaling), needs 210 kcycles on the AMD processor, 32,320 kcycles on the SA-1100 and 3,513 kcycles on the MPC750. The difference in complexity between supply voltage scaling and combined supply and body bias scaling comes from the fact that in the case of V_{dd}-only, for a given frequency there exists one corresponding supply voltage, as opposed to a potentially infinite number of (V_{dd}, V_{bs}) pairs in the other case. Given a certain frequency, an optimization is needed to compute the (V_{dd}, V_{bs}) pair that minimizes the energy. Comparing the results in Tables 5.1 and 5.2 indicates that voltage scaling often surpasses the complexity of the applications itself. For instance, performing a "simple" V_{dd}-only scaling requires more CPU time (on AMD 210k cycles) than decoding a single voice frame using the GSM codec (on AMD 155k cycles). Clearly, such overheads seriously affect the possible energy savings, or even outdo the energy consumed by the application.

Several suboptimal heuristics with lower complexities have been proposed for online computation of the supply voltage. Gruian [Gru02] has proposed a linear time heuristic, while the approaches given in [AMMMA01, ZM04] use a greedy heuristic of constant time complexity. We report their performance in terms of the required number of cycles in Tab. 5.2, including also their additional adaptation for combined supply and body bias scaling. While these heuristics have a smaller online overhead than the optimal algorithms, their cost is still high, except for the greedy algorithm for supply voltage scaling [AMMMA01, ZM04]. However, even the cost of the greedy increases up to 5.4 times when it is used for supply and body bias scaling. The overhead of our proposed algorithm is given in the last line of Tab. 5.2.

Basic Idea: Quasi-Static Voltage Scaling

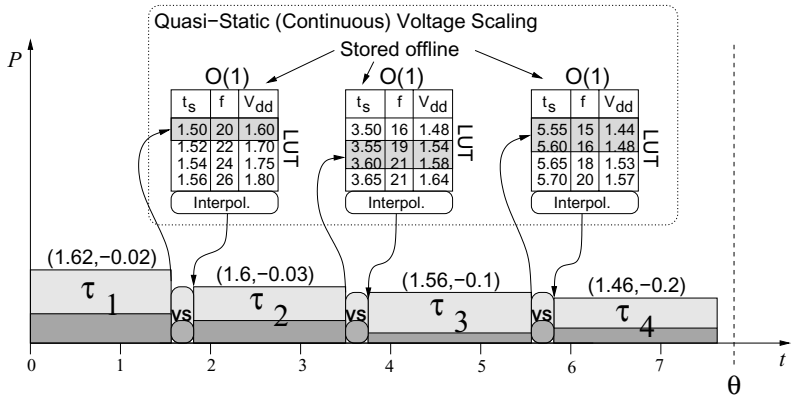
To overcome the voltage selection overhead problem, we propose a quasi-static voltage scaling technique. This approach is divided into two phases. In the first phase, which is performed before the actual execution (i.e., offline), voltage settings for all tasks are pre-computed based on possible task start times. The resulting voltage/frequency settings are stored in look-up tables (LUTs) that are specific to each task. It is important to note that this phase performs the time intensive optimization of the voltage settings.

The second phase is performed online and it is outlined in Fig. 5.3. Each time new voltage settings for a task need to be calculated, the online scheme looks up the voltage/frequency settings from the LUT based on the actual task start time. If there is no exact entry in the LUT that corresponds to the actual start time, then the voltage settings are estimated using a linear interpolation between the two entries that surround the actual start time. For instance, task τ_3 has an actual start time of 3.58ms. As indicated in Fig. 5.3, this start time is surrounded by the LUT entries 3.55ms and 3.60ms. In accordance, the frequency and voltage setting for task τ_3 are interpolated based on these entries. The main advantage of the online quasi-static voltage selection algorithm is its constant time complexity $O(1)$. As shown in the last line of Tab. 5.2, the LUT look-up and voltage interpolation requires only 900 CPU cycles each time new voltage settings have to be calculated. Please note that the complexity of the online quasi-static voltage selection is independent of the number of remaining tasks.

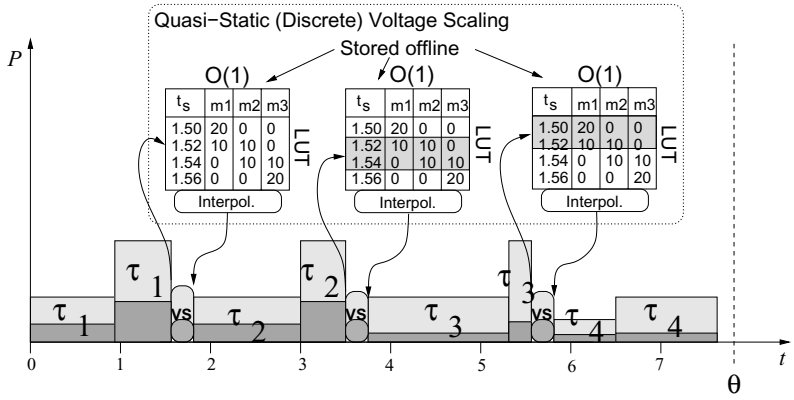
5.3 Problem Formulation

Consider a set of NT tasks, $\Pi = \{\tau_i\}$ such that the execution order is fixed according to a scheduling policy (e.g. EDF [ZHC02]). According to this order, task τ_i has to be executed after τ_{i-1} and before τ_{i+1}. The processor can vary its supply voltage V_{dd} and body-bias voltage V_{bs} and consequently its frequency f within certain continuous ranges (for the continuous optimization) or within a set of discrete modes $m_z = \{V_{dd_z}, V_{bs_z}, f_z\}$ (for the discrete optimization). The dynamic and leakage power dissipation as well as the operational frequency (cycle time) depend on the selected voltage pair (mode). Tasks are executed cycle by cycle and each cycle can be potentially executed at different voltage settings, i.e., a different energy/performance trade-off. Each task τ_i is characterized by a six-tuple,

$$\tau_i = < BNC_i, ENC_i, WNC_i, Ceff_i, dl_i >$$



(a) Optimization based on continouos voltage scaling



(b) Optimization based on discrete voltage scaling

Figure 5.3: Quasi-static voltage scaling based on pre-stored look-up tables

where BNC_i, ENC_i, and WNC_i denote the best-case, the expected-case, and the worst-case number of clock cycles, respectively, that task τ_i requires for its execution. BNC (WNC) is defined as the lowest (highest) number of cycles task τ_i needs for its execution, while ENC is the arithmetic mean value of the probability density function $p(WNC)$ of the task execution cycles WNC, i.e., $ENC = \sum_{j=1}^{WNC} j \cdot p_j(j)$. Further, $Ceff_i$ and dl_i represent the effectively charged capacitance and the deadline. The aim is to reduce the energy consumption by exploiting *dynamic slack* as well as *static slack*. Dynamic slack results from tasks that require less execution

cycles than in their worst case. Static slack is the result of idleness due to system over-performance, observable even when tasks execute with the worst-case number of cycles.

Our goal is to store a look-up table LUT_i for each task τ_i, such that the energy consumption during runtime is minimized. The size of the memory available for storing the look-up tables (and, implicitly the total number NL of table entries) is given as a constraint.

5.4 Offline Algorithm: Overall Approach

Quasi-static voltage scaling aims to reduce the online overhead required to compute voltage settings by splitting the voltage scaling process into two phases. That is, the voltage settings are prepared offline, and the stored voltage settings are used online to adjust the voltage/frequency in accordance to the actual task execution times.

The pseudo-code corresponding to the calculations performed offline is given in Fig. 5.4. The algorithm requires the following input information. The scheduled task set Π, defined in section 5.3. For the tasks $\tau_i \in \Pi$, the expected (ENC_i), the worst-case (WNC_i) and the best-case (BNC_i) number of cycles, the effectively switched capacitance (Ceff_i) and the deadline D_i. Furthermore, the total number of look-up table entries NL is given. The algorithm returns the quasi-static scaling table $\mathtt{LUT}_i$, for each task $\tau_i \in \Pi$. This table includes n_i ($\sum_{i=1}^{n} n_i = NL$) possible start times $t_{s_{i,j}}, j = 1..n_i$ for each task τ_i, and the corresponding optimal settings for the supply voltage `Vdd` and the operational frequency `f`.

Upon initialization, the algorithm computes the earliest and latest possible start times as well as the latest finishing time for each task (lines 01–09). The earliest start time $\mathtt{EST}_i$ is based on the situation in which all tasks would execute with their best-case number of clock cycles at the highest voltage settings, i.e., the shortest possible execution (lines 01–03). The latest start time $\mathtt{LST}_i$ is calculated as the latest start time of task τ_i that allows to satisfy the deadlines for all the tasks τ_j, $j \geq i$, executed with the worst-case number of clock cycles at the highest voltages (lines 04–06). Similarly, we compute the latest finishing time of each task (lines 07–09).

The algorithm proceeds by initializing the set of remaining tasks Π_r with the set of all tasks Π (line 10). In the following (lines 11–29), the voltage and frequency settings for the start time intervals of each task are calculated. More detailed, in line 12 and 13 the size of the interval [$\mathtt{EST}_i$, $\mathtt{LST}_i$] of possible start times is computed and the interval counter `j` is initialized. The number of entry points $\mathtt{n}_i$ that are stored for each task (i.e., the number of possible start times considered)

```
Algorithm:  QUASI_STATIC_VS_OFF-LINE
-------------------------------------------------------------
Input:  - execution order of tasks τ ∈ Π
        - for all tasks τ_i ∈ Π:
          BNC_i, ENC_i, WNC_i, Ceff_i, dl_i
        - NL
Output: - Look up tables LUT_i
-------------------------------------------------------------
01:  for i = 1 to NT {
02:    EST_i ← calc_earliest_start_time
03:  }//end for
04:  for i = NT downto 1 {
05:    LST_i ← calc_latest_start_time
06:  }//end for
07:  for i = NT downto 1 {
08:    LFT_i ← calc_latest_finishing_time
09:  }//end for
10:  Π_r ← Π
11:  for all τ_i ∈ Π { //ordered i=1..NT
12:    I_i ← LIST_i − EIST_i
13:    j ← 0
14:    n_i ← comp_interpolation_points(τ_i, LST_i, EST_i)
15:    for (t_s ← EST_i; t_s ≤ LST_i; t_s ← t_s + I_i/n) {
16:      t_si ← t_s
17:  #if CONT_VS
18:      (Vdd_i, Vbs_i, f_i) ← cont_volt_scaling(Π_r, t_si) //ENC based
19:      LUT_i[j] ← store_QS_lookup(t_si, Vdd_i, f)
20:  #endif
21:  #if DISC_VS
22:      (t_endi, h_i) ← disc_volt_scaling(Π_r, t_si) //ENC based
23:      LUT_i[j] ← store_QS_lookup(t_si, t_endi, h)
24:      compute_compat_mode_pairs();
25:  #endif
26:      j ← j + 1
27:    }//end for
28:    Π_r ← Π_r − τ_i
29:  }//end for all
30:  for all τ_i ∈ Π return LUT_i
```

Figure 5.4: Pseudocode: Quasi-Static Offline Algorithm

is calculated in line 14. This will be further discussed in Section 5.7. For all $\mathtt{n}_i$ possible start times t_s in the start time interval of task τ_i (line 15), the task start time t_{s_i} is set to the possible start time (line 16) and the corresponding optimal voltage and frequency settings of τ_i are computed and stored in the LUT (lines 15-27). For this computation, we use the algorithms presented in Chapter 3, modified to incorporate the optimization for the expected case. Instead of optimizing the energy consumption for the worst-case number of clock cycles, we calculate the voltage levels such that the energy consumption is optimal in the case the tasks execute their expected-case (which, in reality, happens with a higher probability). However, since our approach targets hard real-time systems, we have to guarantee the satisfaction of all deadlines even if tasks execute their worst-case number of clock cycles. In accordance with the problem formulation from section 5.3, the quasi-static algorithm performs the energy optimization and calculates the LUT using continuous (lines 17–20) or discrete voltage scaling (lines 21–25). We will explain both approaches in the following sections, together with their particular online algorithms. The results of the (continuous or discrete) voltage scaling for the current task τ_i, given the start time t_{s_i}, are stored in the LUT. The for-loop (line 15–27) is repeated for all $\mathtt{n}_i$ possible start times of task τ_i. The algorithm returns the quasi-static scaling table $\mathtt{LUT}_i$ for all tasks $\tau_i \in \Pi$.

5.5 Voltage Scaling with Continuous Voltage Levels

5.5.1 Offline Algorithm

In this section, we will present the continuous voltage scaling algorithm used in Fig. 5.4, line 18. The problem can be formulated as a convex nonlinear optimization as follows: Minimize

$$\sum_{k=i}^{|\Pi_r|} \Big(\underbrace{ENC_k \cdot C_{eff_k} \cdot V_{dd_k}^2}_{E_{dyn_k}} + \underbrace{L_g (K_3 \cdot V_{dd_k} \cdot e^{K_4 \cdot V_{dd_k}} \cdot e^{K_5 \cdot V_{bs_k}} + I_{Ju} \cdot |V_{bs_k}|) \cdot t_k}_{E_{leak_k}} \Big) \tag{5.1}$$

subject to:

$$s_i \geq t_{s_i} \tag{5.2}$$

$$t_k = \begin{cases} WNC_k \cdot \frac{(K_6 \cdot L_d \cdot V_{dd_k})}{((1+K_1) \cdot V_{dd_k} + K_2 \cdot V_{bs_k} - V_{th_1})^\alpha} & \text{if } k = i \\ ENC_k \cdot \frac{(K_6 \cdot L_d \cdot V_{dd_k})}{((1+K_1) \cdot V_{dd_k} + K_2 \cdot V_{bs_k} - V_{th_1})^\alpha} & \forall \tau_k \in \Pi_r \; k \neq i \end{cases} \tag{5.3}$$

$$s_k + t_k \leq s_{k+1} \quad \forall\, \tau_k, k = 1..(|\Pi_r| - 1) \tag{5.4}$$

$$s_k + t_k \leq dl_k \quad \forall\, \tau_k \in \Pi_r \text{ that have a deadline} \tag{5.5}$$

$$s_i + t_i \leq LFT_i \quad \tau_i \text{ is the first task in } \Pi_r \tag{5.6}$$

$$s_k \geq 0 \quad \forall\, \tau_k \in \Pi_r \tag{5.7}$$

$$V_{dd_{min}} \leq V_{dd_k} \leq V_{dd_{max}} \text{ and } V_{bs_{min}} \leq V_{dd_k} \leq V_{bs_{max}} \quad \forall\, \tau_k \in \Pi_r \tag{5.8}$$

The variables that need to be optimized in this formulation are the task execution times t_k, the task start times s_k as well as the voltages V_{dd_k} and V_{bs_k}. The start time of the current task has to match the start time assumed for the currently calculated LUT entry, Eq. 5.2. The whole formulation can be explained as follows. The total energy consumption, which is the combination of dynamic and leakage energy, has to be minimized. As we aim the energy optimization in the most likely case, the expected number of clock cycles ENC_k is used in the objective. The minimization has to comply to the following relations and constraints. The task execution time has to be equivalent to the number of clock cycles of the task multiplied by the circuit delay for a particular V_{dd_k} and V_{bs_k} setting, as expressed by Eq. 5.3. In order to guarantee that the current task τ_i ends before the deadline, its execution time t_i is calculated using the worst-case number of cycles WNC_i. Please remember from the computation of the latest finishing time (LFT), that if task τ_i finishes its execution before LFT_i, then the rest of the tasks are guaranteed to meet their deadlines even in the worst case. This condition is enforced by Eq. 5.6.

As opposed to the current task τ_i, for the remaining $\tau_k \in \Pi_r, k \neq i$, the expected number of clock cycles ENC_k is used when calculating their execution time in Eq. 5.3. This is important for a distribution of the slack that minimizes the energy consumption in the expected case. Please note that this is possible because after performing the voltage selection algorithm, only the results for the current task are stored in the LUT. The settings calculated for the rest of the tasks are discarded.

The rest of the nonlinear formulation is similar to the one presented in Chapter 3, section 3.6.1 for solving the continuous voltage selection without overheads. Eq. 5.4 expresses the task execution order, while deadlines are enforced in Eq. 5.5.

5.5.2 Online Algorithm

Having prepared, for all tasks of the system, a set of possible voltage and frequency settings depending on the task start time, we outline next how this information is used online to compute the voltage and frequency settings for the effective (i.e., actual) start time of a task. Fig. 5.5 gives the pseudocode of the online algorithm. This algorithm is called each time, after a task finishes its execution, in order to calculate the voltage settings for the next task τ_n. The input consists of the task

```
Algorithm:  QUASI_STATIC_VS_ONLINE_CONTINUOUS
-------------------------------------------------------------------
Input:  - start time t_sn of next task τ_n
        - Quasi-Static Scaling Table LUT_n
        - number of start time interval steps LUT_SIZE_n
        - latest finishing time LFT_n of task τ_n
Output: - frequency and voltage settings for task τ_n
-------------------------------------------------------------------
01:  (x,y) ← calc_st_interval(LUT_n,t_sn)
02:  f_n ← inter_freq(LUT_n,x,y,t_sn)
03:  Vdd_n ← inter_Vdd(LUT_n,x,y,t_sn)
04:  if t_sn + WNC_n/f_n > LFT_n {
05:     f_n ← f_y
06:     Vdd_n ← Vdd_y
07:  }
08:  Vbs_n ← calc_Vbs(f_n,Vdd_n)
09:  return (f_n,Vdd_n,Vbs_n)
```

Figure 5.5: Pseudocode: Continuous Online Algorithm

start time t_{s_n}, the quasi-static scaling table $\mathtt{LUT}_n$, and the number of interval steps $\mathtt{LUT\_SIZE}_n$. As output, the algorithm returns the frequency $\mathtt{f}_n$ and voltage settings $\mathtt{Vdd}_n$ and $\mathtt{Vbs}_n$ for the next task, τ_n. In the first step, the algorithm calculates the two entries x and y from the quasi-static scaling table $\mathtt{LUT}_n$ that contain the start times which surround the actual time t_{s_n} (line 01). According to the identified entries, the frequency setting $\mathtt{f}_n$ for the execution of task τ_n is linearly interpolated using the two frequency settings from the quasi-static scaling table $\mathtt{LUT}_n\mathtt{[x]}$ and $\mathtt{LUT}_n\mathtt{[y]}$ (line 02). Similarly, in step 03 the supply voltage $\mathtt{Vdd}_n$ is linearly interpolated from the two surrounding voltage entries in $\mathtt{LUT}_n$.

As will be shown in Appendix D, task frequency, considered as a function of the start time, is piecewise convex. This means that any frequency, calculated by linear interpolating two frequencies from the look-up table that are situated on a convex region, is safe. However, if the frequencies from $\mathtt{LUT}_n\mathtt{[x]}$ and $\mathtt{LUT}_n\mathtt{[y]}$ are not on a convex region, no guarantees regarding the resulting real-time behavior can be made. It is not possible to calculate the start times that bound the convex regions of each task frequency. The online algorithm handles this issue in line 04. If, assuming the task executes the worst-case number of clock cycles, uses

the interpolated frequency and exceeds its latest finishing time, the frequency and supply voltage are set to the ones from $\texttt{LUT}_n\texttt{[y]}$ (line 05–06). This guarantees the correct real-time execution, since the frequency from $\texttt{LUT}_n\texttt{[y]}$ was calculated assuming a start time higher than the actual one.

As mentioned in Section 5.4, we do not directly interpolate the setting for the body-bias voltage $\texttt{Vbs}_n$, due to the nonlinear relation between frequency, supply voltage, and body-bias voltage. That is, interpolating V_{dd} and V_{bs} at the same time, can result in an operational frequency that does not match the actually needed frequency—resulting in possible deadline violations. Therefore, we calculate the body-bias voltage directly from the interpolated frequency and supply voltage values, using Eq. 3.3 (line 08). The algorithm returns the settings for the frequency, supply and body-bias voltage (line 09). It is worthwhile to mention that all steps that are necessary to perform the online calculation are computed in constant time, i.e., the time complexity of the quasi-static online algorithm is $O(1)$.

The quality of this algorithm depends directly on the number of intermediate start times used. This aspect will be discussed in section 5.7. It is interesting to note that even though it cannot be formally demonstrated, the number of convex regions is reduced and, thus, in most cases the frequency and supply voltage are calculated using the linear interpolation. More details are given in Appendix D.

In section 5.5, we have presented the offline and online phases of the quasi-static algorithm, under the assumption that the processor is able to scale its frequency and voltages in continuous ranges. We outline in the next section the corresponding discrete algorithms.

5.6 Voltage Scaling Algorithm with Discrete Voltage Levels

We consider that processors can run in different modes $m \in \mathcal{M}$. Each mode m is characterized by a voltage pair (V_{dd_m}, V_{bs_m}) that determines the operational frequency f_m, the normalized dynamic power P_{dnom_m}, and the leakage power dissipation P_{leak_m}. The frequency and the leakage power are given by Eqs. 3.3 and 3.2, respectively. The normalized dynamic power is given by $P_{dnom_m} = f_m \cdot V_{dd_m}^2$. Accordingly, the dynamic power of a task τ_k operating in mode m is computed as $C_{eff_k} \cdot P_{dnom_m}$. Similar to the previous section, we discuss first the offline calculation of the LUTs, followed by the outline of the online algorithm.

