

SYSTEM-LEVEL DESIGN TECHNIQUES FOR ENERGY-EFFICIENT EMBEDDED SYSTEMS

System-Level Design Techniques for Energy-Efficient Embedded Systems

by

MARCUS T. SCHMITZ

University of Southampton, United Kingdom

BASHIR M. AL-HASHIMI

University of Southampton, United Kingdom

and

PETRU ELES

Linköping University, Sweden

KLUWER ACADEMIC PUBLISHERS

NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW

eBook ISBN: 0-306-48736-5
Print ISBN: 1-4020-7750-5

©2005 Springer Science + Business Media, Inc.

Print ©2004 Kluwer Academic Publishers
Dordrecht

All rights reserved

No part of this eBook may be reproduced or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without written consent from the Publisher

Created in the United States of America

Visit Springer's eBookstore at:
and the Springer Global Website Online at:

<http://ebooks.kluweronline.com>
<http://www.springeronline.com>

To our beloved families

Contents

List of Figures	ix
List of Tables	xiii
Preface	xv
Acknowledgments	xvii
1. INTRODUCTION	1
1.1 Embedded System Design Flow	2
1.2 System Specification	5
1.3 Co-Synthesis	6
1.4 Hardware and Software Synthesis	14
1.5 Book Overview	17
2. BACKGROUND	19
2.1 Energy Dissipation of Processing Elements	19
2.2 Energy Minimisation Techniques	24
2.3 Energy Dissipation of Communication Links	29
2.4 Further Readings	30
2.5 Concluding Remarks	33
3. POWER VARIATION-DRIVEN DYNAMIC VOLTAGE SCALING	35
3.1 Motivation	36
3.2 Algorithms for Dynamic Voltage Scaling	44
3.3 Experimental Results: Energy-Gradient based Dynamic Voltage Scaling	50
3.4 Concluding Remarks	58
4. OPTIMISATION OF MAPPING AND SCHEDULING FOR DYNAMIC VOLTAGE SCALING	61

4.1	Schedule Optimisation	62
4.2	Optimisation of Task and Communication Mapping	81
4.3	Optimisation of Allocation	94
4.4	Concluding Remarks	97
5.	ENERGY-EFFICIENT MULTI-MODE EMBEDDED SYSTEMS	99
5.1	Preliminaries	100
5.2	Motivational Examples	104
5.3	Previous Work	107
5.4	Problem Formulation	109
5.5	Co-Synthesis of Energy-Efficient Multi-Mode Systems	111
5.6	Experimental Results: Multi-Mode	122
5.7	Concluding Remarks	130
6.	DYNAMIC VOLTAGE SCALING FOR CONTROL FLOW-INTENSIVE APPLICATIONS	133
	by Dong Wu, Bashir M. Al-Hashimi, and Petru Eles	
6.1	The Conditional Task Graph Model	133
6.2	Schedule Table for CTGs	135
6.3	Dynamic Voltage Scaling for CTGs	136
6.4	Voltage Scaling Technique for CTGs	139
6.5	Conclusions	148
7.	LOPOCOS: A LOW POWER CO-SYNTHESIS TOOL	151
7.1	Smart Phone Description	151
7.2	LOPOCOS	157
7.3	Concluding Remarks	172
8.	CONCLUSION	173
8.1	Summary	174
8.2	Future Directions	177
	References	181
	Index	193

List of Figures

1.1	Example of a typical embedded system (smart-phone)	2
1.2	Typical design flow of a new embedded computing system	4
1.3	MP3 decoder given as (a) task graph specification (17 tasks and 18 communications) and (b) high-level language description in C	7
1.4	System-level co-synthesis flow	8
1.5	Architectural selection problem	9
1.6	Application mapping onto hardware and software components	10
1.7	Two different scheduling variants based on the same allocated architecture and identical application mapping	12
1.8	System schedule with idle and slack times	13
1.9	The concept of dynamic voltage scaling	14
1.10	Hardware synthesis flow	15
1.11	Software synthesis flow	16
2.1	Dynamic power dissipation of an inverter circuit [37]	20
2.2	Supply voltage dependent circuit delay	22
2.3	Energy versus delay function using fixed and dynamic supply voltages (considering $V_{max} = 3.3V$ and $V_t = 0.8V$)	24
2.4	Block diagram of DVS-enabled processor [36]	25
2.5	Shutdown during idle times (DPM)	27
2.6	Voltage scaling to exploit the slack time (DVS)	28
2.7	Combination of dynamic voltage scaling and dynamic power management	28
3.1	Architecture and specification for the motivational example	37