5.6.1 Offline Algorithm

We present the voltage scaling algorithm used in Fig. 5.4, line 22. The problem is formulated using integer linear programming as follows:

Minimize

$$\sum_{k=1}^{|\Pi_r|} \sum_{m \in \mathcal{M}} \left(C_{eff_k} \cdot P_{dnom_m} \cdot t_{k,m} + P_{leak_m} \cdot t_{k,m} \right) \tag{5.9}$$

Subject to:

$$s_i \geq t_{s_i} \tag{5.10}$$

$$c_{k,m} = t_{k,m} \cdot f_m \quad \forall \tau_k \in \Pi_r, m \in \mathcal{M} \tag{5.11}$$

$$\sum_{m \in \mathcal{M}} c_{k,m} = \begin{cases} WNC_k & \text{if } k = i \\ ENC_k & \text{if } k \neq i \end{cases} \tag{5.12}$$

$$s_k + \sum_{m \in \mathcal{M}} t_{k,m} \leq dl_k \quad \forall\, \tau_k \in \Pi_r \text{ that have a deadline} \tag{5.13}$$

$$s_k + \sum_{m \in \mathcal{M}} t_{k,m} \leq s_{k+1} \quad \forall\, \tau_k, k = 1..(|\Pi_r| - 1) \tag{5.14}$$

$$s_i + \sum_{m \in \mathcal{M}} t_{i,m} \leq LFT_i \quad \tau_i \text{ is the current task} \tag{5.15}$$

$$s_k \geq 0 \ , \ t_{k,m} \geq 0 \ \text{ and } \ c_{k,m} \text{ is integer } \ \forall\, \tau_k \in \Pi_r \tag{5.16}$$

The task execution time $t_{k,m}$ and the number of clock cycles $c_{k,m}$ spent within a mode are the variables in the MILP formulation. The number of clock cycles has to be an integer and hence $c_{k,m}$ is restricted to the integer domain (Eq.5.16). The total energy consumption to be minimized, expressed by the objective in Eq. 5.9, is given by two sums. The inner sum indicates the energy dissipated by an individual task τ_k, depending on the time $t_{k,m}$ spent in each mode m. While the outer sum adds up the energy of all tasks. Similar to the continuous algorithm from section 5.5.1, the expected number of clock cycles is used for each task in the objective function.

The start time of the current task has to match the start time assumed for the currently calculated LUT entry, Eq. 5.10. The relation between execution time and number of clock cycles is expressed in Eq. 5.11. For similar reasons as in section 5.5.1, the worst-case number of clock cycles WNC_i is used for the current task τ_i. For the remaining tasks, the execution time is calculated based on the expected number of clock cycles ENC_k. In order to guarantee that the deadlines are met in the worst case, Eq. 5.15 forces task τ_i to complete in the worst case before its latest finishing time, LFT_i.

Similar to the continuous formulation from section 5.5.1, Eq. 5.13 and Eq. 5.14 are needed for distributing the slack according to the expected case.

As shown in Chapter 3, the discrete voltage scaling problem is NP hard. Thus, performing the exact calculation inside an optimization loop as in Fig. 5.4 is not feasible in practice. If the restriction of the number the clock cycles to the integer domain is relaxed, the problem can be solved efficiently in polynomial time using linear programming. The difference in energy between the optimal solution and the relaxed problem is below 1%. This is due to the fact that the number of clock cycles is large and thus the energy differences caused by rounding a clock cycle for each task are very small.

Using this linear programming formulation, we compute offline for each task τ_i, the number of clock cycles to be executed in each mode and the resulting end time, given several possible start times.

At this point it is interesting to make the following observations.

For each task, if the variables $c_{k,i}$ are not restricted to the integer domain, after performing the optimal voltage selection computation, the resulting number of clock cycles assigned to a task is different of zero for at most two of the modes. The demonstration is given in Appendix E. This property will be used by the online algorithm outlined in the next section.

Moreover, for each task, a table of so called *compatible_modes* can be derived offline. Given a mode m_h, there exists, independently of the available execution time for that task, one single mode m_l with $f_l \leq f_h$ such that the energy obtained using the pair (m_h, m_l) is lower than the energy achievable using any other mode m_j ($j \neq l$, $f_j < f_h$) paired with m_h.

Let us denote with $e_{k,i}$ the energy consumed per clock cycle by task τ_k running in mode m_i. If the modes m_h and m_l are compatible, with $f_l \leq f_h$, we have shown in Appendix E that the following holds:

$$e_{k,l} \cdot (\frac{1}{f_j} - \frac{1}{f_h}) - e_{k,j} \cdot (\frac{1}{f_l} - \frac{1}{f_h}) < e_{k,h}(\frac{1}{f_j} - \frac{1}{f_l}), \forall j = 1..|\mathcal{M}| \tag{5.17}$$

It is interesting to note that Eq.5.17 and consequently the pair of compatible modes depend only on the task power profile and the frequencies that are available on the processor. To conclude the description of the discrete offline algorithm, in addition to the look-up table calculation, the table *compatible_modes* is also computed offline (Fig. 5.4, line 24), for each task. The computation is based on Eq. 5.17. The pseudocode for this algorithm is given in Appendix E.

t_s	m1	m2	m3
1.50	20	0	0
1.52	10	10	0
1.54	0	10	10
1.56	0	0	20

LUT
(a)

t_s	t_e	h
1.50	1.70	1
1.52	1.71	2
1.54	1.72	3
1.56	1.72	3

LUT

h	l
1	1
2	1
3	2
4	3

Compatible mode pairs

(b)

Figure 5.6: Look-up tables with discrete modes

5.6.2 Online Algorithm

We present in this section the algorithm that is used online to select the discrete modes and their associated number of clock cycles for the next task τ_n, based on the actual start time and precomputed values from the look-up table $\texttt{LUT}_n$.

In section 5.5.2, for the continuous voltages case, every LUT entry contains the frequency calculated by the voltage scaling algorithm. At runtime, a linear interpolation of the two consecutive LUT entries with start times surrounding the actual start time of the next task, is used to calculate the new frequency. As opposed to the continuous calculation, in the discrete case, a task can be executed using several frequencies. This makes the interpolation difficult.

Let us assume, for example, a LUT like the one illustrated in Fig.5.3(a), and a start time of 1.53 for the next task. The look-up table stores for several possible start times, the number of clock cycles associated to each execution mode. Following the same approach as in the continuous case, based on the actual start time, the number of clock cycles for each performance mode should be interpolated using the entries with start times at 1.52 and 1.54. However, such a linear interpolation cannot guarantee the correct hard real-time execution. In order to guarantee the correct timing, among the two surrounding entries, the one with a higher start time has to be used. For our example, if the actual start time is 1.53, the LUT entry with start time 1.54 should be used. The drawback of this approach, as will be shown by the experimental results in Section 5.9, is the fact that a slack of $1.54 - 1.53 = 0.01$ time units cannot be exploited by the next task. We present in the following an algorithm that does not have this drawback, and at the same time, needs a smaller memory for storing the look-up tables.

Let us consider that the LUT stores the possible start times and their corresponding end times, as well as the mode with the highest frequency, as illustrated in Fig 5.6(b). Moreover, for each task, the table of compatible modes is also calculated offline. The online algorithm is outlined in Fig. 5.7. The input consists of the

```
Algorithm:  QUASI_STATIC_VS_ONLINE_DISC
-------------------------------------------------------
Input:  - start time t_si of current task τ_i
        - Quasi-Static Scaling Table LUT_i
        - number of start time interval steps n
Output: - active modes l and h the corresponding
          number of clock cycles c_l and c_h for τ_i
-------------------------------------------------------
01:  (x,y) ← calc_st_interval(LUT_n ,t_sn )
02:  t_endi    ← maximum(t_endx ,t_endy )
03:  min=∞
04:  t_exe = t_endi − t_sti
05:  for j = h_y downto h_x {
06:    if WNC_i / f_j ≥ t_exe break;
07:    c=compatible_mode[j];
08:    (nc_c , nc_j )=compute_number_of_cycles(c,j);
09:    e=compute_task_energy(nc_c ,nc_j ,c, j);
10:    if (e<min) {
11:      min=e; l=c; h=j; nc_l = nc_c ; nc_h = nc_j ;
12:    }
13:  }
14:  return (l, h, nc_l , nc_h );
```

Figure 5.7: Pseudocode: Discrete Online Algorithm

task actual start time t_{s_i}, the quasi-static scaling table $\mathtt{LUT}_i$, the table with the compatible modes and the number of interval steps $\mathtt{i}_n$. As output, the algorithm returns the number of clock cycles to be executed in each mode. In line 01, similar to the continuous online algorithm presented in section 5.5.2, we must find the LUT entries x and y surrounding the actual start time. In the next step, using the end time values from the lines x and y, together with the actual start time, we must calculate the end time of the task (line 02). In the continuous online algorithm from section 5.5.2, the two consecutive entries x and y are interpolated, and, mathematically, it can be demonstrated that the interpolation is safe from the real-time perspective. However, in case of the discrete online algorithm presented in this section, the interpolation of the end times is not safe. The reason is that, as opposed to section 5.5.2, the functions expressing the task execution time and energy are no longer

continuous. Thus, the algorithm selects as the end time for the next task the maximum between the end times from the LUT entries x and y. In this way, the hard real-time behavior is guaranteed.

At this point, given the actual start and the end time, we must determine the two active modes and the number of clock cycles to be executed in each. This is done in lines 05–13. From the LUT, the upper and the lower bound h_y and h_x of the higher execution mode are extracted. Using the table of compatible modes calculated offline, for each possible pair having the higher mode in the interval $[h_x, h_y]$, the number of cycles in each mode and the resulting energy consumption are calculated (line 07-09). The pair that provides the lowest energy is selected (lines 10-12). The algorithm finishes either when all the modes in the interval $[h_x, h_y]$ have been inspected, or, when, during the for loop, a mode m_i that cannot satisfy the timing requirements is encountered (line 06).

The complexity of the online algorithm increases linearly with the number of available performance modes $|\mathcal{M}|$. It is important to note that real processors have only a reduced set of modes. Furthermore, due to the fact that we use consecutive entries from the look-up table, the difference $h_y - h_x$ will be even smaller (typically 0, 1), leading to a low online overhead.

5.6.3 Consideration of the Mode Transition Overheads

As shown in Chapter 3, it is important to carefully consider the overheads resulted due to the transition between different execution modes. We have presented in Chapter 3 optimal algorithms as well as a heuristic that address this issue, assuming an offline optimization for the worst-case. The consideration of the mode switching overheads is particularly interesting in the context of the expected case optimization presented in this chapter. Furthermore, since the actual modes that are used by the tasks are only known at runtime, the online algorithm has to be aware of the switching overheads. We have shown in section 5.6.1 that at most two modes are used during the execution of a task. The overhead aware optimization has to decide, in which order to use these modes. Intuitively, starting a task in a mode with a lower frequency can potentially lead to a better energy consumption than starting with a higher frequency. This is the key difference between an optimization that is aware of early completion times and a worst-case optimization. Of course, it is not always better to start with the lower mode. We will deal with this issue in the following.

Let us consider the example shown in Fig. 5.8. Task τ_1 is running in mode m_1. The voltage scaling algorithm has assigned x clock cycles of τ_2 to mode m_3 and y clock cycles to mode m_2. τ_2 executes the expected case ENC_2 cycles. The overhead implied by a transition between mode i and j is $\varepsilon_{i,j}$. The energy consumed per

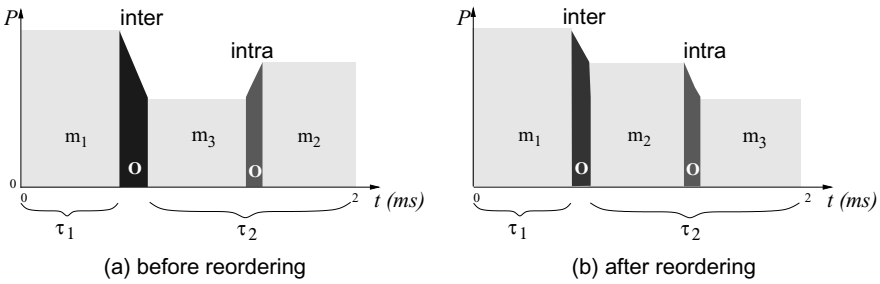


Figure 5.8: Mode Transition Overheads

clock cycle in mode i by task τ_2 is denoted as e_i. If we examine the schedules presented in Fig. 5.8(a) and (b), we notice that in the first case, the energy overhead is $\varepsilon_{1,3}+\varepsilon_{3,2}$ versus $\varepsilon_{1,2}+\varepsilon_{2,3}$ for the second schedule. We denote the energy resulted from the schedule in Fig. 5.8(a) and (b) by E_a and E_b, respectively. Depending on the relation between ENC_2, x and y we can distinguish four possible scenarios:

1) $x \geq ENC_2$, $y \geq ENC_2$

In this case, it is expected that only one mode will be used at runtime, since in both execution modes m_2 and m_3 we have a number of clock cycles higher than the expected one.

$$E_a = \varepsilon_{1,3} + ENC_2 \cdot e_3 \tag{5.18}$$

$$E_b = \varepsilon_{1_2} + ENC_2 \cdot e_2 \tag{5.19}$$

In this case, it is more energy efficient to begin the execution of τ_2 in mode m_3 ($E_a \leq E_b$), if $ENC_2 \geq \frac{\varepsilon_{1,3}-\varepsilon_{1,2}}{e_2-e_3}$.

2) $x \leq ENC_2$, $y \leq ENC_2$

As opposed to the previous case when only one mode is used online, in this case it is expected that two modes will be used at runtime. The energy consumption in each alternative is:

$$E_a = \varepsilon_{1,3} + e_3 \cdot ENC_2 \tag{5.20}$$

$$E_b = \varepsilon_{1,2} + e_2 \cdot ENC_2 \tag{5.21}$$

Thus, it is more energy efficient to begin the execution of τ_2 in mode m_3 if $ax + y + ENC_2 \geq \frac{\varepsilon_{1,3}+\varepsilon_{3,2}-\varepsilon_{1,2}-\varepsilon_{2,3}}{e_2-e_3}$.

3) $x \geq ENC_2$, $y \leq ENC_2$

$$E_a = \varepsilon_{1,3} + ENC_2 \cdot e_3 \quad (5.22)$$

$$E_b = \varepsilon_{1_2} + y \cdot e_2 + \varepsilon_{2,3} + (ENC_2 - y) \cdot e_3 \quad (5.23)$$

It is more energy efficient to begin the execution of τ_2 in mode m_3 if $y \geq \frac{\varepsilon_{1,3} - \varepsilon_{1,2} - \varepsilon_{2,3}}{e_2 - e_3}$.

4) $x \leq ENC_2$, $y \geq ENC_2$ Similar to the previous case, if $x \geq \frac{\varepsilon_{1,3} - \varepsilon_{1,2} + \varepsilon_{2,3}}{e_2 - e_3}$, it is better to start τ_2 in mode m_3.

The four possible scenarios identified before are included at the end of the online algorithm in Fig. 5.7.

5.7 Calculation of the Look-Up Table Sizes

In this section we address the problem of how many entries to assign to each LUT under a given memory constraint, such that the resulting entries yield high energy savings. As we have mentioned before, the number of entries in the LUT of each task has an influence on the solution quality, i.e., the energy consumption. This is due to the fact that the frequency and voltage approximations in the online algorithm become more accurate as the number of points increases.

A simple approach to distribute the memory among the LUTs is to allocate the same number of entries for each LUT. However, due to the fact that different tasks have different start time interval sizes and nominal energy consumptions, the memory should be distributed using a more effective scheme (i.e. reserving more memory for critical tasks). In the following we will introduce a heuristic approach to solve the LUT size problem. The two main parameters that determine the criticality (in the sense that it should be allocated more entries in the LUT) of a task τ_i are the size of the interval of possible start times ($LST_i - EST_i$) and the nominal expected energy consumption (E_i). The expected energy consumption of a task E_i is the energy consumed by that task when executing the expected number of clock cycles (ENC_i) at the nominal voltages. Consequently, in order to allocate the n_i look-up table entries for each tasks, we use the following formula:

$$n_i = NL \cdot \frac{E_i \cdot (LST_i - EST_i)}{\sum_{i=1}^{NT} E_i \cdot (LST_i - EST_i)} \quad (5.24)$$

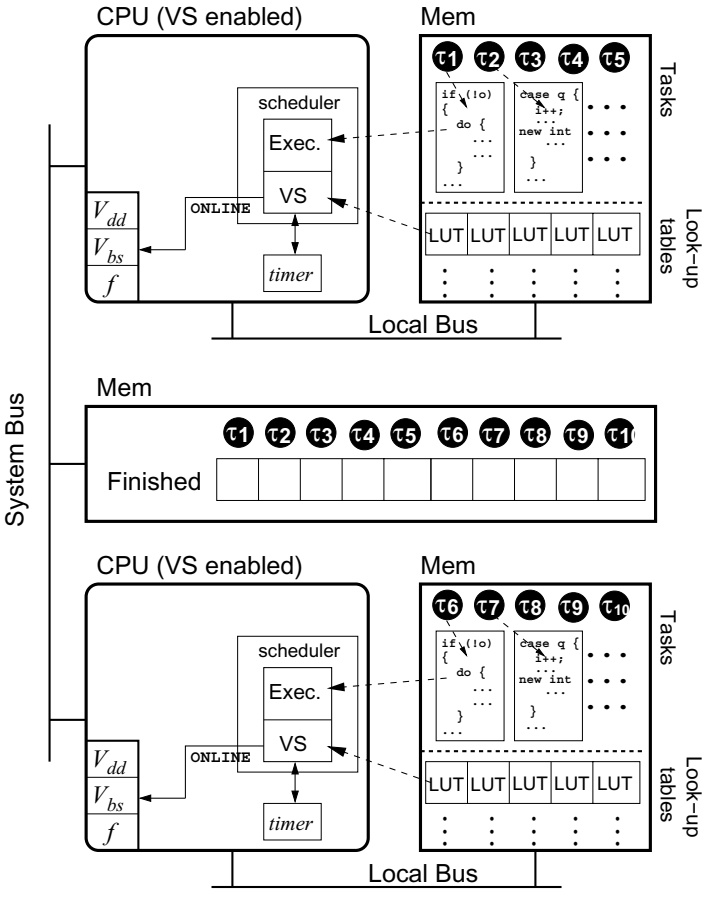


Figure 5.9: Multiprocessor system architecture

5.8 Quasi-Static Voltage Scaling for Multiprocessor Systems

In this section we address the online voltage scaling problem for multiprocessor systems. We consider that the mapping of the tasks on the processors and the schedule are given. Similar to the single processor problem, the aim is to reduce the energy consumption by exploiting dynamic slack resulted from tasks that require less execution cycles than in their worst-case. For efficiency reasons, the same quasi-static approach, based on storing a look-up table LUT_i for each task τ_i, is used.

The hardware architecture is depicted in Fig. 5.9, assuming, for example, a system with two processors. Please note that each processor has a dedicated memory that stores the instructions and data for the tasks mapped on it, and, their look-up tables. The dedicated memories are connected to the corresponding processor via a local bus. The shared memory, connected to the system bus, is for synchronization and it records, for each task, wheather or not it has completed the execution. When a task ends, it marks the corresponding entry in the shared memory. This information is used by the scheduler, invoked when a task finishes, on the processor where the finished task is mapped. The scheduler has to decide when to start and which performance modes to assign to the next task on that processor. The next task, determined by an offline schedule, can start only when its predecessors, from all the other processors, have finished. The performance modes are calculated using the look-up tables.

The quasi-static algorithms presented in section 5.5 and 5.6 were designed for systems with a single processor. Nevertheless, they can also be used in the case of multiprocessor systems, with a few modifications:

1) **Continuous approach**

In the offline algorithm from Section 5.5.1, Eq. 5.4 that captures the precedence constraints between tasks, has to be replaced by:

$$s_k + t_k \leq s_l \quad \forall (k,l) \in \mathcal{E} \tag{5.25}$$

Please remember from Chapter 3 that $\mathcal{E}$ is the set of all edges in the extended task graph (precedence constrains and scheduling dependencies).

2) **Discrete approach**

In the offline algorithm from Section 5.6.1, Eq. 5.14 that captures the precedence constraints, has to be replaced by:

$$s_k + \sum_{m \in \mathcal{M}} t_{k,m} \leq s_l \quad \forall (k,l) \in \mathcal{E} \tag{5.26}$$

Both online algorithms described in Sections 5.5.2 and 5.6.2 can be used without modifications.

At this point, it is interesting to note that the correct real-time behavior is guaranteed even in the case of a multiprocessor system. The key is the fact that all the tasks, no matter when they are started, will complete before or at their latest finishing time.

5.9 Experimental Results

We have conducted several experiments using numerous generated benchmarks as well as a real-life application, in order to demonstrate the applicability of the proposed approach. The processor parameters have been adopted from [MFMB02].

The first set of experiments was conducted in order to investigate the quality of the results provided by different online voltage selection techniques in the case when their actual run-time overhead is ignored. In Fig. 5.10(a) we show the results obtained with the following five different approaches:

1) the ideal online voltage selection approach (the scheduler that calculates the optimal voltage selection with no overhead).

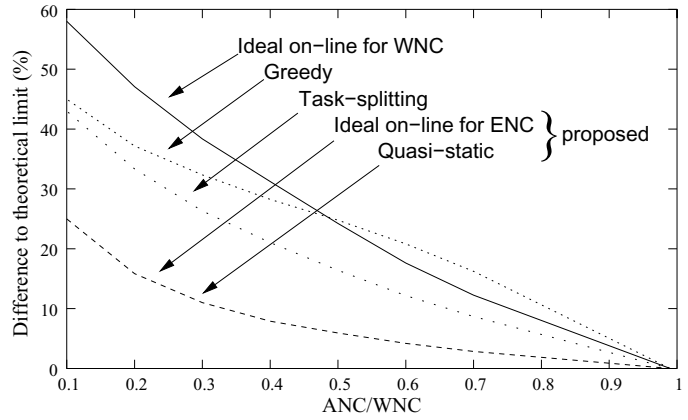
2) the quasi-static voltage selection technique proposed in this chapter in Section 5.5.

3) the greedy heuristic proposed in [AMMMA01].

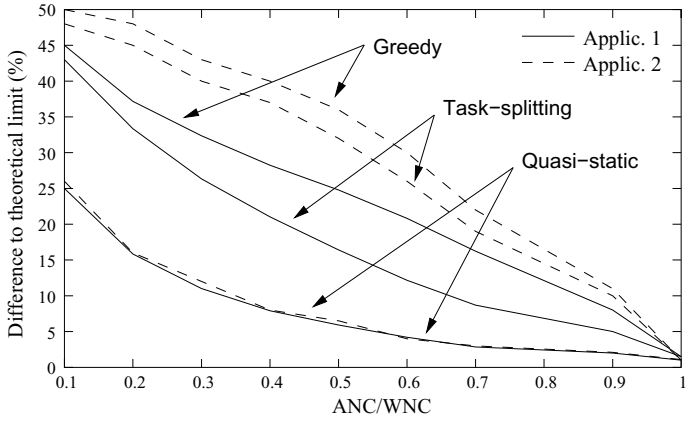
4) the task splitting heuristic proposed in [ZM04].

5) the ideal online voltage scaling algorithm for WNC proposed in [YDS95].

Originally, approaches given in [AMMMA01, ZM04, YDS95] perform DVS only. However, for comparison fairness, we have extended these algorithms towards combined supply and body-bias scaling. The results of all five techniques are given as the percentage deviation from the results produced by a hypothetical voltage scaling algorithm that would know in advance the exact number of clock cycles executed by each task. Of course such an approach is practically impossible. Nevertheless, we use this theoretical lower limit as baseline for the comparison. During the experiments, we varied the ratio of actual number of clock cycles (ANC) and worst case number of clock cycles (WNC) from 0.1 to 1 with a step width of 0.1. For each step, 1000 randomly generated task graphs were evaluated, resulting in a total of 10000 evaluations for each plot. As mentioned earlier, for this first experiment we ignored the computational overheads of all the investigated approaches, i.e., we assumed that the voltage scaling requires zero time. Furthermore, the actual number of clock cycles (ANC) are set based on a normal distribution using the expected number of cycles (ENC) as the mean value. Observing Fig. 5.10(a) leads to the following interesting conclusions. Firstly, if the



(a) Scaling for the expected−case execution time (assuming zero overhead)



(b) Influence of the online overhead on different online VS approaches

Figure 5.10: Experimental results: online voltage scaling

actual number of cycles (ANC) corresponds to the worst-case number (WNC), all voltage selection techniques approach the theoretical limit. In other words, if the application has been scaled for the WNC and all task execute with WNC, then all online techniques perform equally well. This, however, changes if the ANC differs from the WNC, which is always the case in practice. For instance, in the case that

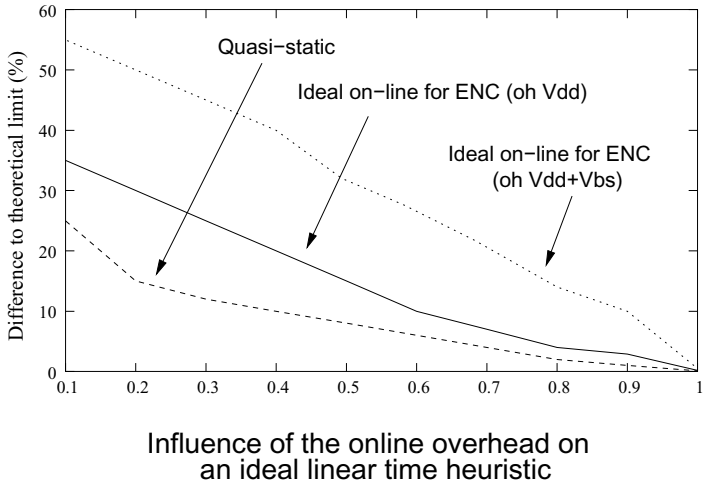


Figure 5.11: Experimental results: online voltage scaling

the ratio between ANC and WNC is 0.1, we can observe that ideal online voltage selection is 25% off the theoretical limit. On the other hand, the technique described in [YDS95] is 60% worse than the theoretical limit. The approaches based on the methods proposed in [AMMMA01, ZM04] yield results that are 42% and 45% below the theoretical optimum. Another interesting observation is the fact that the ideal online scaling and our proposed quasi-static technique produce results of the same high quality. Of course, the quality of the quasi-static voltage selection depends on the number of entries that are stored in the look-up tables (LUTs). Due to the importance of this influence, we have devoted a supplementary experiment to demonstrate how the number of entries affects the voltage selection quality. In the experiment illustrated in Fig. 5.10(a) and (b) the total number of entries was set to 4000, which was sufficient to achieve results that differed with less then 0.5% from the ideal online scaling for task graphs with up to 100 nodes. In summary, Fig. 5.10(a) demonstrates the high quality of the voltage settings produced by the quasi-static approach, which are very close of those produced by the ideal algorithm and substantially better than the values produced by any other proposed approach.

In order to evaluate the global quality of the different voltage selection approaches (taking into consideration the online overheads), we conducted two sets of experiments (Fig. 5.10(b) and Fig. 5.10(c)). In Fig. 5.10(b) we have compared our quasi-static algorithm with the approaches proposed in [AMMMA01, ZM04]. The influence of the overheads is tightly linked with the size of the applications.

Therefore, we use two sets of applications (Applic. 1 and Applic. 2) of different sizes. Applic. 1 has the size comparable to that of the MPEG encoder and Applic. 2 has a size similar to the GSM codec. As we can observe, the proposed quasi-static voltage scaling achieves considerably higher savings than the other two approaches. Although all three approaches illustrated in Fig. 5.10(b) have constant online complexity ($O(1)$), the overhead of the quasi-static approach is considerably lower. At the same time, as shown in Fig. 5.10(a), the quality of settings produced by QSVS is much higher.

In Fig. 5.11 we have compared our quasi-static approach with a hypothetical "best possible" dynamic voltage scaling algorithm. Such a hypothetical algorithm would produce the optimal voltage settings with a linear overhead similar to that of the heuristic proposed in [Gru02] (see Tab. 2). Please note that such an algorithm has not been proposed since all known optimal solutions incur a higher complexity than the one in [Gru02]. We evaluated 10000 randomly generated task graphs. In this particular experiment we set the size of the task graphs similar to the MPEG encoder. We considered two cases: the hypothetical online algorithm is executed with the overhead from [Gru02] for V_{dd}-only and with the overhead that would result if the algorithm is rewritten for the combined (V_{dd}, V_{bs}) scaling. Please note that in both of the above cases we consider that the hypothetical algorithm performs V_{dd} as well as V_{bs} scaling. As we can see, the quasi-static algorithm is superior by up to 10% even to the hypothetical algorithm with the lower V_{dd}-only overhead, while in the case which is still optimistic but closer to reality of the higher (V_{dd}, V_{bs}) overhead the superiority of the quasi-static approach is up to 30%. Overall these experiments demonstrate that the quasi-static solution is superior to any proposed and foreseeable dynamic voltage scaling approach.

The next set of experiments was conducted in order to demonstrate the influence of the memory size used for the look-up tables on the possible energy savings with the quasi-static voltage scaling. For this experiment we have used three sets of tasks graphs with 20, 50, and 100 tasks, respectively. Fig. 5.12 shows the percentage deviation of energy savings with respect to an ideal online voltage selection as a function of the memory size. For example, in order to obtain a deviation below 0.5%, a memory of 40kB is needed for systems consisting of 100 tasks. For the same quality, 20 and 8kB are needed for 50 and 20 tasks, respectively. It is interesting to observe that with a small penalty in the energy savings, the required memory decreases almost by half. For instance, for 100 tasks, the quasi-static algorithm achieves 2% deviation relative to the ideal algorithm with a memory of only 24kB. It is important to note, that in all the performed experiments we have taken into consideration the energy overhead due to the memories. This overheads have been calculated based on the energy values reported in [HAM$^+$03, MMP03] in the case of SRAM memories.

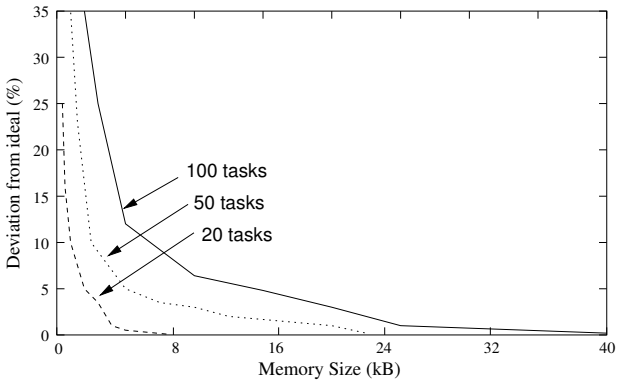


Figure 5.12: Experimental results: influence of LUT sizes

In the experiments presented until now, we have used the quasi-static algorithm based on continuous voltage selection. Another set of experiments was performed in order to evaluate the approach based on discrete voltages, presented in section 5.6. We have used taskgraphs with 50 tasks and a processor with 4 discrete modes. The 4 voltage pairs ((V_{dd}, V_{bs})) are: (1.8V,0V), (1.4V,-0.3V), (1.0V, -0.6V), (0.6V, -1.0V). The processor parameters have been adopted from [MFMB02]. The results are shown in Fig. 5.13. During this set of experiments, we have compared the energy savings achievable by two discrete quasi-static approaches. In the first approach, the LUT stores for each possible start time, the number of clock cycles associated to each mode. Online, the first LUT entry that has the start time higher than the actual one is selected. This approach, corresponding to the first alternative proposed in Section 5.6.2, is denoted in Fig. 5.13 with LUT_Y. The second approach uses the online algorithm presented in Fig. 5.7, and is denoted in Fig. 5.13 with LUT_XY. During the experiments, we varied the ratio of actual number of clock cycles (ANC) to the worst-case number of clock cycles (WNC) from 0.1 to 1, with a step width of 0.1. For each step, 1000 randomly generated task graphs were evaluated. As indicated in the figure, we present the deviation from the theoretical limit for the two approaches, assuming two different look-up table sizes: 8kB and 24 kB. Clearly, the savings achieved by both approaches depend on the size of the LUTs. We notice in Fig. 5.7 that the size of LUT has a much bigger influence in case of LUT_X than in case of LUT_XY. This is no surprise, since when LUT_X is used a certain amount of slack proportional with the distance between the LUT entries is not exploited. For a LUT size of 8kB, LUT_XY produces energy savings which can be 40% better than those produced with LUT_Y.

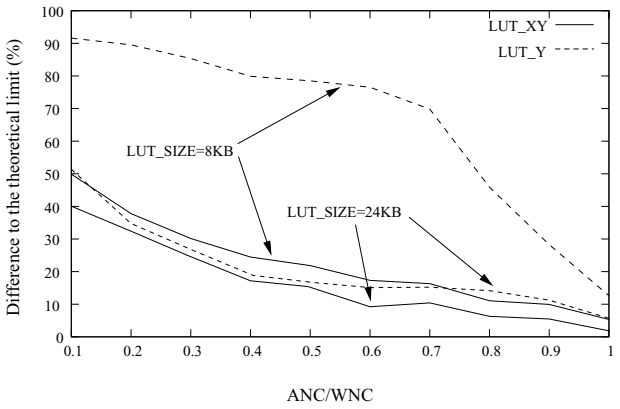


Figure 5.13: Experimental results: discrete voltage scaling

The efficiency of the multiprocessor quasi-static algorithm is investigated during the next set of experiments. We assumed an architecture composed of three processors, 50 tasks and a total LUT size of 24kB. The results are summarized in Fig. 5.13 and show the deviation from the theoretical limit of the discrete approach, considering several ratios of actual (ANC) to worst-case number of clock cycles (WNC). We can see that the trend does not change, compared to the single processor case. For example, for a ratio of 0.5 (the tasks execute half the worst-case), the quasi-static is 22% away from the ideal. At the same ratio, for the single processor case, the quasi-static approach was at 15% from the ideal algorithm. As opposed to a single processor system, in the multiprocessor case, there are tasks that are executed in parallel, potentially resulting in certain amount of slack that cannot be used by the quasi-static algorithm.

In addition to the above given benchmark results, we have conducted experiments on a real-life MPEG encoder. The MPEG encoder consists of 25 tasks

Approach	E(μJ)	Reduc. (%)
Nominal	1.63	–
Static VS	1.39	15
Greedy [AMMMA01]	0.55	67
Task Splitting [ZM04]	0.52	69
Quasi-static (cont.)	0.36	78
Quasi-static (disc.)	0.32	0.80

Table 5.3: Optimization results for the MPEG algorithm

and is considered to run on a MPC750 processor. Tab. 5.3 shows the resulting

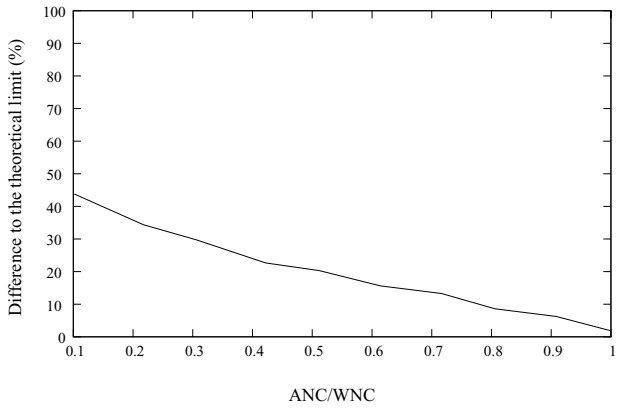


Figure 5.14: Experimental results: voltage scaling on multiprocessor systems

energy consumption obtained with different scaling approaches. The first line gives the energy consumption of the MPEG encoder running at the nominal voltages. Line two shows the result obtained with an optimal static voltage scaling approach. The energy improvement in this case is approximately 15%. Lines 3-4 show the improvements produced using the greedy online techniques proposed in [AMMMA01, ZM04] which achieve reductions of 67% and 69%, respectively. The next two rows present the results obtained by the continuous and the discrete quasi-static algorithms. The continuous algorithm improves over the nominal consumption by 78%, while the discrete one by 80%. The results confirm the high quality of the solutions produced by the quasi-static scaling technique.

Part III

Predictability of Multiprocessor Implementations

Chapter 6

Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip

Worst-case execution time (WCET) analysis and, in general, the predictability of real-time applications implemented on multiprocessor systems has been addressed only in very restrictive and particular contexts. One important aspect that makes the analysis difficult is the estimation of the system's communication behavior. The traffic on the bus does not solely originate from data transfers due to data dependencies between tasks, but is also affected by memory transfers as result of cache misses. As opposed to the analysis performed for a single processor system, where the cache miss penalty is constant, in a multiprocessor system each cache miss has a variable penalty, depending on the bus contention. This affects the tasks' WCET which, however, is needed in order to perform system scheduling. At the same time, the WCET depends on the system schedule due to the bus interference. In this context, we propose, an approach to worst-case execution time analysis and system scheduling for real-time applications implemented on multiprocessor SoC architectures.

6.1 Introduction and Related Work

Embedded applications, running on highly parallel architectures are becoming more and more sophisticated and, at the same time, will be used very often in applications for which predictability is very important. Classically, these are safety critical applications such as automotive, medical or avionics systems. However, recently, more and more applications in the multimedia and telecommunications area have to provide guaranteed quality of service and, thus, require a high degree of worst-case predictability [GDR05]. Such applications impose strict constraints not only in terms of their logical functionality but also with concern to timing. The objective of this chapter is to address, at the system-level, the specific issue of predictability for embedded systems implemented on current and future multiprocessor architectures. Providing predictability, along the dimension of time, should be based on scheduling analysis which, itself, assumes as an input the worst case execution times (WCETs) of individual tasks [Kop97, PEPP06]. While WCET analysis has been an investigation topic for already a long time, the basic driving force of this research has been, and still is, to improve the tightness of the analysis and to incorporate more and more features of modern processor architectures. However, one of the basic assumptions of this research is that WCETs are determined for each task in isolation and then, in a separate step, task scheduling analysis takes the global view of the system [TW04]. This approach is valid as long as the applications are implemented either on single processor systems or on multiprocessor architectures in which every processor has a dedicated, private access to an exclusively private memory. Such an architecture was assumed in Chapters 3 and 5.

The main problems that researchers have tried to solve are (1) the identification of the possible execution sequences inside a task and (2) the characterization of the time needed to execute each individual action [PB00]. With advanced processor architectures, effects due to caches, pipelines, and branch prediction have to be considered in order to determine the execution time of individual actions. There have been attempts to model both problems as a single ILP formulation [LMW96]. Other approaches combine abstract interpretation for cache and pipeline analysis with ILP formulations for path analysis [TFW00], or even integrate simulation into the WCET analysis flow [LS99, WSE02]. There have been attempts to build modular WCET estimation frameworks where the particular subproblems are handled separately [EES+03], while other approaches advocate a more integrated view [HLTW03]. More recently, preemption related cache effects have also been taken into consideration [RM05, SSE05].

The basic assumption in all this research is that, for WCET analysis, tasks can be considered in isolation from each other and no effects produced by dependencies or resource sharing have to be taken into consideration (with the very particular

exception of some research results regarding cache effects due to task preemption on monoprocessors, [SSE05]). This makes all the available results inapplicable to modern multiprocessor systems in which, for example, due to the shared access to sophisticated memory architectures, the individual WCETs of tasks are depending on the global system schedule. This is pointed out as one major unsolved issue in [TW04] where the current state of the art and future trends in timing predictability are reviewed. The only solution for the above mentioned shortcomings is to take out WCET analysis from its isolation and place it into the context of system level analysis and optimization. In this chapter we present an approach in this direction.

A framework for system level task mapping and scheduling for a similar type of platforms has been presented in Chapter 4. In order to avoid the problems related to the bus contention, a so called additive bus model has been used. This assumes that task execution times will be stretched only marginally as an effect of bus contention for memory accesses. Consequently, they simply neglect the effect of bus contention on task execution times. The experiments performed by the authors in [BGM+06] show that such a model can be applied with relatively good approximations if the bus load is kept below 60%. There are two severe problems with such an approach:

(1) In order for the additive model to be applicable, the bus utilization has to be kept low.

(2) Even in the case of such a low bus utilization, no guarantees of any kind regarding worst-case behavior can be provided.

The remainder of the chapter is organized as follows. Preliminaries regarding the system and architecture model are given in Section 6.2. The proposed bus access policies are presented in Section 6.3. Section 6.4 outlines the problem with a motivational example and is followed in Section 6.5 by the description of our proposed solution. Experimental results are given in Section 6.6.

6.2 System and Application Model

6.2.1 Hardware Architecture

In this chapter we consider multiprocessor system-on-chip architectures with a shared communication infrastructure that connects processing elements to the memories, similar to the architecture used in Chapter 4. The processors are equipped with instruction and data caches. Every processor is connected via the bus to a private memory. All accesses from a certain processor to its private memory are cached. A shared memory is used for inter-processor communication. The ac-

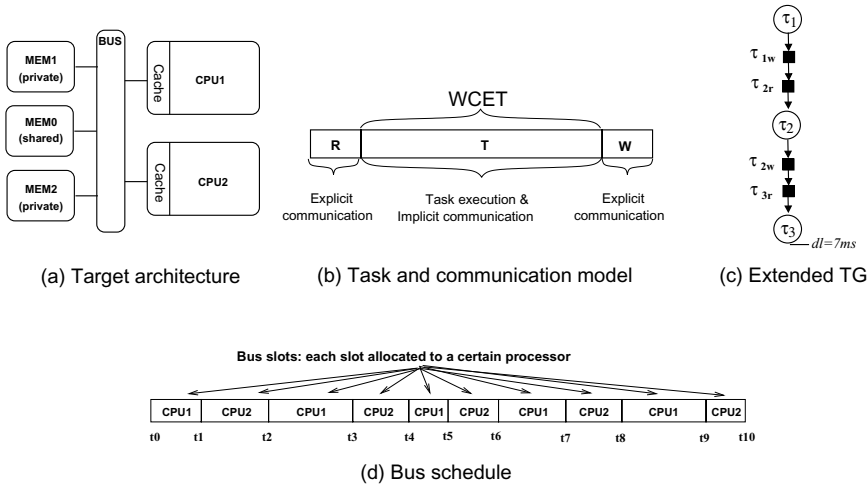


Figure 6.1: System and task models

cesses to the shared memory are not cached. This is a typical, generic, setting for new generation multiprocessors on chip, [KBP+06]. The shared communication infrastructure is used both for private memory accesses by the individual processors (if the processors are cached, these accesses are performed only in the case of cache misses) and for interprocessor communication (via the shared memory). An example architecture is shown in Fig. 6.1(a).

6.2.2 Application Model

The functionality of the software applications is captured by task graphs, $G(\Pi, \Gamma)$. Nodes $\tau \in \Pi$ in these directed acyclic graphs represent computational tasks, while edges $\gamma \in \Gamma$ indicate data dependencies between these tasks (explicit communications). The computational tasks are annotated with deadlines dl_i that have to be met at run-time. Before the execution of a data dependent task can begin, the input data must be available. Tasks mapped to the same processor are communicating through the cached private memory. These communications are handled similarly to the memory accesses during task execution. The communication between tasks mapped to different processors is done via the shared memory. Consequently, a message exchanged via the shared memory assumes two explicit communications: one for writing into the shared memory (by the sending task) and the other for reading from the memory (by the receiving task). Explicit communication is modeled in the task graph as two communication tasks, executed by the sending and the re-

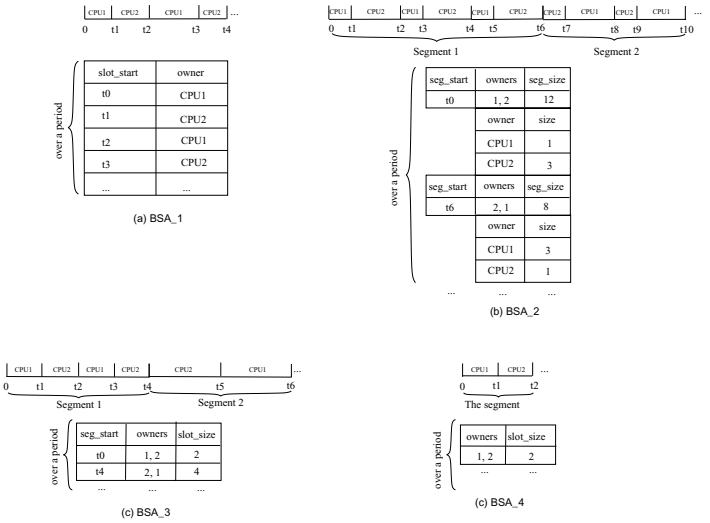


Figure 6.2: Bus Schedule Table (system with two CPUs)

ceiving processor, respectively as, for example, τ_{1w} and τ_{2r} in Fig. 6.1(c). During the execution of a task, all the instructions and data are stored in the corresponding private memory, so there will not be any shared memory accesses. The reads and writes to and from the private memories are cached. Whenever a cache miss occurs, the data has to be fetched from the memory and a cache line replaced. This results in memory accesses via the bus during the execution of the tasks. We will refer to these as implicit communication. This task model is illustrated in Fig. 6.1(b). Previous approaches that are proposing system level scheduling and optimization techniques for real-time applications only consider the explicit communication, ignoring the bus traffic due to the implicit communication [SIE06]. We will show that this leads to incorrect results in the context of multiprocessor systems.