3.2	Power profile of a possible mapping and schedule at nominal supply voltage (no DVS is applied)	39
3.3	Two different voltage scaled schedules	41
3.4	Pseudo code of the proposed heuristic (PV-DVS) algorithm	45
3.5	Capturing the mapping and schedule information into the task graph by using pseudo edges and communication task	46
3.6	Pseudo code of task graph to mapped-and-scheduled task graph transformation	47
3.7	Three identical execution orders of the <code>tgff17_m</code> benchmark: (a) unscaled execution at nominal supply voltage (NO-DVS), (b) using the EVEN-DVS, and (c) the PV-DVS approach	54
3.8	Energy reduction quality dependent on minimal extension time Δt_{min}	56
3.9	Execution time dependent on minimal extension time Δt_{min}	56
3.10	Energy reduction quality dependent on execution time	57
4.1	Co-synthesis flow for the optimisation of scheduling and mapping towards the utilisation of PV-DVS	62
4.2	Specification and DVS-enabled architecture	63
4.3	A possible schedule <i>not</i> optimised for DVS	64
4.4	Schedule optimised for DVS considering the power variation model	65
4.5	List scheduling	70
4.6	Task priority encoding into a priority string	70
4.7	Principle behind the genetic list scheduling algorithm	71
4.8	Proposed EE-GLSA approach for energy-efficient schedules	73
4.9	Hole filling problem	74
4.10	Task mapping string describing the mapping of five tasks to an architecture	82
4.11	Proposed EE-GTMA approach for energy-efficient task mappings	83
4.12	Combined optimisation of task and communication mapping	85
4.13	A combined priority and communication mapping string	86
4.14	Proposed EE-GLSCMA approach for combined optimisation of energy-efficient schedules and communication mappings	88
4.15	Three scheduling and mapping concepts	90

4.16	Nine different implementation possibilities of the OFD algorithm	97
5.1	Example operational mode state machine of a smart phone	101
5.2	Relation between OMSM and individual task graph specifications	102
5.3	Distributed Architectural Model	103
5.4	Mode execution probabilities	105
5.5	Multiple task type implementations	107
5.6	Typical Activation Profile of a Mobile Phone	112
5.7	Task mapping string for multi-mode systems	113
5.8	Pseudo Code: Multi-Mode Co-Synthesis	114
5.9	Pseudo Code: Mapping Modification towards component shutdown	117
5.10	DVS Transformation for HW Cores	119
5.11	DVS Transformation for HW Cores considering inter-PE communication	120
5.12	Pseudo code: Task graph transformation for DVS-enabled hardware cores	121
5.13	Pareto optimal solution space achieved through a single optimisation run of <code>mul115</code> (without DVS), revealing the solution trade-offs between energy dissipation and area usage	125
5.14	A system specification consisting of two operational modes optimized for three different execution probabilities (solid line–0.1:0.9, dashed–0.9:0.1, dotted–0.5:0.5)	127
5.15	Energy dissipation of the Smart phone using different optimisation strategies	130
6.1	Conditional Task Graph and its Tracks	134
6.2	Schedules of the CTG of Figure 6.1 (a) (in this figure t_i corresponds to tasks τ_i)	137
6.3	Schedules scaled for energy minimisation	138
6.4	Improper scaling with violated timing constraint	138
6.5	CTG with one disjunction node	140
6.6	Schedules	141
6.7	Pseudo-code: Voltage scaling approach for CTGs	142
6.8	Actual, scaled schedules	145
7.1	Block diagram of the GSM RPE-LTP transcoder [73]	152
7.2	Task graph of the GSM voice encoder	154
7.3	Task graph of the GSM voice decoder	155

7.4	Block diagram of the MPEG-1 layer 3 audio decoder	156
7.5	Block diagram of the JPEG encoder and decoder [149]	156
7.6	Task graphs of the JPEG encoder and decoder	157
7.7	Design flow used within LOPOCOS	159
7.8	File description of the top-level finite state of the smart phone	160
7.9	File description of a single mode task graph	162
7.10	Technology library file	163
7.11	Co-synthesis results of Architecture 1	168
7.12	Co-synthesis results of Architectures 2 and 3	169
7.13	Co-synthesis results for Architectures 2 and 3, exploiting DVS	170
7.14	Co-synthesis results for Architecture 4	171
8.1	Network-on-Chip	179