6.3 Bus Access Policy

In order to obtain a predictable system, which also assumes a predictable bus access, we consider a TDMA-based bus sharing policy. Such a policy can be used efficiently with the contemporary SoC buses, especially if QoS guarantees are required, [SLKK02, PDBR04, GDR05].

We introduce in the following the concept of bus schedule. The bus schedule contains slots of a certain size, each with a start time, that are allocated to a proces-

sor, as shown in Fig. 6.1(d). The bus schedule is stored as a table in a memory that is directly connected to the bus arbiter. It is defined over one application period, after which it is periodically repeated. An access from the arbiter to its local memory does not generate traffic on the system bus. The bus schedule is given as input to the WCET analysis algorithm. At runtime, the bus arbiter is enforcing the bus schedule, such that when a processor sends a bus request during a slot that belongs to another processor, the arbiter will keep it waiting until the start of the next slot that was assigned to it.

The bus schedule has a strong influence on the worst-case execution time. Ideally, from the point of view of task execution times, we would like to have an irregular bus schedule, in which slot sequences and individual slot sizes are customized according to the needs of currently active tasks.Such a schedule table is illustrated in Fig. 6.2(a) for a system with two CPUs. This bus scheduling approach, denoted as BSA_1, would offer the best task WCETs at the expense of a very complex bus slot optimization algorithm and of a very large schedule table.

Alternatively, in order to reduce the controller complexity, the bus schedule is divided in segments. Each segment is an interval in which the bus schedule follows a regular pattern, in the form of TDMA rounds that are repeated throughout the segment. A round is composed of bus slots with a certain size, each slot allocated to a different processor. In Fig. 6.2(b) we illustrate a schedule consisting of two bus segments with a size of 9 and 8 time units, respectively. In the first segment, the TDMA round is repeated three times. The first slot in the round is assigned to CPU_1 and has a size of 1, the second slot, with size 2, belongs to CPU_2. The second segment consists of two rounds. The first slot (size 1) belongs to CPU_2, the second one (size 3) to CPU_1. This bus scheduling approach is denoted BSA_2.

The approach presented in Fig. 6.2(c) and denoted BSA_3 further reduces the memory needs for the bus controller. As opposed to BSA_2, in this case, all slots inside a segment have the same size.

In the final approach, BSA_4, all the slots in the bus have the same size and repeated according to a fix sequence.

6.4 Motivational Example

Let us assume a multiprocessor system, consisting of two processors CPU_1 and CPU_2, connected via a bus. Task τ_1 runs on CPU_1 and τ_2 on CPU_2. The imposed deadline is 63 time units. When τ_2 finishes, it updates the shared memory during the explicit communication E_1. We have illustrated this situation in Fig. 6.3(a). During the execution of the tasks τ_1 and τ_2, some of the memory accesses result in cache misses and consequently the corresponding caches must be refilled. The

time interval spent due to these accesses is indicated in Fig. 6.3 as M_1,M_3,M_5 for τ_1 and M_2, M_4 for τ_2. The memory accesses are executed by the implicit bus transfers I_1,I_2,I_3,I_4 and I_5. If we analyze the tasks using classical WCET analysis, we conclude that τ_1 will finish at time 57 and τ_2 at 24. For this example, we have assumed that the cache miss penalty is 6 time units. CPU_2 is controlling the shared memory update carried out by the explicit message E_1 via the bus after the end of task τ_2.

A closer look at the execution pattern of the tasks reveals that the cache misses may overlap in time. For example, the cache miss I_1 and I_2 are both happening at time 0. Similar conflicts can occur between implicit and explicit communications (for example I_5 and E_1). Since the bus cannot be accessed concurrently, a bus arbiter will allow the processors to refill the cache in a certain order. An example of a possible outcome is depicted in Fig. 6.3(b). The bus arbiter allows first the cache miss I_1, so after 6 time units needed to handle the miss, task τ_1 can continue its execution. After serving I_1, the arbiter grants the bus to CPU_2 in order to serve the miss I_2. Once the bus is granted, it takes 6 time units to refill the cache. However, CPU_2 was waiting 6 time units to get access to the bus. Thus, handling the cache miss I_2 took 12 time units, instead of 6. Another miss I_3 occurs on CPU_1 at time 9. The bus is busy transferring I_2 until time 12. So CPU_1 will be waiting 3 time units until it is granted the bus. Consequently, in order to refill the cache as a result of the miss I_3, task τ_1 is delayed 9 time units instead of 6, until time 18. At time 17, the task τ_2 has a cache miss I_4 and CPU_2 waits 1 time unit until time 18 when it is granted the bus. Compared with the execution time from Fig. 6.3(a), where an ideal, constant, cache miss penalty is assumed, task τ_2 finishes at time 31, instead of time 24. Upon its end, τ_2 starts immediately sending the explicit communication message E_1, since the bus is free at that time. In the meantime, τ_1 is executing on CPU_1 and has a cache miss, I_5 at time 36. The bus is granted to CPU_1 only at time 43, after E_1 was sent, so τ_1 can continue to execute at time 49 and finishes its execution at time 67 causing a deadline violation. The example in Fig. 6.3(b) shows that using worst-case execution time analysis algorithms that consider tasks in isolation and ignore system level conflicts leads to incorrect results.

In Fig. 6.3(b) we have assumed that the bus is arbitrated using a simple First Come First Served (FCFS) policy. In order to achieve worst-case predictability, however, we use a TDMA bus scheduling approach, as outlined in Section 6.2.

Let us assume the bus schedule in Fig. 6.3(c). According to this schedule, processor CPU_1 is granted the bus at time 0 for 15 time units and at time 32 for 7 time units. Thus, the bus is available to task τ_1 for each of its cache misses (M_1, M_3, M_5) at times 0, 9 and 33. Since these are the arrival times of the cache misses, the execution of τ_1 is not delayed and finishes at time 57, before its deadline. Task τ_2 is granted the bus at times 15 and 26 and finishes at time 39, resulting in a longer

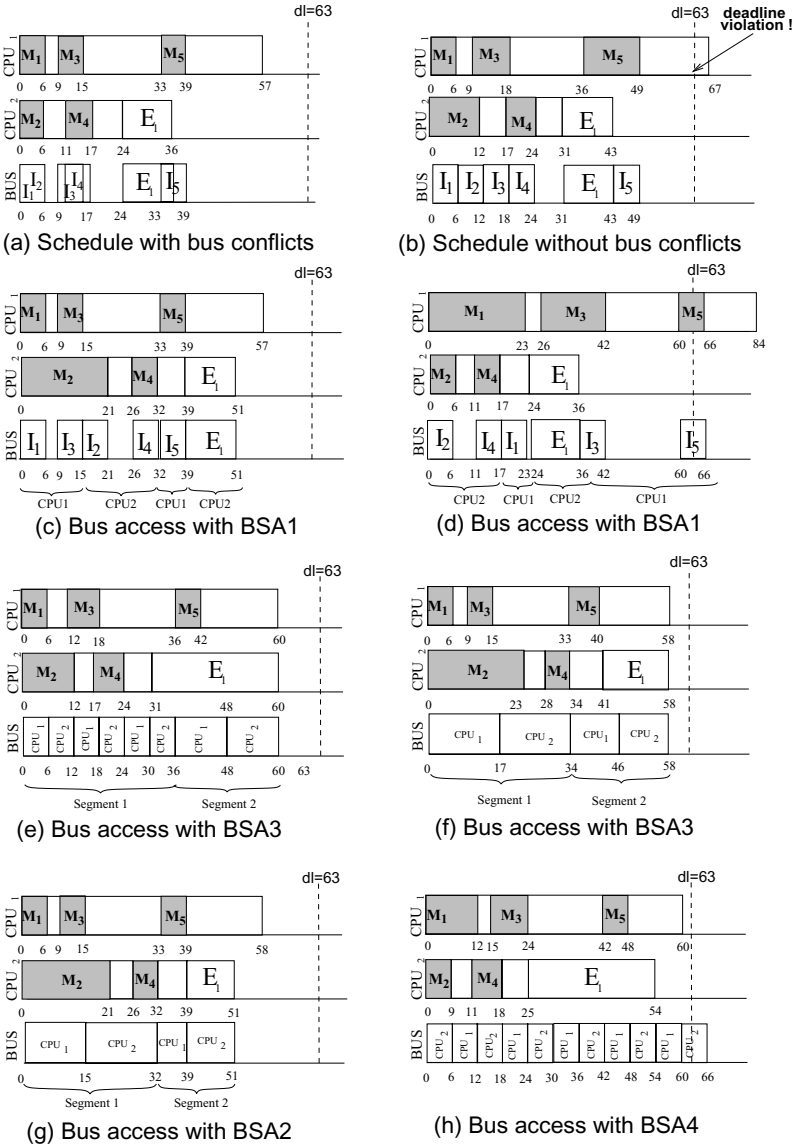


Figure 6.3: Schedule with various bus access policies

execution time than in the ideal case (time 24). The explicit communication E_1 is started at time 39 and completes at time 51.

While the bus schedule in Fig. 6.3(c) is optimized according to the requirements from task τ_1, the one in Fig. 6.3(d) eliminates all bus access delays for task τ_2. According to this bus schedule, while τ_2 will finish earlier than in Fig. 6.3(c), task τ_1 will finish at time 84 and, thus, miss its deadline.

A fine grained bus schedule, such as in Fig. 6.3(c) and (d), potentially can provide good worst-case execution times at the expense of a complex bus arbiter that requires a very large memory for storing the schedule. We will show with the next example that a simpler bus schedule, where the allocated slots follow a certain pattern, leads to a very good compromise between arbiter complexity and task delays. For example, in Fig. 6.3(e), the bus access is organized according to the BSA_3 approach. We divide the period into two segments. The slots from the first segment are assigned a size of 6 time units, while the slots in the second segment have 12 units. For this particular example, the order is kept the same in both segments, starting with CPU_1. Following this bus schedule τ_1 finishes latest at time 60, τ_2 at 31 and E_1 at 60. A different BSA_3-based schedule, with the slot size of 17 in the first segment and 12 in the second one is illustrated in Fig. 6.3(f). Please note, that different BSA3-based bus schedules may lead to different worst-case execution times. In Fig. 6.3(e), τ_1 finishes at time 60 and τ_2 at 31, while in Fig. 6.3(f) τ_1 finishes at 58 and τ_2 at 41. It is crucial to choose, during the system scheduling a BSA_3 bus schedule that favors the tasks on the critical path.

Fig. 6.3(g) illustrates the BSA2 approach. The bus schedule consists of two segments. The first one starts at time 0 and ends at 32. The slot sizes are 15 time units for CPU_1 and 17 for CPU_2. The second segment, starting at time 32 and finishing at 51 has a slot of 7 units allocated to CPU_1 and another slot of 12 units for CPU_2. If we compare the worst-case execution times obtained using BSA2 in Fig. 6.3(g), to the ones obtained using BSA3 in 6.3(e) and (f), we notice that BSA2 performs better, since it allows for a better customization of the bus schedule according to needs of the particular tasks.

The least flexible bus access policy, BSA4, is illustrated in Fig. 6.3(h). Here, the same slot size and order is kept unchanged over the whole period. This alternative, potentially produces lower quality results, but it requires a controller with a very small memory.

The examples presented in this section demonstrate two issues:

1) Ignoring bus conflicts due to implicit communication can lead to gross subestimations of WCETs and, implicitly, to incorrect schedules.

2) The organization of the bus schedule has a great impact on the WCET of tasks. A good bus schedule does not necessarily minimize the WCET of a certain task, but has to be fixed considering also the global system deadlines.

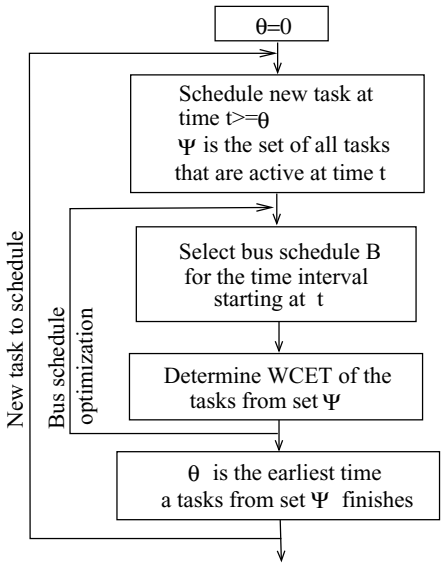
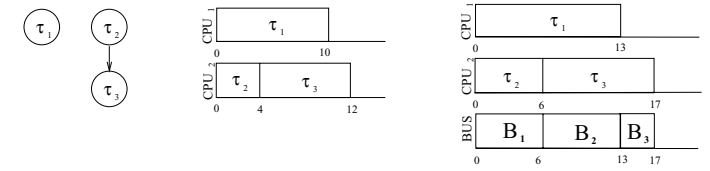


Figure 6.4: Overall Approach

6.5 Analysis, Scheduling and Optimization Flow

We consider as input the application task graph capturing the dependencies between the tasks and the target hardware platform. Each task has associated the corresponding code and potentially a deadline that has to be met at runtime. As a first stage, mapping of the tasks to processors is performed. Traditionally, after the mapping is done, the WCET of the tasks can be determined and is considered to be constant and known. However, as mentioned before, the basic problem is that memory access times are, in principle, unpredictable in the context of the potential bus conflicts between the processors that run in parallel. These conflicts (and implicitly the WCETs), however, depend on the global system schedule. System scheduling, on the other side, traditionally assumes that WCETs of the tasks are fixed and given as input. This cyclic dependency is not just a technical detail or inconvenience, but a fundamental issue with large implications and which invalidates one of the basic assumptions that support current state of the art. In order to solve this issue, we propose a strategy that is based on the following basic decisions:

1) We consider a TDMA-based bus access policy as outlined in Section 6.2. The actual bus access schedule is determined at design time and will be enforced during the execution of the application.



(a) Task graph (b) Scheduling with traditional WCET analysis (c) List scheduling with the proposed approach

Figure 6.5: System level scheduling with WCET analysis

2) The bus access schedule is taken into consideration at the WCET estimation. WCET estimation, as well as the determination of the bus access schedule are integrated with the system level scheduling process (Fig. 6.4).

We will present our overall strategy using a simple example. It consists of three tasks mapped on two processors, as in Fig. 6.5.

The system level static cyclic scheduling process is based on a list scheduling technique [CG72]. List scheduling heuristics are based on priority lists from which tasks are extracted in order to be scheduled at certain moments. A task is placed in the ready list if all its predecessors have been already scheduled. All ready tasks from the list are investigated, and that task τ_i is selected for placement in the schedule which has the highest priority. We use the modified partial critical path priority function presented in [EDPP00]. The process continues until the ready list is empty.

Let us assume that, using traditional WCET estimation (considering a given constant time for main memory access, ignoring bus conflicts), the task execution times are 10, 4, and 8 for τ_1, τ_2, and τ_3, respectively. Classical list scheduling would generate the schedule in Fig. 6.5(b), and conclude that a deadline of 12 can be satisfied.

In our approach, the list scheduler will choose tasks τ_1 and τ_2 to be scheduled on the two processors at time 0. However, the WCET of the two tasks is not yet known, so their worst case termination time cannot be determined. In order to calculate the WCET of the tasks, a bus configuration has to be decided on. This configuration should, preferably, be fixed so that it is favorable from the point of view of WCETs of the currently running tasks (τ_1 and τ_2, in our case). Given a certain bus configuration, our WCET-analysis will determine the WCET for τ_1 and τ_2. Inside an optimization loop, several alternative bus configurations are considered. The goal is to reduce the WCET of τ_1 and τ_2, with an additional weight on reducing the WCET of that task that is assumed to be on the critical path (in our case τ_2).

Let us assume that B1 is the selected bus configuration and the WCETs are 12 for τ_1 and 6 for τ_2. At this moment the following is already decided: τ_1 and τ_2 are scheduled at time 0, τ_2 is finishing, in the worst case, at time 6, and the bus configuration B1 is used in the time interval between 0 and 6. Since τ_2 is finishing at time 6, in the worst case, the list scheduler will schedule task τ_3 at time 6. Now, τ_3 and τ_1 are scheduled in parallel. Given a certain bus configuration B, our WCET analysis tool will determine the WCETs for τ_1 and τ_3. For this, it will be considered that τ_3 is executing under the configuration B, and τ_1 under configuration B1 for the time interval 0 to 6, and B for the rest. Again, an optimization is performed in order to find an efficient bus configuration for the time interval beyond 6. Let us assume that the bus configuration B2 has been selected and the WCETs are 9 for τ_3 and 13 for τ_1. The final schedule is illustrated in figure 6.5.

The overall approach is illustrated in Fig. 6.4. At each iteration, the set ψ of tasks that are active at the current time t, is considered. In an inner optimization loop a bus configuration B is fixed. For each candidate configuration the WCET of the tasks in the set ψ is determined. During the WCET estimation process, the bus configurations determined during the previous iterations are considered for the time intervals before t, and the new configuration alternative B for the time interval after t. Once a bus configuration B is decided on, θ is the earliest time a task in the set ψ terminates. The configuration B is fixed for the time interval (t, θ], and the process continues from time θ, with the next iteration.

In the above discussion, we have not addressed the explicit communication of messages on the bus, to and from the shared memory. As shown in Section 6.2, a message exchanged via the shared memory assumes two explicit communications: one for writing into the shared memory (by the sending task) and the other for reading from the memory (by the receiving task). Explicit communication is modeled in the task graph as two communication tasks, executed by the sending and the receiving processor, respectively (Fig. 6.1 in section 6.2). A straightforward way to handle these communications would be to schedule each as one compact transfer over the bus. This, however, would be extremely harmful for the overall performance, since it would block, for a relatively long time interval, all memory access for cache misses from active processes. Therefore, the communication tasks are considered, during scheduling, similar to the ordinary tasks, but with the particular feature that they are continuously requesting for bus access (they behave like a hypothetical task that continuously generates successive cache misses such that the total amount of memory requests is equal to the worst case message length). Such a task is considered together with the other currently active tasks in the set Ψ. Our algorithm will generate a bus configuration and will schedule the communications such that it efficiently accommodates both the explicit message communication as well as the memory accesses issued by the active tasks.

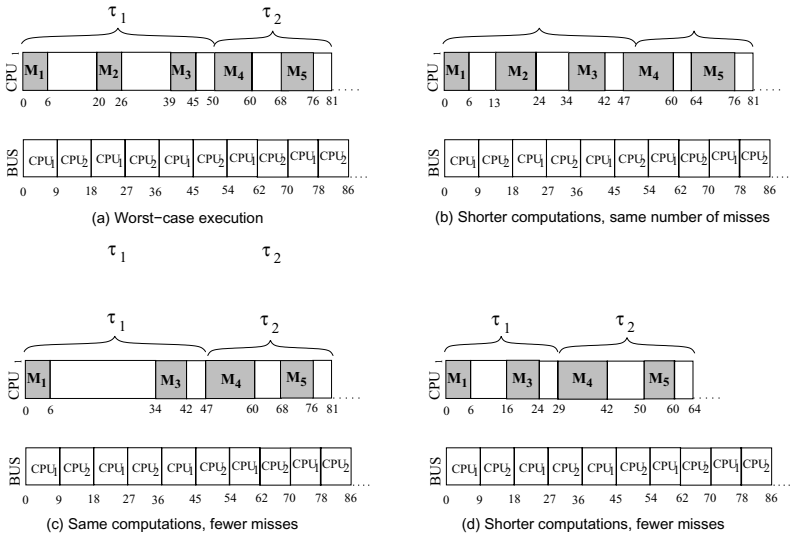


Figure 6.6: Tasks executing less than their worst-case

It is important to mention that the approach proposed in this chapter guarantees that the worst-case bounds derived by our analysis are correct even when the tasks execute less than their worst-case. The assumption is that, at runtime, tasks are scheduled nonpreemptive and the offline determined order is enforced. We will illustrate the demonstration with the example from Fig. 6.6. Let us assume a system composed of two processors, CPU_1 and CPU_2, interconnected via a bus. For simplicity, in Fig. 6.6, only the tasks τ_1 and τ_2 mapped on CPU_1 are shown. Fig. 6.6(a) shows the execution of the two tasks, assuming the worst-case, derived using the illustrated bus schedule. 3 cache misses (M_1, M_2 and M_3) are predicted in the worst-case for τ_1, while for τ_2 there are 2 predicted misses (M_4 and M_5). It is important to note that an earlier bus request issued by a processor does not affect any task mapped on another processor, due to the fact that the bus slots are assigned exclusively to processors. In general, tasks can execute less than their worst-case in the following cases:

1) If instruction sequences finish in shorter time than predicted by the worst-case analysis. An example is given in Fig. 6.6(b), for the two tasks with the worst-case illustrated in Fig. 6.6(a). The worst-case number of cache misses occurs for both tasks. However, the sequence of instructions after M_1 finishes earlier than predicted and M_2 occurs at time 13, instead of 20. As the bus is granted to CPU_2 until time 18, M_2 is served starting at time 18, in the same slot predicted by the

worst-case analysis, but 2 time units earlier. Similarly, M_3 is served earlier than predicted. As a result, τ_1 finishes ahead of its estimated worst-case, at time 47 instead of 50. Thus τ_2 may start at time 47. Due to the bus schedule, M_4 is served at time 60, identical to the worst-case estimation. Similarly, M_5 occurs earlier, but is served at time 76. To conclude, if instruction sequences finish in shorter time, tasks may end at earlier times, but never later then estimated by the worst-case analysis. This is due to the fact that any cache miss is served latest at the time predicted by the analysis.

2) If a memory access results in a hit, although predicted as a miss during the worst-case analysis. An example is given in Fig. 6.6(c). During the execution of τ_1, M_2 is a hit, and thus M_3 occurs at time 34. It is served by the slot predicted by the worst-case analysis. Similarly, M_4 occurs earlier but is served at the time predicted in the worst-case. To conclude, even if some of the memory accesses considered as misses by the analysis result in hits at runtime, the remaining misses are served in the worst-case at the predicted.

3) Fig. 6.6(d) illustrates a combination of the previous two cases, with shorter instruction sequences and fewer cache misses than estimated. Even in this case, cache misses that occur earlier than predicted in the worst-case will, possibly, be served by an earlier bus slot than predicted, but never by a later one than considered during the WCET analysis.

In the following section we will address the WCET estimation algorithm.

6.5.1 WCET Analysis

We will present the algorithm used for the computation of the worst-case execution time of a task, given a start time and a bus schedule [1]. Our approach builds on techniques developed for "traditional" WCET analysis. Consequently, it can be adapted on top of any WCET analysis approach that handles prediction of cache misses. Our technique is also orthogonal to the issue of cache associativity supported by this cache miss prediction technique. The current implementation is built on top of the approach described in [WSE02, SE06] that supports set associative and direct mapping.

In a first step, the control flow graph (CFG) is extracted from the code of the task. The nodes in the CFG represent basic blocks (consecutive lines of code without branches) or control nodes (capturing conditional instructions or loops). The edges capture the program flow. In Fig. 6.7(a) and (b), we have depicted an example task containing a for loop and the corresponding CFG, extracted from this

[1] In the classical approach WCET analysis returns the worst-case time interval between the start and the finishing of a task. In our case, what we determine is the worst-case finishing time of the task.

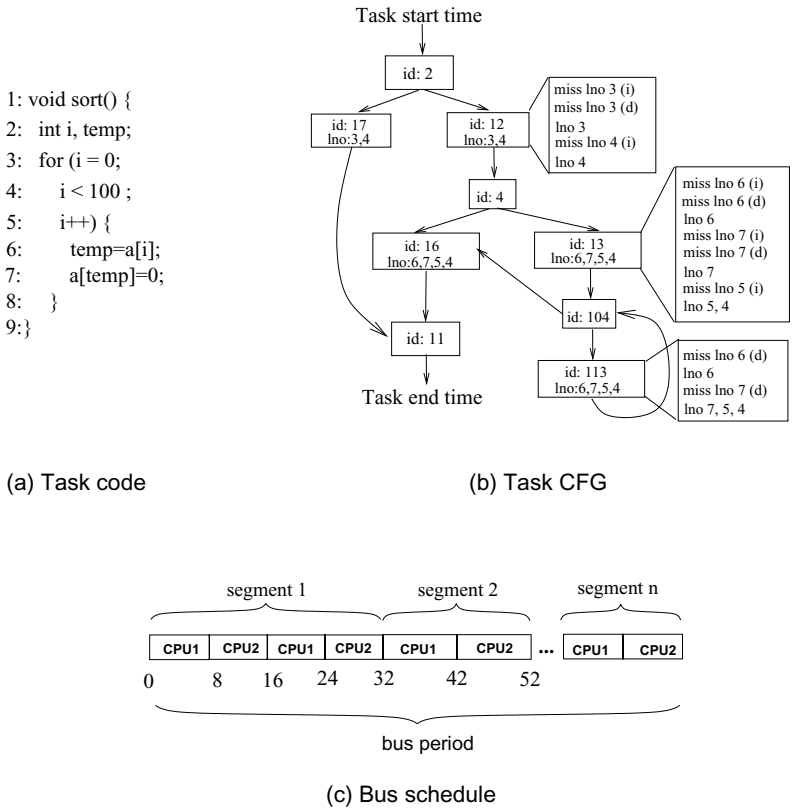


Figure 6.7: Example task WCET calculation

task. For the nodes associated to basic blocks we have depicted the code line numbers. For example, node 12 (id:12) captures the execution of lines 3 ($i = 0$) and 4 ($i < 100$). A possible execution path, with the for loop iteration executed twice, is given by the node sequence 2, 12, 4 and 13, 104, 113, 104, 16, 11. Please note that the for loop was automatically unrolled once when the CFG was extracted from the code (nodes 13 and 113 correspond to the same basic block representing an iteration of the for loop). This is useful when performing the instruction cache analysis. Intuitively, when executing a loop, at the first iteration all the instruction accesses result in cache misses. However, during the next iterations there is a chance to find the instructions in the cache.

We have depicted in Fig. 6.7(b) the resulting misses obtained after performing instruction (marked with an "i") and data (marked with an "d") cache analysis. For example, let us examine the nodes 13 and 113 from the CFG. In node 13, we obtain instruction cache misses for the lines 6, 7 and 5, while in the node 113 there is no instruction cache miss. In order to study at a larger scale the interaction between the basic blocks, data flow analysis is used. This propagates between consecutive nodes from the CFG the addresses that are always in the cache, no matter which execution path is taken. For example, the address of the instruction from line 4 is propagated from the node 12 to the nodes 13, 16 and 113.

Let us consider now the data accesses. While the instruction addresses are always known, this is not the case with the data [SE06, RM05]. This, for example, is the case with an array that is accessed using an index variable whose value is data dependent, as in Fig.6.7(a), on line 7. Using data dependency analysis performed on the abstract syntax tree extracted from the code of the task, all the data accesses are classified as predictable or unpredictable [SE06]. For example, in Fig.6.7(a) the only unpredictable data memory access is in line 7. The rest of the accesses are predictable. All the unpredictable memory accesses are classified as cache misses. Furthermore, they have a hidden impact on the state of the data cache. A miss resulted from an unpredictable access replaces an unknown cache line. One of the following predictable memory accesses that would be considered as hit otherwise, might result in a miss due to the unknown line replacement. Similar to the instruction cache, dataflow analysis is used for propagating the addresses that will be in the cache, no matter which program path is executed.

Until this point, we have performed the same steps as the traditional WCET analysis that ignores resource conflicts. In the classical case, the analysis would continue with the calculation of the execution time of each basic block. This is done using local basic block simulations. The number of clock cycles that are spent by the processor doing effective computations, ignoring the time spent to access the cache (hit time) or the memory (miss penalty) is obtained in this way. Knowing the number of hits and misses for each basic block and the hit and miss penalties, the worst case execution time of each CFG node is easily computed. Taking into account the dependencies between the CFG nodes and their execution times, an ILP formulation can be used for the task WCET computation, [SE06, RM05, LMW96].

In a realistic multiprocessor setting, however, due to the variation of the miss penalties as a result of potential bus conflicts, such a simple approach does not work. The main difference is the following: in traditional WCET analysis it is sufficient for each CFG node to have the total number of misses. In our case, however, this is not sufficient in order to take into consideration potential conflicts. What is needed is, for each node, the exact sequence of misses and the worst-case duration of computation sequences between the misses. For example, in the case of node 13

in Fig. 6.7(b), we have three instruction sequences separated by cache misses: (1) line 6, (2) line 7 and (3) lines 5 and 4. Once we have annotated the CFG with all the above information, we are prepared to solve the actual problem: determine the worst-case execution time corresponding to the longest path through the CFG. In order to solve this problem, we have to determine the worst-case execution time of a node in the CFG. In the classical WCET analysis, a node's WCET is the result of a trivial summation. In our case, however, the WCET of a node depends on the bus schedule and also on the node's worst-case start time.

Let us assume that the bus schedule in Fig. 6.7(c) is constructed. The system is composed of two processors and the task we are investigating is mapped on $CPU1$. There are two bus segments, the first one starting at time 0 and the second starting at time 32. The slot order during both segments is the same: $CPU1$ and then $CPU2$. The processors have slots of equal size during a segment.

The start time of the task that is currently analyzed is decided during the system level scheduling (see section 6.5) and let us suppose that it is 0. Once the bus is granted to a processor, let us assume that 6 time units are needed to handle a cache miss. For simplicity, we assume that the hit time is 0 and every instruction is executed in 1 time unit.

Using the above values and the bus schedule in Fig. 6.7(c), the node 12 will start its execution at time 0 and finish at time 39. The instruction miss (marked with "i" in Fig. 6.7(b)) from line 3 arrives at time 0, and, according to the bus schedule, it gets the bus immediately. At time 6, when the instruction miss is solved, the execution of node 12 cannot continue because of the data miss from line 2 (marked with "d"). This miss has to wait until time 16 when the bus is again allocated to $CPU1$ and, from time 16 to time 22 the cache is updated. Line 3 is executed starting from time 22 until 23, when the miss generated by the line 4 requests the bus. The bus is granted to $CPU1$ at time 32, so line 4 starts to be executed at time 38 and is finished, in the worst case, at time 39.

In the following we will illustrate the algorithm that performs the WCET computation for a certain task. The algorithm must find the longest path in the control flow graph. For example, there are four possible execution paths (sequences of nodes) for the task in Fig. 6.7(a) that are captured by the CFG in Fig. 6.7(b):

(1): 2, 17, 11,

(2): 2, 12, ,4, 16, 11,

(3): 2, 12, 4, 13, 104, 16, 11

(4): 2, 12, 4, 13, 104, 113, 104, ..., 104, 16, 11.

The execution time of a particular node in the CFG can be computed only after the execution times of all its predecessors are known. For example, the execution time of node 16 can be computed only after the execution time for the nodes 4 and 104 is fixed. At this point it is interesting to note that the node 104 is the entry in

a CFG loop (104, 113, 104). Due to the fact that the cache miss penalties depend on the bus schedule, the execution times of the loop nodes will be different at each loop iteration. Thus, loop nodes in the CFG must be visited a number of times, given by the loop bound (extracted automatically or annotated in the code). In the example from Fig. 6.7(b), the node 113 is visited 99 times (the loop is executed 100 times, but it was unrolled once). Each time a loop node is visited, its start time is updated and a new end time is calculated using the bus schedule, in the manner illustrated above for node 12. Consequently, during the computation of the execution time of the node 16, the start time is the maximum between the end time of the node 4 and node 104, obtained after 99 iterations. The worst-case execution time of the task will be the end time of the node 11.

The worst-case complexity of the WCET analysis is exponential (this is also the case for the classical WCET analysis). However, in practice, the approach is very efficient, as experimental results presented in Section 6.6 show.

6.5.2 Bus Schedule Optimization

The bus schedule is a key parameter that influences the worst-case execution time of the tasks. As shown in Section 6.5 the bus schedule is determined during the system scheduling process. Referring to Fig. 6.4, successive portions of the bus schedule are fixed during the internal optimization loop. The aim is to find a schedule for each portion, such that globally the worst-case execution time is minimized. In order to find an efficient bus schedule for each portion, information produced by the WCET analysis is used in the optimization process. In particular, this information captures the distribution of the cache misses along the detected worst case paths, for each currently active task (for each task in set Ψ). We have deployed several bus access optimization algorithms, specific to the proposed bus schedule alternative (BSA_1, BSA_2, BSA_3, BSA_4).

In the case of BSA_1, each portion of the bus schedule (fixed during the internal optimization loop in Fig.6.4) is determined without any restriction. For BSA_2 and BSA_3, each portion of the bus schedule corresponds to a new bus segment, as defined in section 6.3. In case of BSA_2, the optimization has to find, for each segment, the size of the slots allocated for each processor, as well as their order. The search space for BSA_3 is reduced to finding for each bus segment, a unique slot size and an order in which the processors will access the bus.

In the case of BSA_4, the slot sequence and size is unique for the whole schedule. Therefore, the scheme in Fig. 6.4 is changed: a bus configuration alternative is determined before system scheduling and the list scheduling loop is included inside the bus optimization loop.

The bus access optimization process is based on a simulated annealing strategy [OvG89, Ree93] and determines the order and size of slots according to the restrictions imposed by the selected bus scheduling approach. The cost function used by the simulated annealing algorithm in order to select a schedule for each segment of the bus is:

$$\sum_{\tau_i \in \Psi} \frac{current\_wcet_i - ideal\_wcet_i}{ideal\_wcet_i} \cdot time\_left(CPU[\tau_i]) \tag{6.1}$$

Two types of parameters are used in the cost function. The first category includes the parameters $ideal\_wcet_i$ and $time\_left(CPU[\tau_i])$ that capture the global importance of this task relative to the other tasks in the system. They are independent of the actual bus schedule used. $ideal\_wcet_i$ is the worst-case execution time of the task τ_i, assuming that no conflicts occur on the bus. $time\_left(CPU[\tau_i])$ captures the total worst-case number of clock cycles remaining to be executed on the processor where task τ_i is mapped, assuming that the bus can be accessed without conflicts. At each list scheduling iteration, after a task is scheduled, the corresponding $time\_left$ variable is updated. Thus, the cost function is guided to select bus optimization that favor the tasks from the critical path.

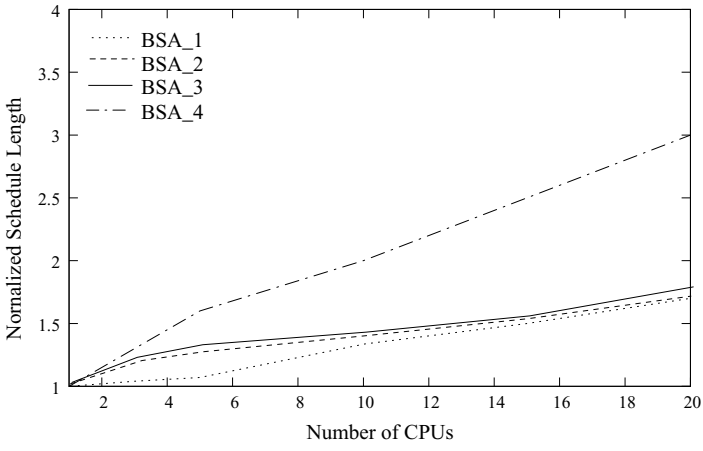
The second category of parameters in the cost function consists of $current\_wcet_i$. This parameter depends on the actual bus schedule considered. $current\_wcet_i$ captures the worst-case execution time of the task τ_i, obtained with the current candidate bus schedule. In order to treat the tasks fairly, independent of their size, the deviation of the execution time obtained with the current bus schedule relative to the ideal no conflicts case is normalized against the ideal execution time.

The simulated annealing strategy shortly outlined here serves as a proof of concept for the experiments presented in the next section. An improved algorithm that addresses the same problem is presented in [RAEP07].

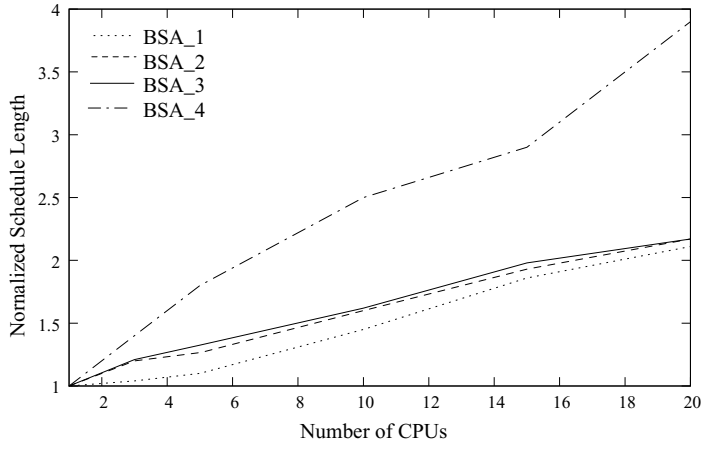
6.6 Experimental Results

The complete flow illustrated in Fig. 6.4 has been implemented and used as a platform for the experiments presented in this section. They were run on a dual core Pentium4 processor at 2.8 GHz.

First, we have performed experiments using a set of synthetic benchmarks consisting of random task graphs with the number of tasks varying between 50 and 200. The tasks are mapped on architectures consisting of 2 to 20 processors. The tasks are corresponding to CFGs extracted from various C programs (e.g. sorting, searching, matrix multiplications, DSP algorithms). For the WCET analysis, it was



(a)



(b)

Figure 6.8: The four bus access policies

assumed that ARM7 processors are used. We have assumed that 12 clock cycles are required for a memory access due to a cache miss.

We have explored the efficiency of the proposed approach in the context of the four bus scheduling approaches introduced in Section 6.3. The results are presented

in Fig. 6.8. We have run experiments for configurations consisting of 2, 4, 6, ... 20 processors. For each configuration, 50 randomly generated task graphs were used. For each task graph, the worst-case schedule length has been determined in 5 cases: the four bus scheduling policies BSA_1 to BSA_4, and a hypothetical ideal situation in which memory accesses are never delayed. This ideal schedule length (which in practice, is unachievable, even by a theoretically optimal bus schedule) is considered as the baseline for the diagrams presented in Fig. 6.8. The diagram corresponding to each bus scheduling alternative indicates how many times larger the obtained bus schedule is relative to the ideal length. The diagrams correspond to the average obtained for the 50 graphs considered for each configuration.

A first conclusion is that BSA_1 produces the shortest delays. This is not unexpected, since it is based on highly customized bus schedules. It can be noticed, however, that the approaches BSA_2 and BSA_3 are producing results that are close to those produced by BSA_1, but with a much lower cost in controller complexity. It is not surprising that BSA_4, which restricts very much the freedom for bus optimization, produces very low quality results.

The actual bus load is growing with the number of processors and, implicitly, that of simultaneously active tasks. Therefore, the delays at low bus load (smaller number of processors) are close to the ideal ones. The deviation from the ideal schedule length is growing with the increased bus load due to the inherent delays in bus access. This phenomenon is confirmed also by the comparison between the diagrams in Fig. 6.8(a) and (b). The diagrams in Fig. 6.8(b) were obtained considering a bus load that is 1.5 times higher (bus speed 1.5 times smaller) than in Fig. 6.8(a). It can be observed that the deviation of schedule delays from the ideal one is growing faster in Fig. 6.8(b).

The execution times for the whole flow, in the case of the largest examples (consisting of 200 tasks on 20 processors) are as follows: 125 min. for BSA_1, 117 min. for BSA_2, 47 min. for BSA_3, and 8 min. for BSA_1.

The amount of memory accesses relative to the computations has a strong influence on the worst case execution time. We have performed another set of experiments in order to asses this issue. The results are presented in Fig. 6.9. We have run experiments for configurations consisting of 2, 4, 6, ... 10 processors. For each configuration, 50 randomly generated task graphs were used. For each task graph, we have varied the ratio of clock cycles spent during computations and memory accesses. We have used eight different ratios: 5.0, 4.0, 3.0, 2.6, 2.2, 1.8, 1.4, 1.0. A ratio of 3.0 means that the number of clock cycles spent by the processors performing computations (assuming that all the memory accesses are cache hits) is three time higher than the number of cache misses multiplied with the cache miss penalty (assuming that each cache miss is handled in constant time, as if there are no conflicts on the bus). So, for example, if a task spends on the worst case CFG

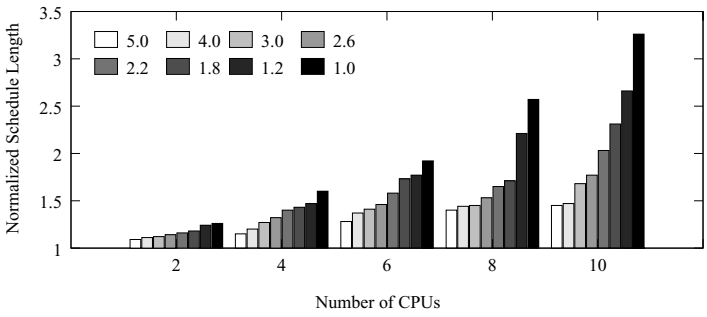


Figure 6.9: BSA3 with different amount of memory accesses

path 300000 clock cycles for computation and 100000 cycles for memory accesses due to cache misses (excluding the waiting time for bus access), the ratio will be 3.0. During this set of experiments we have assumed that the bus is scheduled according to the BSA_3 policy. Similar to the previous experiments from Fig. 6.8, the ideal schedule length is considered as the baseline for the diagrams presented in Fig. 6.9. Each bar indicates how many times larger the calculated worst case execution is relative to the ideal length. For example, on an a architecture with six processors and a ratio of 5.0, the worst case execution time is 1.28 times higher than the ideal one. Using the same architecture but with a smaller ratio (this means that the application is more memory intensive), the deviation increases: for a ratio of 3.0, the worst case execution time is 1.41 times the ideal one, while if the ratio is 1.0 the worst case execution time is 1.92 times higher than the ideal one.

In order to validate the real-world applicability of this approach we have analyzed a smart phone application. It consists of a GSM codec (encoder and decoder) [DB] and an MP3 decoder [Hag], that were mapped on 4 ARM7 processors (the GSM encoder and decoder are mapped each one on a processor, while the MP3 decoder is mapped on two processors). The software applications have been partitioned into 64 tasks. The size of one task is between 1304 and 70 lines of C code in case of the GSM codec and between 2035 and 200 lines in case of the MP3 decoder. We have assumed a 4-way set associative instruction cache with a size of 4KB and a direct mapped data cache of the same size. The results of the analysis are presented in table 6.1, where the deviation of the schedule length from the ideal one is presented for each bus scheduling approach.

BSA_1	BSA_2	BSA_3	BSA_4
1.17	1.33	1.31	1.62

Table 6.1: Results for the smart phone

Part IV

Conclusions and Future Work

Chapter 7

Conclusions

7.1 Offline Energy Minimization by Voltage Selection

Energy reduction techniques, such as supply voltage selection and adaptive body-biasing can be effectively exploited at the system-level. In Chapter 3, we have investigated different alternatives of the combined supply voltage selection, adaptive body-biasing and processor shutdown problems at the system-level. These include the consideration of transition overheads as well as the discretization of the supply and threshold voltage levels. We have shown that nonlinear programming and mixed integer linear programming formulations can be used to solve these problems. Further, the NP-hardness of the discrete voltage selection case was shown, and a heuristic to efficiently solve the problem has been proposed. Similarly, if the shutdown of processors is considered, the problem becomes NP complete, even in the continuous case. Therefore, we have proposed an efficient heuristic to solve this problem. The voltage selection technique achieves additional efficiency by simultaneously scaling the voltages of processors and communication. We have investigated two alternatives, considering both buses with repeaters and fat wires. Several generated benchmark examples as well as two real-life applications were used to show the applicability of the introduced approaches.

7.2 Quasi-Static Energy Minimization by Voltage Selection

In Chapter 5, we have presented a novel quasi-static voltage scaling technique for time-constraint applications that addresses in an efficient manner the problem of exploiting the dynamic slack. The method avoids an unnecessarily high run-time overhead by precomputing possible voltage scaling scenarios and storing the outcome in look-up tables. The avoided overheads can be turned into additional energy savings. Furthermore, we have addressed both dynamic and leakage power through supply and body-bias voltage scaling. We have shown that the proposed approach is superior to both static and dynamic approaches proposed so far in literature. Experiments conducted on numerous automatically generated examples and real-life benchmarks demostrate the quality of the proposed technique.

7.3 Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip

In Chapter 6, we have presented an approach for the implementation of predictable real-time applications on multiprocessor SoCs. The proposed algorithms take into consideration potential conflicts between parallel tasks that try to access concurrently the memory. The approach comprizes worst-case execution time estimation and bus access optimization in the global context of system level scheduling. Experiments carried out on synthetic benchmarks and a real-life application have shown the efficiency of the approach.

Chapter 8

Future Work

8.1 Energy Minimization

We have presented an offline and an online approach for dynamic and leakage energy minimization. Throughout Chapters 3-5, we assumed that the leakage power of the tasks depends only on their voltages. In reality, leakage depends strongly on the temperature [SSS+04, HLS04]. We are currently investigating the impact of the temperature on the results achieved by the voltage selection algorithms proposed in this thesis. The questions to be answered are: (1) what are the differences between the voltages calculated for each task if the temperature conditions change, and, (2) how to design a voltage selection algorithm that is able to capture the temperature variations.

8.2 Predictability

In Chapter 6, we have presented an approach for worst-case execution time analysis and system scheduling for real-time applications implemented on multiprocessor systems on chip. At the core of this approach is the TDMA organization of the bus, captured by the bus schedule. Several alternatives, trading off the complexity of the bus arbiter against the achievable worst-case execution times were discussed. Nevertheless, this work is only starting to show its potential. Further improvements related to the optimization of the bus access (new policies and better algorithms for the optimization of the bus schedule given a certain policy) are subject of future investigations. A first attempt in this direction is presented in [RAEP07].

An interesting aspect that is currently under investigation is the coexistence between real-time and non real-time applications on the same system. In this regard, we are extending the MPARM cycle accurate simulator [BBB+03, LPB04] with models of the strict bus arbiters proposed in Chapter 6. We plan to perform simulations of different mixes of real-time and non real-time applications. A more complex bus arbiter, that is capable of improving the average case execution time of non-real time applications and, in the same time enforces the worst-case guarantees of the real-time tasks is under study.

The voltage selection algorithms presented in the thesis have been designed without the consideration of the interactions between the processors and memory. In this regard, it is interesting to develop new voltage selection algorithms that work on hardware platforms similar to the one from Chapter 6. Moreover, since the memories contribute significantly to the power consumed by a system, algorithms that address the energy optimization globally (processors, bus, memories) at the system level are needed.

Appendix A

The complete discrete voltage selection with overheads MILP formulation

This appendix outlines in more detail the MILP formulation for the discrete voltage selection problem, i.e., it provides additional information that has been withhold from Section 3.7.3 due to clarity reasons. The complete formulation is given by: Minimize

$$\underbrace{\sum_{k=1}^{|\mathcal{T}|}\sum_{s\in\mathcal{M}}\sum_{m\in\mathcal{M}}\left(C_{eff_k}\cdot P_{dnom_m}\cdot t_{k,s,m}+P_{leak_m}\cdot t_{k,s,m}\right)}_{\text{Task energy dissipation}} + \underbrace{\sum_{k=1}^{|\mathcal{T}|}\sum_{s\in\mathcal{M}}\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{M}}\left(b_{k,s,i,j}\cdot EP_{i,j}\right)}_{\text{Transition energy overhead}} \tag{A.1}$$

subject to

$$\delta_k = \sum_{s\in\mathcal{M}^*}\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{M}} b_{k,s,i,j}\cdot DP_{i,j} \tag{A.2}$$

$$\delta_{k,l} = \sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{M}} b_{k,m,i,j}\cdot DP_{i,j} \quad \text{where } (k,l)\in\mathcal{E}^{\bullet} \tag{A.3}$$

$$D_k + \sum_{s\in\mathcal{M}} \sum_{m\in\mathcal{M}} t_{k,s,m} + \delta_k \leq dl_k \tag{A.4}$$

$$D_k + \sum_{s\in\mathcal{M}} \sum_{m\in\mathcal{M}} t_{k,s,m} + \delta_k + \delta_{pl,l} \leq D_l \quad \forall (k,l) \in \mathcal{E}, (pl,l) \in \mathcal{E}^\bullet \tag{A.5}$$

$$c_{k,s,i} = t_{k,s,i} \cdot f_i \quad k \text{ in } 1,...,|\mathcal{T}|,\ s \in \mathcal{M},\ i \in \mathcal{M},\ c \in \mathbb{N} \tag{A.6}$$

$$\sum_{s\in\mathcal{M}} \sum_{i\in\mathcal{M}} c_{k,s,i} = WNC_k \quad k \text{ in } 1,...,n \tag{A.7}$$

Up to this point the model corresponds to the formulation given in Section 3.7.3. The additional constraints that complete the formulation are:

$$\sum_{m\in\mathcal{M}} c_{k,s,m} \geq 1 \ \ \forall \tau_k,\ s \in \mathcal{M} \tag{A.8}$$

$$a_{k,s,m} \cdot WNC_k \geq c_{k,s,m} \quad k \in 1,...,|\mathcal{T}|,\ s \in \mathcal{M},\ m \in \mathcal{M} \tag{A.9}$$

$$\sum_{m\in\mathcal{M}} a_{k,s,m} = 1 \quad k \in 1,...,|\mathcal{T}|,\ s \in \mathcal{M} \tag{A.10}$$

$$\begin{cases} a_{k,s,i} &= \sum_{j=1}^{m} b_{k,s,i,j} \quad k \in 1,...,|\mathcal{T}|,\ s \text{ in } 1,...,|\mathcal{M}|-1,\ i \in \mathcal{M} \\ a_{k,s+1,j} &= \sum_{i=1}^{m} b_{k,s,i,j} \quad k \in 1,...,|\mathcal{T}|,\ s \text{ in } 2,...,|\mathcal{M}|,\ j \in \mathcal{M} \end{cases} \tag{A.11}$$

$\forall (k,l) \in E$

$$\begin{cases} a_{k,s,i} &= \sum_{j=1}^{m} b_{k,m,i,j} \quad s \in \mathcal{M},\ i \in \mathcal{M} \\ a_{l,1,j} &= \sum_{i=1}^{m} b_{k,m,i,j} \quad j \in \mathcal{M} \end{cases} \tag{A.12}$$

$\forall (k,l) \in E$

$$c_{k,s,m} \in \mathbb{N}, \quad a_{k,s,m},\ b_{k,s,i,j} \in \{0,1\} \quad k = 1..|\mathcal{T}|,\ s \in \mathcal{M},\ i \in \mathcal{M},\ j \in \mathcal{M} \tag{A.13}$$

In Section 3.7.3, we have briefly introduced the MILP model with the transition overheads. We detail now how we capture the mode variations in our MILP formulation. Please remember that in order to compute the corresponding delay and energy penalties, the concepts of subtasks and execution modes have been introduced in Section 3.7.3. Eq. (A.8) states that for each subtask τ_k^s of a task τ_k, at least one mode m is active (at least one clock cycle is executed by a subtask). This can be observed, for example, in Fig. 3.4(c).

Further, we introduced two sets of auxiliary variables: $a_{k,s,m}$ and $b_{k,m,i,j}$. The binary variables $a_{k,s,m}$ indicate, for a given task τ_k, the active mode m for each sub-

task τ_k^s. $a_{k,s,m}$ is 1 when $c_{k,s,m} \geq 1$ and 0 when $c_{k,s,m} = 0$. For instance, Fig. 3.4(d) gives the binary variables $a_{k,s,m}$ for the solution vector in Fig. 3.4(c). The other binary variables, $b_{k,s,i,j}$, are the instrument directly used to compute the penalties, both in terms of energy and delay. For all tasks τ_k, a mode change from subtask τ_k^s with mode i to subtask τ_k^{s+1} with mode j (s in $1,...,|\mathcal{M}|-1$) is expressed by $b_{k,s,i,j} = 1$. Otherwise, i.e., in the case of no mode change, $b_{k,s,i,j} = 0$. The binary variables $b_{k,s,i,j}$ are used in Equations (A.1), (A.2) and (A.3), the equations in which the energy and delay overheads are computed. Please note that Equations (A.8) to (A.13) are solely introduced to pinpoint where mode changes are situated.

Appendix B

The DDVS Problem is strongly NP-Hard

In this appendix we will prove that the discrete supply voltage scaling problem, in general, i.e., even V_{dd} scaling only and V_{bs} scaling only, is NP-hard.

Theorem 3 *The discrete supply and body-bias voltage scaling problem (DDVS) is NP-hard.*

In order to prove this theorem, we start with showing that the discrete supply voltage scaling problem (DDVddSnoOH) is NP-hard, even without the consideration of body-bias scaling and transition overheads.

Theorem 4 *The discrete supply voltage scaling problem (DDVddSnoOH) without transition overheads in terms of delay and energy overheads is NP-hard.*

The formulation for the DDVddSnoOH problem is give as: Consider a set of tasks with precedence constraints $\mathcal{T} = \{\tau_i\}$ which have been mapped and scheduled on a set of variable voltage processors. For each task τ_i its deadline dl_i, its number of clock cycles to be executed WNC_i and the switched capacitance C_{eff_i} are given. Each processor can vary its supply voltage V_{dd} within a set of discrete voltages $m_z = \{(V_{dd_z},)\}$. The power dissipation (dynamic) and the cycle time (processor speed) depend on the selected voltage (mode of operation). Tasks are executed cycle by cycle, and each cycle can potentially execute at a different voltage, i.e., at a different speed. Our goal is to find voltage assignments for each task such that the individual task deadlines are met and the dynamic energy consumption is

minimal. No transition overheads, in terms of energy or time is required when changing the voltage settings of the processors.

We prove the NP-hardness of the DDVddnoOH problem, by reducing the discrete time-cost trade-off problem (DTCT) to DDVddnoOH. Thus, solving the DDVddnoOH is as hard as solving the DTCT problem.

Theorem 5 *The discrete time-cost trade-off problem (DTCT) is NP-hard.*

The proof for theorem 5 is given in [DDGW97].

The formulation for the discrete time-cost trade-off problem is the following: Given a set of N tasks $\mathcal{T} = \{\tau_i\}$, $i = 1..N$ with precedence constraints. Each task has to execute within a given deadline, dl_i. Tasks can be executed with several speeds, from a given set $S = \{s_i\}$, $i = 1..M$. The choice of a certain speed has a corresponding cost $C = c_i$. A higher cost corresponds to a faster execution time. The task execution is not preemptive and its speed cannot be changed during execution. The goal is to find the speed assignment for each task such that the individual task deadlines are met and the total cost is minimal.

In the following we present an algorithm for reducing DTCT to a particular instance of the DDVddSnoOH problem. The input sets of tasks with precedence constraints are identical in the two problems. In the DDVddSnoOH problem there is a one-to-one relation between the supply voltage and the processor speed. For simplification we denote this as $s = f(V_{dd})$ (s is the speed corresponding to a particular choice of a supply voltage V_{dd}). We can express then the set of speeds from DTCT as $S = \{s_i\} = \{f(V_{dd_i})\}$. The dynamic energy consumption for a task τ_i executing WNC_i clock cycles with V_{dd_i} is $C_{eff_i} \cdot WNC_i \cdot V_{dd_i}^2$. We choose the set of costs for DTCT as $C = \{c_i\} = \{C_{eff_i} \cdot WNC_i \cdot f(V_{dd_i})\}$. We assume a particular instance of DDVddSnoOH such that $WNC_i = 1$, for all the tasks $\tau_i \in \mathcal{T}$. This implies that such tasks can only execute with one voltage, as a single clock cycle is not preemptable. It is easy to see that this transformation can be done in polynomial (linear) time. As DTCT problem is NP-hard, we conclude that DDVddSnoOH problem is also NP-hard. QED.

Appendix C

Shutdown Problem Complexity

C.1 The Knapsack Problem

The continuous multiple choice knapsack (*CMCK*1) is a well known NP complete problem, [GJ79].

Given a set of n items U, partitioned into m disjoint subsets, U_m. The size of the knapsack is B. Each item has a size s_i and a value v_i. Select from each subset U_j one item and assign to it a weight $r_j \in [0,1]$, such that the total size of the selected items is less than B and the total value is maximized.

$$\sum_{j=1}^{m} r_j \cdot s_j \leq B \tag{C.1}$$

$$\sum_{j=1}^{m} r_j \cdot v_j \geq K \tag{C.2}$$

It can be easily proven that the following instance (*CMCK*2) is a generalization of *CMCK*1 and thus another NP complete problem. Given a set of n items U, partitioned into m disjoint subsets, U_m. There are p knapsacks, with size B_p each. A relationship mapping each subset to several of the p knapsacks is also given (when selecting one item, it is placed in several knapsacks). Each item has a size s_i and a value v_i. Select from each subset U_j one item and assign to it a weight $r_j \in [0,1]$, such that the size of each knapsack is not exceeded and the total value (considering

each item that was placed in several knapsack only once) is maximized.

$$\sum_{j \in Kn_p} r_j \cdot s_j \leq B_p \tag{C.3}$$

$$\sum_{j=1}^{m} r_j \cdot v_j \geq K \tag{C.4}$$

C.2 The Shutdown Problem

Given a set of n tasks running on m processors. The task mapping on the processors and the execution order is known (precedence constraints on tasks mapped on different processors and scheduling order on each processor). Tasks have a certain fixed length and a deadline. Due to dependencies between tasks on different processors, there is a certain amount of idleness (time intervals when no task in running). The start time of each task is variable. When the processor is idle, it consumes energy but does not perform any useful computation. In order to save energy, during such an idle interval one processor could be shut down. A shutdown operation comes with a fixed time and energy penalty.

The goal is to compute, for each possible idle time, whether to shutdown a processor and for how long or to let it idle.

We will prove that deciding whether or not to shutdown in these conditions is in fact an NP complete problem.

In the following we formally state the problem. Each task τ_i has the execution time $wcet_i$ and it consumes the energy E_i. At the end of each τ_i, the processor can remain idle for t_{idle_i}, or it can be shut down start for t_{off_i} with the time penalty t_{soh}. Let us consider that each task τ_i has two versions, corresponding to the possible actions following its end: idle or shutdown. The size of these two versions are $wcet_i + t_{idle_i} \cdot r_i$ and $wcet_i + t_{soh} + t_{off_i} \cdot r_i$, where $r_i \in [0,1]$. t_{idle_i} and t_{off_i} are the maximum available time for idling or shutdown for task τ_i. It is important to note that the amount of time to be spent idling or off, after each task, is different, due to the dependencies between tasks. The associated energy consumptions for the two alternatives are: $E_i + P_{idle} \cdot t_{idle_i} \cdot r_i$ and $E_i + E_{soh} + P_{off} \cdot t_{off_i} \cdot r_i$. P_{idle} is the power consumption when the processor is idle. P_{off} is the power consumption when the processor is shutdown ($P_{off} << P_{idle}$). Due to the precedence constraints there are several paths in the task graph. The deadlines dl_π have to be met on all these paths, π.

$$\sum_{\tau_i \in \pi} (wcet_i + t_{idle_i} \cdot r_i \;\; or \;\; wcet_i + t_{soh} + t_{off_i} \cdot r_i) \leq dl_\pi \tag{C.5}$$

The total idle time has to be filled either by idle periods or by shutdowns.

$$\sum_{i=1}^{m}(t_{idle_i} \cdot r_i \ \ or \ \ t_{soh} + t_{off_i} \cdot r_i) \leq dl_\pi = T_{idle} \tag{C.6}$$

The objective is to minimize the amount of power consumed, ie. to maximize the shutdown intervals.

$$\sum_{\tau_i \in \pi}(E_i + P_{idle} \cdot t_{idle_i} \cdot r_i \ \ or \ \ E_i + E_{soh} + P_{off} \cdot t_{off_i} \cdot r_i) \tag{C.7}$$

We can reformulate slightly the problem by starting from the observation that for each task, the execution time and the energy consumption are constants that are only important when expressing the deadlines on the dependency paths of the task graph. Moreover, for each version of one task, $wcet_i$ and E_i are common. Eq. C.5 can be rewritten as:

$$\sum_{\tau_i \in \pi}(t_{idle_i} \cdot r_i \ \ or \ \ t_{soh} + t_{off_i} \cdot r_i) \leq dl_\pi - \sum_{\tau_i \in \pi} wnc_i \tag{C.8}$$

Let us consider that the two possibilities of execution after each task finishes are items of a subset. Consequently, there is such a subset for each task. The sizes of the items are: $s_i^1 = t_{idle_i}$ and $s_i^2 = t_{soh} + t_{off_i}$. Their values are $v_i^1 = P_{idle} * t_{idle_i}$ and respectively $v_i^2 = E_{soh} + P_{off} \cdot t_{off_i}$. We have to select from each subset one item and assign a weight $r_i \in [0,1]$ such that:

$$\sum_{\tau_i \in \pi}(t_{idle_i} \cdot r_i \ \ or \ \ t_{soh} + t_{off_i} \cdot r_i) \leq B_\pi \ \text{ for each task graph path } \pi \tag{C.9}$$

$$\sum_{i=1}^{m}(t_{idle_i} \cdot r_i \ \ or \ \ t_{soh} + t_{off_i} \cdot r_i) \leq dl_\pi = T_{idle} \tag{C.10}$$

The objective is to minimize the value:

$$\sum_{\tau_i \in \pi}(P_{idle} \cdot t_{idle_i} \cdot r_i \ \ or \ \ E_{soh} + P_{off} \cdot t_{off_i} \cdot r_i) \tag{C.11}$$

Clearly, *CMCK*2 is equivalent to this, and, thus, the shutdown problem is NP complete as well.

Appendix D

Continuous Online Interpolation

In this section we will show that the continuous frequencies calculated online in Section 5.5 will not lead to deadline misses in case when the tasks execute the worst-case number or clock cycles. Please note that we consider single processor systems. Throughout the proof, we will examine the frequency of a task, as a function of its start time, calculated in several iterations by the algorithm presented in Section 5.5.1.

If there is only one task in the system, then it will be the only one scaled from the start time to the deadline. If the start time *st* is a variable, (in the interval earliest start time to latest start time),

$$f(st) = \frac{NC}{dl - st}$$

It can be easily shown that $f(st)$ is a convex function.

Let us consider now the case when there are several tasks in the system, among which τ_i, the task under examination. There are several possibilities:

1) If τ_i consumes less energy then some of the other tasks. Let us focus on what happens when voltage scaling is performed using the algorithm from Section 5.5.1, for different start times, beginning from the latest start time LST_i. The outcome is illustrated in Fig. D.1, for start times in the interval $[t_2, LFT]$. Since the energy in the system is dominated by other tasks, τ_i is not scaled, even tough the amount of slack increases when the start time decreases. Thus, the frequency remains constant.

2) If the start time is decreased beyond t_2, the energy of τ_i is comparable with the energy of the other tasks. Thus, starting with t_2 for smaller start times, τ_i is scaled. Given a small slack increase (infinitesimal) δ, the voltage scaling algorithm calculates which of the tasks benefit the most from extending their execution time with δ, and, δ is divided between those tasks.

Lets assume that we have 3 arbitrary possible start times for the first task, s_1, s_2 and s_3 with $s_1 < s_2 < s_3$, $s_1 \geq t_1$. To each start time it corresponds an end time e_1, e_2, e_3, $e_1 \leq e_2 \leq e_3$. If the start time is s_3, then we have the tightest schedule and the frequency will be $f_3 = \frac{1}{e_3 - s_3}$. If the start time is $s_2 = s_3 - \delta$, there are n tasks that have the same energy, so they will be scaled simultaneously. The end time of τ_i is $e_2 = e_3 - \delta - \frac{\delta}{n}$. The corresponding frequency is $f_2 = \frac{1}{e_2 - s_2} = \frac{1}{e_3 - s_3 - \frac{\delta}{n}}$. At start time $s_1 = s_2 - \delta = s_3 - 2 \cdot \delta$, there will be m tasks with the same energy profile that are scaled simultaneously. The end time of one such task is $e_1 = e_2 - \delta + \frac{\delta}{m} = e_3 - 2 \cdot \delta + \frac{\delta}{n} + \frac{\delta}{m}$. The corresponding frequency is $f_1 = \frac{1}{e_3 - s_3 + \frac{\delta}{n} + \frac{\delta}{m}}$. f_1,f_2,f_3 are discrete values on a convex curve if

$$f_2 - f_1 \leq f_3 - f_2$$

. It can be shown that this relation holds as long as $m \geq n$.

3) If the start time is decreased beyond t_1, potentially, one of the tasks that was scaled in the previous iterations, cannot be further scaled, because it reached the lowest possible frequency. Thus the convexity condition $m \geq n$ does not hold. The extra slack gained this way is divided among the remaining tasks that are scaled more aggressively. This can potentially lead to a discontinuity of the frequency function, marking the intersection between two convex regions.

This demonstrates that frequency, considered as a function of the task start time, is piecewise convex. Let us examine now the online algorithm described in Section 5.5.2. Consider that t_a, t_b, t_x, t_y, t_z are consecutive entries in LUT$_i$. If, for example, the actual start time t_{s_i} is in the interval [t_a, t_b], then the frequency calculated by linearly interpolating f_a and f_b is larger then any other frequency corresponding to a start time from this interval. Similarly, if the actual start time is between t_x and t_y, the frequency can safely be interpolated.

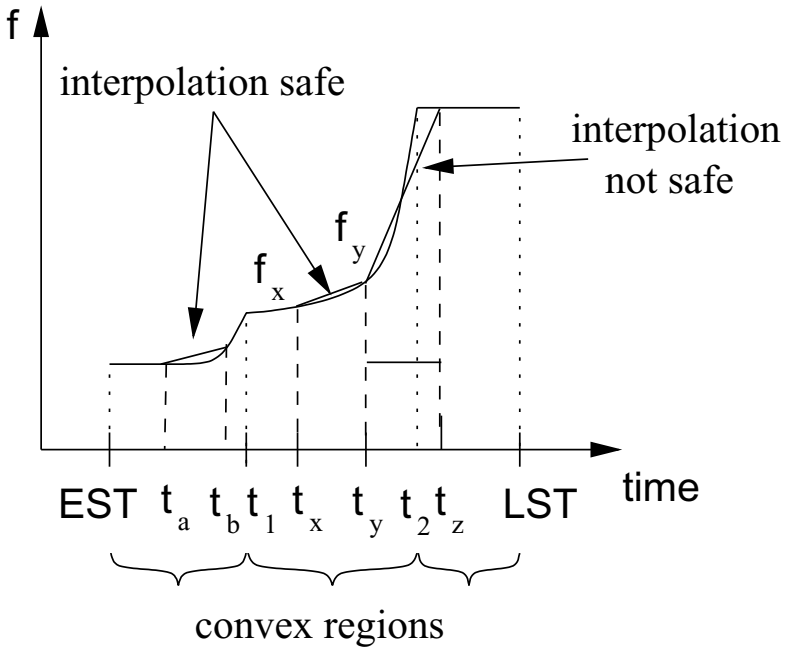


Figure D.1: Continuous interpolation

Appendix E

Quasi-Static Discrete Voltage Selection

In this section, we will prove the two theorems used during the discrete online voltage selection algorithm.

Theorem 6 *Given a task τ_k, its worst case number of clock cycles WNC_k and an available execution time t_{exe_k}. The task is executed on a voltage scalable processor having $|\mathcal{M}|$ modes. Assuming that the number of cycles to be assigned to each mode is not restricted to the integer domain, the energy is minimized when the task is using at most two execution modes.*□

We will give the proof of the theorem in the following. Let us denote with $c_i, i = 1..m$ the number of clock cycles executed in each of the m discrete modes. In mode i, the task is running with frequency f_i and the energy spent per clock cycle is e_i. The problem that must be solved online is the following linear optimization:
Minimize

$$E_k = \sum_{i=1}^{m} e_i \cdot c_i \tag{E.1}$$

Such that:

$$\sum_{i=1}^{m} c_i = WNC_k \text{ and } \sum_{i=1}^{m} \frac{c_i}{f_i} \leq t_{exe_k} \tag{E.2}$$

Based on the simplex algorithm, it can be mathematically proven that at most two of the $c_i, i = 1..m$ variables have non zero values. This is due to the fact that the linear formulation has two constrains and thus the solution vector will have two basic variables [Thi88]. Please note, that this is true only when the variables are

not restricted to be integers. If we add this restriction, then, potentially, a lower energy value can be obtained using more then two modes.

Theorem 7 *Given a task τ_k executing in the worst-case WNC_k clock cycles on a voltage scalable processor that has $|\mathcal{M}|$ modes. The frequency in mode m_i is f_i and the corresponding energy consumed per clock cycle is e_i, $i = 1..m$. The mode m_h with the highest frequency that is used by the τ_k is considered to be known. The number of cycles to be allocated for each mode is not restricted to the integer domain. The lower mode m_l that together with m_h achieves the minimal energy consumption has the following property:*

$$e_l \cdot (\frac{1}{f_j} - \frac{1}{f_h}) - e_j \cdot (\frac{1}{f_l} - \frac{1}{f_h}) < e_h(\frac{1}{f_j} - \frac{1}{f_l}), \forall j \in \mathcal{M} \tag{E.3}$$

The proof of this theorem is equivalent to showing that the energy consumed in modes (m_h,m_l) is lower than the energy consumed with any other mode pair (m_h,m_j):

$$e_l \cdot x_l + e_h \cdot x_h < e_j \cdot y_j + e_h \cdot y_h, \forall j \tag{E.4}$$

under the assumptions that the total execution times and the number of clock cycles in the two alternative executions (alternative 1: modes m_l and m_h are used; alternative 2: modes m_j and m_h are used) are identical. Let us note with x_l and x_h the numbers of clock cycles executed in the first alternative, and, with y_j and y_h the numbers of clock cycles executed in the second alternative.

$$x_l + x_h = y_j + y_h \quad (= WNC_k) \tag{E.5}$$

$$\frac{x_l}{f_l} + \frac{x_h}{f_h} = \frac{y_j}{f_j} + \frac{y_h}{f_h} \quad (= t_{exe_k}) \tag{E.6}$$

$$\tag{E.7}$$

The demonstration is based on rewriting the equations as:

$$x_h - y_h = y_j - x_l \tag{E.8}$$

$$\frac{x_l}{f_l} - \frac{y_j}{f_j} = \frac{y_h - x_h}{f_h} \quad (= t_{exe_k}) \tag{E.9}$$

$$\tag{E.10}$$

Let us consider this as a system of two equations with two unknowns (x_l and y_j). After solving the system, we get the following equations for x_l and y_j:

$$x_l = (x_h - y_h) \cdot \frac{\frac{1}{f_j} - \frac{1}{f_h}}{\frac{1}{f_l} - \frac{1}{f_j}} \tag{E.11}$$

$$y_j = (x_h - y_h) \cdot \frac{\frac{1}{f_l} - \frac{1}{f_h}}{\frac{1}{f_l} - \frac{1}{f_j}} \tag{E.12}$$

The relation E.4 that needs to be demonstrated can be rewritten as:

$$e_h \cdot (x_h - y_h) < e_j \cdot y_j - e_l \cdot x_l, \forall j \tag{E.13}$$

If we replace x_l and y_j with their equivalent from Eq. E.11, we need to demonstrate that:

$$e_h \cdot (x_h - y_h) < e_j \cdot (x_h - y_h) \cdot \frac{\frac{1}{f_l} - \frac{1}{f_h}}{\frac{1}{f_l} - \frac{1}{f_j}} - e_l \cdot (x_h - y_h) \cdot \frac{\frac{1}{f_j} - \frac{1}{f_h}}{\frac{1}{f_l} - \frac{1}{f_j}}, \forall j \tag{E.14}$$

An examination of this equation reveals its similarity to Eq. E.3. In order them to be equivalent, $(\frac{1}{f_l} - \frac{1}{f_j})/(x_h - y_h) \geq 0$. Let us examine two possibilities:

1)$f_l > f_j$, and, consequently, $\frac{1}{f_l} - \frac{1}{f_j} < 0$

In this case, $x_h > y_h$. This is so, because, given a task, the voltage selection algorithm assigns the maximum number of clock cycles possible, given the available execution time, to the mode with the lowest frequency. Thus, if $f_l > f_j$, more time can be spent in m_l (if the task is executed using (m_l,m_h)), then in m_j (if the task is executed using (m_l,m_h)). Consequently, less clock cycles are executed in m_h if the modes (m_l,m_h) are used the alternative execution in modes (m_j,m_h).

So, if $f_l > f_j$, then $\frac{1}{f_l} - \frac{1}{f_j} < 0$ and $x_h < y_h$.

2)$f_l < f_j$

Similar to the previous case, if $f_l < f_j$ then $\frac{1}{f_l} - \frac{1}{f_j} > 0$ and $x_h > y_h$, which finally proofs the theorem.

This theorem shows that for a given execution mode h, there can be only one corresponding mode l ($l < h$), that minimizes the energy, no matter how large is the available slack. The theorem also provides the means of calculating the pair, given the mode with the higher frequency, m_h. This calculation is performed offline, for each task τ_k and its result is the table $compatible\_modes_k$.

The algorithm used for the computation of the compatible mode pairs is given in Fig. E.1.

```
Algorithm:  QUASI_STATIC_VS__COMPAT_DISC_MODES
_____________________________________________________
Input:  - task τ_k
Output: - compatible mode pairs (h, compat_mode[h])
_____________________________________________________
01:  for h = |M| downto 1 {
02:    for i = h downto 2 {
03:      l=i
04:      not_pair=0
05:      for j = l-1 to 1
06:          if e_{k,l}·(1/f_j − 1/f_h) − e_{k,j}·(1/f_l − 1/f_h) > e_{k,h}(1/f_j − 1/f_l) {
07:            not_pair=1;break;
08:          }
09:      }
10:      if (not_pair==0) compat_mode[h]=l; break;
11:    }
12:}
13:  return compat_modes[]
```

Figure E.1: Pseudocode: Calculation of the Compatible Mode Pairs

Bibliography

[ACD74] T. Adam, K. Chandy, and J. Dickson. A Comparison of List Scheduling for Parallel Processing Systems. *J. Communications of the ACM*, 17(12):685–690, December 1974.

[AEP+07a] A. Andrei, P. Eles, Z. Peng, M. Schmitz, and B. M. Al-Hashimi. Energy Optimization of Multiprocessor Systems on Chip by Voltage Selection. *IEEE Transactions on Very Large Scale Integration Systems*, 15(3):262–275, March 2007.

[AEP+07b] A. Andrei, P. Eles, Z. Peng, M. Schmitz, and B. M. Al-Hashimi. Voltage selection for time-constrained multiprocessor systems on chip. In J. Henkel and S.Parameswaran, editors, *Designing Embedded Processors: A Low Power Perspective*, pages 259–282. Springer, 2007.

[AERP07] A. Andrei, P. Eles, J. Rosen, and Z. Peng. Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip. In *submitted*, 2007.

[AMMMA01] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proc. RTSS'01*, pages 95–105, 2001.

[AMR+06] Amit Agarwal, Saibal Mukhopadhyay, Arijit Raychowdhury, Kaushik Roy, and Chris H. Kim. Leakage power analysis and reduction for nanoscale circuits. *IEEE Micro*, 26(2):68–80, 2006.

[And06] A. Andrei. System Design of Embedded Systems Running on an MPSoC Platform. Technical report, Linkoping University, Department of Computer and Information Science, Sweden, January 2006.

[ASE+04a] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al-Hashimi. Overhead-Conscious Voltage Selection for Dynamic and Leakage Power Reduction of Time-Constraint Systems. In *Design, Automation and Test in Europe Conference*, pages 518–523, Feb 2004.

[ASE+04b] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al Hashimi. Simultaneous Communication and Processor Voltage Scaling for Dynamic and Leakage Energy Reduction in Time-Constrained Systems. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 362–369, Nov 2004.

[ASE+05a] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al-Hashimi. Overhead-Conscious Voltage Selection for Dynamic and Leakage Power Reduction of Time-Constraint Systems. *IEE Proceedings Computers & Digital Techniques, special issue with the best contributions from the DATE 2004*, 152(01):28–38, January 2005.

[ASE+05b] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. Al Hashimi. Quasi-Static Voltage Scaling for Energy Minimization with Time Constraints. In *Design, Automation and Test in Europe Conference*, pages 514–519, Nov 2005.

[ASEP04] A. Andrei, M. Schmitz, P. Eles, and Z. Peng. Simultaneous Communication and Processor Voltage Scaling for Dynamic and Leakage Energy Reduction in Time-Constrained Systems. In *Power-Aware Real Time Computing Workshop*, Sep 2004.

[BBB+03] Luca Benini, Davide Bertozzi, Davide Bruni, Nicola Drago, Franco Fummi, and Massimo Poncino. Systemc cosimulation and emulation of multiprocessor soc designs. *Computer*, 36(4):53–59, 2003.

[BD00] Luca Benini and Giovanni De Micheli. System-Level Power Optimizatin: Techniques and Tools. *ACM Transactions on Design Automation of Electronic Systems*, 5(2):115–192, April 2000.

[BGM+06] D. Bertozzi, A. Guerri, M. Milano, F. Poletti, and M. Ruggiero. Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip. In *Design, Automation and Test in Europe Conference*, pages 3–8, 2006.

[BJM97] Peter Bjørn-Jørgensen and Jan Madsen. Critical Path Driven Cosynthesis for Heterogeneous Target Architectures. In *Proc. International Workshop on Hardware/Software Codesign*, pages 15 – 19, 1997.

[BM02] K. Banerjee and A. Mehrotra. A Power-Optimal Repeater Insertion Methodology for Global Interconnects in Nanometer Designs. *IEEE Transactions on Electron Devices*, 49(11):2001–2006, No 2002.

[BMM+98] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano. Address bus encoding techniques for system-level power optimization. In *Design, Automation and Test in Europe Conference*, pages 861–867, 1998.

[Bor99] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, pages 23–29, July 1999.

[BTT98] Tobias Blickle, Jurgen Teich, and Lothar Thiele. System-level synthesis using evolutionary algorithms. *Design Automation for Embedded Systems*, 3(1):23–58, January 1998.

[CB95] A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publisher, 1995.

[CG72] E.G. Coffman and R.L. Graham. Optimal Scheduling for two processor systems. *Acta Inform.*, 1:200–213, 1972.

[CS02] P. Caputa and C. Svensson. Low-Power, Low-Latency Global Interconnects. In *Proc. IEEE ASIC/SOC'02*, pages 394–398, 2002.

[CTH05] Yu Ching Chang, King Ho Tam, and Lei He. Power-optimal repeater insertion considering vdd and vth as design freedoms. In *International Symposium on Low Power Electronics and Design*, pages 137–142, 2005.

[DB] Jutta Degener and Carsten Bormann. GSM 06.10 lossy speech compression. Source code available at http://kbs.cs.tu-berlin.de/∼jutta/toast.html.

[DDGW97] P. De, E. Dunne, J. Ghosh, and C. Wells. Complexity of the Discrete Time-Cost Tradeoff problem for Project Networks. *Operations Research*, 45(2):302–306, March 1997.

[DJ98] Robert P. Dick and Niraj K. Jha. MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-Synthesis of Distributed Embedded Systems. *IEEE Transactions on Computer-Aided Design*, 17(10):920–935, Oct 1998.

[DJ99] R. Dick and N. K. Jha. MOCSYN: Multiobjective core-based single-chip system synthesis. In *Design, Automation and Test in Europe Conference*, pages 263–270, March 1999.

[DVI+02] D. Duarte, N. Vijaykrishnan, M. Irwin, H. Kim, and G. McFarland. Impact of Scaling on The Effectiveness of Dynamic Power Reduction. In *Proc. ICCD*, Sept. 2002.

[EDPP00] Petru Eles, Alexa Doboli, Paul Pop, and Zebo Peng. Scheduling with Bus Access Optimization for Distributed Embedded Systems. *IEEE Transactions on VLSI Systems*, 8(5):472–491, Oct 2000.

[EES+03] J. Engblom, A. Ermedahl, M. Sjodin, J. Gustafsson, and H. Hansson. Worst-case execution-time analysis for embedded real-time systems. *International Journal on Software Tools for Technology Transfe r*, 4(4):437–455, 2003.

[ETZ00] Michael Eisenring, Lothar Thiele, and Eckart Zitzler. Conflicting criteria in embedded system design. *IEEE Design and Test of Computers*, 17(2):51–59, 2000.

[FA05] F.Poletti and A.Poggiali. Flexible Hardware/Software Support for Message Passing on a Distributed Shared Memory Architecture. In *Design Automation and Test in Europe*, pages 736–741, March 2005.

[FSS99] W. Fornaciari, D. Sciuto, and C. Silvano. Power Estimation for Architectural Exploration of HW/SW Communication on System-Level Buses. In *Proc. 7th Int. Workshop Hardware/Software Co-Design (CODES'99)*, pages 152–156, May 1999.

[GDR05] K. Goossens, J. Dielissen, and A. Radulescu. AEthereal Network on Chip: Concepts, Architectures, and Implementations. *IEEE Design & Test of Computers*, 2/3:115–127, 2005.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[GK01] F. Gruian and K. Kuchcinski. LEneS: Task Scheduling for Low-Energy Systems Using Variable Supply Voltage Processors. In *Proc. ASP-DAC'01*, pages 449–455, Jan 2001.

[GK03] F. Gruian and K. Kuchcinski. Uncertainty-Based Scheduling: Energy-Efficient Ordering for Tasks with Variable Execution Time. In *International Symposium on Low Power Electronics and Design*, pages 465–468, August 2003.

[Glo89] Fred Glover. Tabu search—part I. *ORSA Journal on Computing*, 1(3):190–206, Summer 1989.

[Glo90] Fred Glover. Tabu search– part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, 1989.

[Gro] Alchemy Research Group. Microlib. Available at http://www.microlib.org.

[Gru01] F. Gruian. Hard Real-Time Scheduling for Low-Energy Using Stoachastic Data and DVS Processors. In *International Symposium on Low Power Electronics and Design*, pages 46–51, August 2001.

[Gru02] F. Gruian. Energy-Centric Scheduling for Real-Time Systems. In *Phd Thesis*, 2002.

[Hag] Johan Hagman. mpeg3play-0.9.6. Source code available at http://home.swipnet.se/∼w-10694/tars/mpeg3play-0.9.6-x86.tar.gz.

[HAM+03] S. Hsu, A. Alvandpour, S. Mathew, S.-L. Lu, R. K. Krishnamurthy, and S. Borkar. A 4.5-ghz 130-nm 32-kb l0 cache with a leakage-tolerant self reverse-bias bitline scheme. *Journal of Solid State Circuits*, 38(5):755–761, May 2003.

[HASM+03] S. Hsu, A. Alvandpour, S. Lu S. Mathew, R. K. Krishnamurthy, and S. Borkar. A 4.5GHz 130nm 32kB L0 Cache With a Leakage-Tolerant Self Reverse-Bias Bitline Scheme. *IEEE Journal of Solid-State Circuits*, 38(5):755–761, May 2003.

[HLS04] Lei He, Weiping Liao, and Mircea R. Stan. System level leakage reduction considering the interdependence of temperature and leakage. In *Design Automation Conference*, pages 12–17, New York, NY, USA, 2004. ACM Press.

[HLTW03] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm. The influence of processor architecture on the design and the r esults of WCET tools. *Proceedings of the IEEE*, 91(7):1038–1054, 2003.

[HM03] J. Hu and R. Marculescu. Energy-aware mapping for tile-based NoC architectures under performance constraints. In *Proc. ASP-DAC'03*, pages 233 – 239, 2003.

[HO03] J. N. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, pages 33–60, 2003.

[HP02] C-T. Hsieh and M. Pedram. Architectural Energy Optimization by Bus Splitting. *IEEE Transactions on CAD*, 21(4):408–414, April 2002.

[HQPS98] Inki Hong, Gang Qu, Miodrag Potkonjak, and Mani B. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proc. Real-Time Systems Symposium*, 1998.

[IF99] Y. Ismail and E. Friedman. Repeater Insertion in RLC Lines for Minimum Propagation Delay. In *Proc. ISCAS'99*, pages 404–407, 1999.

[IHS98] M. Potkonjak I. Hong and M. B. Srivastava. On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processors. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 653–656, 1998.

[ITR] *The International Technology Roadmap for Semiconductors*. http://www.itrs.net.

[IY98] Tohru Ishihara and Hiroto Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proc. Int. Symp. Low Power Electronics and Design (ISLPED'98)*, pages 197–202, 1998.

[KA99] Yu-Kwong Kwok and Ishfaq Ahmad. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *ACM Computing Surveys*, 31(4):406–471, December 1999.

[KBP+06] I. A. Khatib, D. Bertozzi, F. Poletti, L. Benini, and et.all. A multiprocessor systems-on-chip for real-time biomedical monitoring and analysis: Architectural design space exploration. In *Design Automation Conference*, pages 125–131, 2006.

[KCS02] P. Kapur, G. Chandra, and K. Saraswat. Power Estimation in Global Interconnects and its Reduction using a Novel Repeater Optimiazation Methodology. In *Design Automation Conference*, 2002.

[KK05] W. Kwon and T. Kim. Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors. *ACM Transactions on Embedded Computing Systems*, February 2005.

[Kla00] Alexander Klaiber. The technology behind crusoe processors. *Transmeta Corporation*, January 2000. http://www.transmeta.com.

[Kop97] H. Kopetz. *Real-Time Systems-Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[KR02] C. Kim and K. Roy. Dynamic Vth Scaling Scheme for Active Leakage Power Reduction. In *Design, Automation and Test in Europe Conference*, pages 163–167, March 2002.

[LCB02] J. Liu, P. Chou, and N. Bagherzdeh. Communication Speed Selection for Embedded Systems with Networked Voltage-Scalable Proceossors. In *Proc. CODES'02*, 2002.

[LJ03] J. Luo and N. Jha. Power-profile Driven Variable Voltage Scaling for Heterogeneous Distributed Real-time Embedded Systems. In *Proc. VLSI'03*, 2003.

[LJ07] J. Luo and N.K. Jha. Power-Efficient Scheduling for Heterogeneous Distributed Real-Time Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(6):1161–1170, June 2007.

[LMW96] Y.T.S. Li, S. Malik, and A. Wolfe. Cache modeling for real-time software: Beyond direct mapped instruction caches. In *IEEE Real-Time Systems Symposium*, pages 254–263, 1996.

[LPB04] Mirko Loghi, Massimo Poncino, and Luca Benini. Cycle-accurate power analysis for multiprocessor systems-on-a-chip. In *ACM Great Lakes symposium on VLSI*, pages 410–406, New York, NY, USA, 2004. ACM Press.

[LS99] T. Lundqvist and P. Stenstrom. An Integrated Path and Timing Analysis Method based on Cycle-Level Symbolic Execution. *Real-Time Systems*, 17(2/3):183–207, 1999.

[LS04] Jacob R. Lorch and Alan Jay Smith. Pace: A new approach to dynamic voltage scaling. *IEEE Transactions on Computers*, 53(7):856–869, 2004.

[LTK04] L-F. Leung, C-Y. Tsui, and W-H. Ki. Minimizing Energy Consumption of Multiple-Processors-Core Systems with Simultaneous Task Allocation, Scheduling and Voltage Assignment. In *Asia South PacifiDesign Automation (ASP-DAC)*, pages 647–652, Jan 2004.

[LZS+06] Zhijian Lu, Yan Zhang, Mircea Stan, John Lach, and Kevin Skadron. Procrastinating voltage scheduling with discrete frequency sets. In *Design, Automation and Test in Europe Conference*, pages 456–461, 2006.

[MFMB02] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 721–725, 2002.

[MHQ02] B. Mochocki, X. Hu, and G. Quan. A Realistic Variable Voltage Scheduling Model for Real-Time Applications. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 726–731, 2002.

[MHQ07] Bren Mochocki, Xiaobo Sharon Hu, and Gang Quan. Transition-overhead-aware voltage scheduling for fixed-priority real-time systems. *ACM Transactions on Design Automation of Electronic Systems*, 12(2):11, 2007.

[MMP03] A. Macii, E. Macii, and M. Poncino. Improving the Efficiency of Memory Partitioning by Address Clustering. In *Design, Automation and Test in Europe Conference*, pages 18–22, March 2003.

[Moo65] Gordon Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, April 1965.

[MOS] Mosek optimization software. http://www.mosek.com.

[NN94] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Studies in Applied Mathematics, 1994.

[Ogn07] Jens Ogniewski. *Development and Optimization of an MPEG2 Video Decoder on a Multiprocessor Embedded Platform*. Master thesis LITH-IDA/DS-EX–07/006–SE, Linkoping University, 2007.

[OH96] Hyunok Oh and Soonhoi Ha. A Static Scheduling Heuristic for Heterogeneous Processors. In *2nd International EuroPar Conference Vol. II*, August 1996.

[OvG89] R. H. J. M. Otten and L. P. P. P. van Ginneken. *The annealing algorithm*. Kluwer Academic Publishers, 1989.

[PB00] P. Puschner and A. Burns. A Review of Worst-Case Execution-Time Analysis. *Real-Time Systems*, 2/3:115–127, 2000.

[PDBR04] S. Pasricha, N. Dutt, and M. Ben-Romdhane. Fast exploration of bus-based on-chip communication architectures. In *CODES+ISSS*, pages 242–247, 2004.

[PEPP06] P. Pop, P. Eles, Z. Peng, and T. Pop. Analysis and Optimization of Distributed Real-Time Embedded Systems. *ACM Transactions on Design Automation of Electronic Systems*, Vol. 11:593–625, 2006.

[PMT04] Daniel Gracia Perez, Gilles Mouchard, and Olivier Temam. Microlib: A case for the quantitative comparison of micro-architecture mechanisms. In *International Symposium on Microarchitecture*, 2004.

[pow00] Mobile AMD Athlon^{TM}4, Processor Model 6 CPGA Data Sheet, November 2000. Publication No 24319 Rev E.

[PP92] S. Prakash and A. Parker. SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems. *J. Parallel & Distributed Computing*, pages 338–351, Dec 1992.

[PPE+06] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the flexray communication protocol. In *Euromicro Conference on Real-Time Systems*, pages 203–216. IEEE Computer Society, 2006.

[QM03] W. Qin and S. Malik. Flexible and Formal Modeling of Microprocessors with Application to Retargetable Simulation. In *Design, Automation and Test in Europe Conference*, pages 556–561, March 2003.

[RAEP07] J. Rosen, A. Andrei, P. Eles, and Z. Peng. Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip. In *IEEE Real-Time Systems Symposium*, 2007.

[RE97] W. Ye R. Ernst. Embedded program timing alalysis based on path clustering and architecture classification. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 598–604, 1997.

[Ree93] C.R. Reevs. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publishers, 1993.

[RGA+06] M. Ruggiero, P. Gioia, G. Alessio, L. Benini, M. Milano, D. Bertozzi, and A. Andrei. A Cooperative, Accurate Solving Framework for Optimal Allocation, Scheduling and Frequency Selection on Energy-Efficient MPSoCs. In *International Symposium on Systems-on-Chip*, pages 1–4, Nov 2006.

[RJ05] R. Gupta R. Jejurikar. Dynamic Slack Reclamation with Procrastination Scheduling in Real-Time Embedded Systems. In *Design Automation Conference*, Jun 2005.

[RM05] H. Ramaprasad and F. Mueller. Bounding Preemption Delay within Data Cache Reference Patterns for Real-Time Tasks. In *Real-Time and Embedded Technology and Applications Symposium*, pages 71–80, 2005.

[SAH01] M. Schmitz and Bashir M. Al-Hashimi. Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems. In *Int. Symp. System Synthesis (ISSS'01)*, pages 250–255, October 2001.

[SAHE02] M. Schmitz, Bashir M. Al-Hashimi, and Petru Eles. Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems. In *Design, Automation and Test in Europe Conference*, pages 514–521, March 2002.

[SAHE04] M. Schmitz, B. Al-Hashimi, and P. Eles. *System-Level Design Techniques for Energy-Efficient Embedded Systems*. Kluwer Academic Publisher, 2004.

[SE06] J. Staschulat and R. Ernst. Worst case timing analysis of input dependent data cache behavior. In *Euromicro Conference on Real-Time Systems*, 2006.

[SHE05] M. Schmitz, B. Al Hashimi, and P. Eles. Cosynthesis of Energy-Efficient Multimode Embedded Systems With Consideration of Mode-Execution Probabilities. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(2):153–169, Feb 2005.

[SIE06] S. Schliecker, M. Ivers, and R. Ernst. Integrated analysis of communicating tasks in mpsocs. In *CODES+ISSS*, pages 288–293, 2006.

[SK01] D. Sylvester and K. Keutzer. Impact of Small Process Geometries on Microarchitectures in Systems on a Chip. *Proceedings of the IEEE*, 89(4):467–489, April 2001.

[SKC04] Jaewon Seo, Taewhan Kim, and Ki-Seok Chung. Profile-based Optimal Intra-Task Voltage Scheduling for Real-Time Applications. In *IEEE Design Automation Conference*, pages 87–92, June 2004.

[SKD05] Jaewon Seo, Taewhan Kim, and N. D. Dutt. Optimal integration of inter-task and intra-task dynamic voltage scaling techniques for hard real-time applications. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 450–455, 2005.

[SKL01] Dongkun Shin, Jihong Kim, and Seongsoo Lee. Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. *IEEE Design & Test of Computers*, pages 20–30, March–April 2001.

[SKL06] Jaewon Seo, Taewhan Kim, and Joonwon Lee. Optimal intratask dynamic voltage-scaling technique and its practical extensions. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 25(1):47–57, 2006.

[SL93] Gilbert C. Sih and Edward A. Lee. A Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions Parallel and Distributed Systems*, 4(2):175–187, February 1993.

[SLKK02] E. Salminen, V. Lahtinen, and T. Hamalainen K. Kuusilinna. Overview of bus-based system-on-chip interconnections. In *ISCAS*, pages 372–375, 2002.

[SPJ02] L. Shang, L. Peh, and N. Jha. Power-efficient Interconnection Networks: Dynamic Voltage Scaling with Links. *Comp. Arch. Letters*, 1(2):1–4, May 2002.

[SSE05] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *Euromicro Conference on Real-Time Systems*, pages 41–48, 2005.

[SSS+04] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions Architecture and Code Optimization*, 1(1):94–125, 2004.

[Sve01] C. Svensson. Optimum Voltage Swing on On-Chip and Off-Chip Interconnects. *IEEE Journal of Solid-State Circuits*, 36(7):1108–1112, July 2001.

[TFW00] H. Theiling, C. Ferdinand, and R. Wilhelm. Fast and Precise WCET Prediction by Separated Cache and Path Analysis. *Real-Time Systems*, 18(2/3):157–179, 2000.

[Thi88] Paul R. Thie. *An Introduction to Linear Programming and Game Theory*. John Wiley & Sons, 1988.

[TW04] L. Thiele and R. Wilhelm. Design for Timing Predictability. *Real-Time Systems*, 28(2/3):157–177, 2004.

[VJ03] K.S. Vallerio and N.K. Jha. Task graph extraction for embedded system synthesis. In *International Conference on VLSI Design*, pages 480–486, Jan 2003.

[VM03] G. Varatkar and R. Marculescu. Communication-Aware Task Scheduling and Voltage Selection for Total System Energy Minimization. In *IEEE/ACM International Conference on Computer-Aided Design*, 2003.

[WG90] M. Wu and D. Gajski. Hypertool: A Programming Aid for Message-passing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):330–343, July 1990.

[Wil99] H. P. Williams. *Model Building in Mathematical Programming*. Wiley, 1999.

[WKL+00] G. Wei, J. Kim, D. Liu, S. Sidiropoulos, and M. Horowitz. A Variable-Frequency Parallel I/O Interface with Adaptive Power-Supply Regulation. *IEEE J. Solid-State Circuits*, 35(11):1600–1610, Nov 2000.

[Wol05] W. Wolf. *Computers as Components: Principles of Embedded Computing System Design*. Morgan Kaufman Publishers, 2005.

[WSE02] F. Wolf, J. Staschulat, and R. Ernst. Associative caches in formal software timing analysis. In *Design Automation Conference*, pages 622–627, 2002.

[xsc00] Intel® XScale™ Core, Developer's Manual, December 2000.

[YC03] P. Yang and F. Catthoor. Pareto-Optimization-Based Run-Time Task Scheduling for Embedded Systems. In *Proc. CODES+ISSS '03*, pages 120–125, 2003.

[YDS95] F. Yao, A. Demers, and S. Shenker. A Scheduling Model for Reduced CPU Energy. *IEEE FOCS*, 1995.

[YLJ03] L. Yan, J. Luo, and N. Jha. Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-time Embedded Systems. In *IEEE/ACM International Conference on Computer-Aided Design*, 2003.

[YLJ05] L. Yan, J. Luo, and N. Jha. Joint dynamic voltage scaling and adpative body biasing for heterogeneous distributed real-time embedded systems,. *IEEE Transactions on Computer-Aided Design*, July 2005.

[ZHC02] Y. Zhang, X. Hu, and D. Chen. Task Scheduling and Voltage Selection for Energy Minimization. In *IEEE Design Automation Conference*, June 2002.

[ZHC03] Y. Zhang, X. Hu, and D. Chen. Energy Minimization of Real-time Tasks on Variable Voltage Processors with Transition Energy Overhead. In *Proc. ASP-DAC'03*, pages 65–70, 2003.

[ZLL+05] Yan Zhang, Zhijian Lu, John Lach, Kevin Skadron, and Mircea R. Stan. Optimal procrastinating voltage scheduling for hard real-time systems. In *Design Automation Conference*, pages 905–908, 2005.

[ZM04] Y. Zhu and F. Mueller. Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 84–93, October 2004.

[ZM05] Y. Zhu and F. Mueller. Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling. *Real-Time Systems*, 31(1-3):33–63, December 2005.

Department of Computer and Information Science
Linköpings universitet

Dissertations

Linköping Studies in Science and Technology

No 14 **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977, ISBN 91-7372-144-1.

No 17 **Bengt Magnhagen:** Probability Based Verification of Time Margins in Digital Designs, 1977, ISBN 91-7372-157-3.

No 18 **Mats Cedwall:** Semantisk analys av process-beskrivningar i naturligt språk, 1977, ISBN 91-7372-168-9.

No 22 **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978, ISBN 91-7372-188-3.

No 33 **Tore Risch:** Compilation of Multiple File Queries in a Meta-Database System 1978, ISBN 91-7372-232-4.

No 51 **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980, ISBN 91-7372-387-8.

No 54 **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980, ISBN 91-7372-404-1.

No 55 **Pär Emanuelson:** Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation, 1980, ISBN 91-7372-403-3.

No 58 **Bengt Johnsson, Bertil Andersson:** The Human-Computer Interface in Commercial Systems, 1981, ISBN 91-7372-414-9.

No 69 **H. Jan Komorowski:** A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, 1981, ISBN 91-7372-479-3.

No 71 **René Reboh:** Knowledge Engineering Techniques and Tools for Expert Systems, 1981, ISBN 91-7372-489-0.

No 77 **Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982, ISBN 91-7372-527-7.

No 94 **Hans Lunell:** Code Generator Writing Systems, 1983, ISBN 91-7372-652-4.

No 97 **Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983, ISBN 91-7372-660-5.

No 109 **Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation,1984, ISBN 91-7372-801-2.

No 111 **Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984, ISBN 91-7372-805-5.

No 155 **Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons, 1987, ISBN 91-7870-133-3.

No 165 **James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning, 1987, ISBN 91-7870-183-X.

No 170 **Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems, 1987, ISBN 91-7870-225-9.

No 174 **Johan Fagerström:** A Paradigm and System for Design of Distributed Systems, 1988, ISBN 91-7870-301-8.

No 192 **Dimiter Driankov***:* Towards a Many Valued Logic of Quantified Belief, 1988, ISBN 91-7870-374-3.

No 213 **Lin Padgham:** Non-Monotonic Inheritance for an Object Oriented Knowledge Base, 1989, ISBN 91-7870-485-5.

No 214 **Tony Larsson:** A Formal Hardware Description and Verification Method, 1989, ISBN 91-7870-517-7.

No 221 **Michael Reinfrank:** Fundamentals and Logical Foundations of Truth Maintenance, 1989, ISBN 91-7870-546-0.

No 239 **Jonas Löwgren:** Knowledge-Based Design Support and Discourse Management in User Interface Management Systems, 1991, ISBN 91-7870-720-X.

No 244 **Henrik Eriksson:** Meta-Tool Support for Knowledge Acquisition, 1991, ISBN 91-7870-746-3.

No 252 **Peter Eklund:** An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies,1991, ISBN 91-7870-784-6.

No 258 **Patrick Doherty:** NML3 - A Non-Monotonic Formalism with Explicit Defaults, 1991, ISBN 91-7870-816-8.

No 260 **Nahid Shahmehri:** Generalized Algorithmic Debugging, 1991, ISBN 91-7870-828-1.

No 264 **Nils Dahlbäck:** Representation of Discourse-Cognitive and Computational Aspects, 1992, ISBN 91-7870-850-8.

No 265 **Ulf Nilsson:** Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs, 1992, ISBN 91-7870-858-3.

No 270 **Ralph Rönnquist:** Theory and Practice of Tense-bound Object References, 1992, ISBN 91-7870-873-7.

No 273 **Björn Fjellborg:** Pipeline Extraction for VLSI Data Path Synthesis, 1992, ISBN 91-7870-880-X.

No 276 **Staffan Bonnier:** A Formal Basis for Horn Clause Logic with External Polymorphic Functions, 1992, ISBN 91-7870-896-6.

No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.

No 281 **Christer Bäckström:** Computational Complexity

of Reasoning about Plans, 1992, ISBN 91-7870-979-2.

No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.

No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.

No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.

No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.

No 338 **Simin Nadjm-Tehrani**: Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.

No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.

No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.

No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.

No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.

No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.

No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.

No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.

No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.

No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.

No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.

No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.

No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.

No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.

No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.

No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms,1997, ISBN 91-7871-857-0.

No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.

No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.

No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.

No 485 **Göran Forslund**: Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.

No 494 **Martin Sköld**: Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.

No 495 **Hans Olsén**: Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.

No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.

No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.

No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Langugaes from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.

No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.

No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.

No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.

No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.

No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-x.

No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis,1998, ISBN 91-7219-369-7.

No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.

No 563 **Eva L Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.

No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.

No 582 **Vanja Josifovski:** Design, Implementation and

Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.

No 589 **Rita Kovordányi**: Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.

No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.

No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.

No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.

No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.

No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.

No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.

No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.

No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.

No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.

No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.

No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.

No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.

No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.

No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.

No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.

No 688 **Marcus Bjäreland:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.

No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.

No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN-91-7373-126-9.

No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91 7373 207 9.

No 725 **Tim Heyer**: Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91 7373 208 7.

No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91 7373 212 5.

No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91 7373 258 3.

No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.

No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.

No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.

No 747 **Anneli Hagdahl:** Development of IT-suppor-ted Inter-organisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.

No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.

No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.

No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.

No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.

No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.

No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informa-

tionsystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing - An Emperical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004. ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linquistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modellling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.

No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.

No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.

No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN: 91-85297-97-6.

No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.

No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.

No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005. ISBN 91-85457-07-8.

No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.

No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.

No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.

No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.

No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.

No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.

No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.

No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.

No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.

No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.

No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.

No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8.

No 1018 **Ioan Chisalita:** Communication and Networking Techniques for Traffic Safety Systems, 2006, ISBN 91-85523-77-1.

No 1019 **Tarja Susi:** The Puzzle of Social Activity - The Significance of Tools in Cognition and Cooperation, 2006, ISBN 91-85523-71-2.

No 1021 **Andrzej Bednarski:** Integrated Optimal Code Generation for Digital Signal Processors, 2006, ISBN 91-85523-69-0.

No 1022 **Peter Aronsson:** Automatic Parallelization of Equation-Based Simulation Programs, 2006, ISBN 91-85523-68-2.

No 1023 **Sonia Sangari:** Some Visual Correlates to Focal Accent in Swedish, 2006, ISBN 91-85523-67-4.

No 1030 **Robert Nilsson:** A Mutation-based Framework for Automated Testing of Timeliness, 2006, ISBN 91-85523-35-6.

No 1034 **Jon Edvardsson:** Techniques for Automatic Generation of Tests from Programs and Specifications, 2006, ISBN 91-85523-31-3.

No 1035 **Vaida Jakoniene:** Integration of Biological Data, 2006, ISBN 91-85523-28-3.

No 1045 **Genevieve Gorrell:** Generalized Hebbian Algorithms for Dimensionality Reduction in Natural Language Processing, 2006, ISBN 91-85643-88-2.

No 1051 **Yu-Hsing Huang:** Having a New Pair of Glasses - Applying Systemic Accident Models on Road Safety, 2006, ISBN 91-85643-64-5.

No 1054 **Åsa Hedenskog:** Perceive those things which cannot be seen - A Cognitive Systems Engineering perspective on requirements management, 2006, ISBN 91-85643-57-2.

No 1061 **Cécile Åberg:** An Evaluation Platform for Semantic Web Technology, 2007, ISBN 91-85643-31-9.

No 1073 **Mats Grindal:** Handling Combinatorial Explosion in Software Testing, 2007, ISBN 978-91-85715-74-9.

No 1075 **Almut Herzog:** Usable Security Policies for Runtime Environments, 2007, ISBN 978-91-85715-65-7.

No 1079 **Magnus Wahlström:** Algorithms, measures, and upper bounds for satisfiability and related problems, 2007, ISBN 978-91-85715-55-8.

No 1083 **Jesper Andersson:** Dynamic Software Architectures, 2007, ISBN 978-91-85715-46-6.

No 1086 **Ulf Johansson:** Obtaining Accurate and Comprehensible Data Mining Models - An Evolutionary Approach, 2007, ISBN 978-91-85715-34-3.

No 1089 **Traian Pop:** Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, 2007, ISBN 978-91-85715-27-5.

No 1091 **Gustav Nordh:** Complexity Dichotomies for CSP-related Problems, 2007, ISBN 978-91-85715-20-6.

No 1106 **Per Ola Kristensson:** Discrete and Continuous Shape Writing for Text Entry and Control, 2007, ISBN 978-91-85831-77-7.

No 1110 **He Tan:** Aligning Biomedical Ontologies, 2007, ISBN 978-91-85831-56-2.

No 1112 **Jessica Lindblom:** Minding the body - Interacting socially through embodied action, 2007, ISBN 978-91-85831-48-7.

No 1113 **Pontus Wärnestål:** Dialogue Behavior Management in Conversational Recommender Systems, 2007, ISBN 978-91-85831-47-0.

No 1120 **Thomas Gustafsson:** Management of Real-Time Data Consistency and Transient Overloads in Embedded Systems, 2007, ISBN 978-91-85831-33-3.

No 1127 **Alexandru Andrei:** Energy Efficient and Predictable Design of Real-time Emdedded Systems, 2007, ISBN 978-91-85831-06-7.

Linköping Studies in Information Science

No 1 **Karin Axelsson:** Metodisk systemstrukturering- att skapa samstämmighet mellan informa-tionssystemarkitektur och verksamhet, 1998. ISBN-9172-19-296-8.

No 2 **Stefan Cronholm:** Metodverktyg och användbarhet - en studie av datorstödd metodbaserad systemutveckling, 1998. ISBN-9172-19-299-2.

No 3 **Anders Avdic:** Användare och utvecklare - om anveckling med kalkylprogram, 1999. ISBN-91-7219-606-8.

No 4 **Owen Eriksson:** Kommunikationskvalitet hos informationssystem och affärsprocesser, 2000. ISBN 91-7219-811-7.

No 5 **Mikael Lind:** Från system till process - kriterier för processbestämning vid verksamhetsanalys, 2001, ISBN 91-7373-067-X

No 6 **Ulf Melin:** Koordination och informationssystem i företag och nätverk, 2002, ISBN 91-7373-278-8.

No 7 **Pär J. Ågerfalk:** Information Systems Actability - Understanding Information Technology as a Tool for Business Action and Communication, 2003, ISBN 91-7373-628-7.

No 8 **Ulf Seigerroth:** Att förstå och förändra systemutvecklingsverksamheter - en taxonomi för metautveckling, 2003, ISBN91-7373-736-4.

No 9 **Karin Hedström:** Spår av datoriseringens värden - Effekter av IT i äldreomsorg, 2004, ISBN 91-7373-963-4.

No 10 **Ewa Braf:** Knowledge Demanded for Action - Studies on Knowledge Mediation in Organisations, 2004, ISBN 91-85295-47-7.

No 11 **Fredrik Karlsson:** Method Configuration - method and computerized tool support, 2005, ISBN 91-85297-48-8.

No 12 **Malin Nordström:** Styrbar systemförvaltning - Att organisera systemförvaltningsverksamhet med hjälp av effektiva förvaltningsobjekt, 2005, ISBN 91-85297-60-7.

No 13 **Stefan Holgersson:** Yrke: POLIS - Yrkeskunskap, motivation, IT-system och andra förutsättningar för polisarbete, 2005, ISBN 91-85299-43-X.

No 14 **Benneth Christiansson, Marie-Therese Christiansson:** Mötet mellan process och komponent - mot ett ramverk för en verksamhetsnära kravspeci-

fikation vid anskaffning av komponentbaserade informationssystem, 2006, ISBN 91-85643-22-X.