List of Tables

1.1	Trade-offs between serveral heterogeneous components (+ + \equiv highly advantageous, + \equiv advantageous, o \equiv moderate, - \equiv disadvantageous, - - \equiv highly disadvantageous)	9
1.2	Task execution properties (time and power) on different processing elements	10
3.1	Nominal task execution times and power dissipations	38
3.2	Communication times and power dissipations of communication activities mapped to the bus	38
3.3	Evolution of the energy-gradients during voltage scaling	42
3.4	Comparison of the presented PV-DVS optimisation with the fixed power model using EVEN-DVS approach	52
3.5	PV-DVS results using the benchmarks of Bambha <i>et al.</i> [20]	53
4.1	Nominal execution times and power dissipations for the mapped tasks	63
4.2	Experimental results obtained using the <i>fixed power model</i> and the <i>power variation model</i> during voltage selection; both integrated into a genetic list scheduling algorithm	77
4.3	Experimental results obtained using the generalised, DVS optimised scheduling approach for benchmark example TG1	79
4.4	Experimental results obtained using the generalised, DVS optimised scheduling approach for benchmark example TG2	80
4.5	Mapping optimisation with and without DVS optimised scheduling using <code>tgff</code> and <code>hou</code> benchmarks	90

4.6	Mapping optimisation of the benchmark set TG1 using NO-DVS (Nominal), EVEN-DVS, and PV-DVS	92
4.7	Comparison between DLS algorithm and the proposed scheduling and mapping approach using Bambha's benchmarks [20]	93
4.8	Increasing architectural parallelism to allow voltage scaling of the OFD algorithm	96
4.9	Relaxing the performance constraints of the OFD algorithm	96
5.1	Task execution and implementation properties	105
5.2	Considering mode execution probabilities (excluding DVS)	123
5.3	Considering mode execution probabilities (including DVS)	125
5.4	Smart phone experiments without DVS	128
5.5	Smart phone experiments with DVS	129
6.1	Example Schedule Table for the CTG of Figure 6.1(a)	135
6.2	Schedule Table for the CTG of Figure 6.5	140
6.3	Scaled Schedule Table for the CTG of Figure 6.5	141
6.4	Pre-processed schedule table	142
6.5	Result after processing column <i>true</i> (values are rounded)	144
6.6	Results after processing column A	144
6.7	Final schedule table (scaled)	144
6.8	Results of the real-life example	146
6.9	Results of the generated examples	147
6.10	Results of the mapping optimisation	148
7.1	Task independent components parameters	164
7.2	Task dependent parameters	165
7.3	Components in a typical technology library	166

Preface

It is likely that the demand for embedded computing systems with low energy dissipation will continue to increase. This book is concerned with the development and validation of techniques that allow an effective automated design of energy-efficient embedded systems. Special emphasis is placed upon system-level co-synthesis techniques for systems that contain dynamic voltage scalable processors which can trade off between performance and power consumption during run-time.

The first part of the book addresses energy minimisation of distributed embedded systems through dynamic voltage scaling (DVS). A new voltage selection technique for single-mode systems based on a novel energy-gradient scaling strategy is presented. This technique exploits system idle and slack time to reduce the power consumption, taking into account the individual task power dissipation. Numerous benchmark experiments validate the quality of the proposed technique in terms of energy reduction and computational complexity.

The second part of the book focuses on the development of genetic algorithm-based co-synthesis techniques (mapping and scheduling) for single-mode systems that have been specifically developed for an effective utilisation of the voltage scaling approach introduced in the first part. The schedule optimisation improves the execution order of system activities not only towards performance, but also towards a high exploitation of voltage scaling to achieve energy savings. The mapping optimisation targets the distribution of system activities across the system components to further improve the utilisation of DVS, while satisfying hardware area constraints. Extensive experiments including a real-life optical flow detection algorithm are conducted, and it is shown that the proposed co-synthesis techniques can lead to high energy savings with moderate computational overhead.

The third part of this book concentrates on energy minimisation of emerging distributed embedded systems that accommodate several different appli-

cations within a single device, i.e., multi-mode embedded systems. A new co-synthesis technique for multi-mode embedded systems based on a novel operational-mode-state-machine specification is presented. The technique increases significantly the energy savings by considering the mode execution probabilities that yields better resource sharing opportunities.

The fourth part of the book addresses dynamic voltage scaling in the context of applications that expose extensive control flow. These applications are modelled through conditional task graphs that capture control flow as well as data flow. A quasi static scheduling technique is introduced, which guarantees the fulfilment of imposed deadlines, while at the same time, reduces the energy dissipation of the system through dynamic voltage scaling.

The new co-synthesis and voltage scaling techniques have been incorporated into the prototype co-synthesis tool LOPOCOS (Low Power Co-Synthesis). The capability of LOPOCOS in efficiently exploring the architectural design space is demonstrated through a system-level design of a realistic smart phone example that integrates a GSM cellular phone transcoder, an MP3 decoder, as well as a JPEG image encoder and decoder.

Acknowledgments

Financial support of the work was provided by the Department of Electronics and Computer Science at the University of Southampton, the Embedded Systems Laboratory (ESLAB) at Linköping University, as well as the Engineering and Physical Sciences Research Council (EPSRC), UK.

Special thanks go to the members of the Electronic Systems Design Group (ESD) at the University of Southampton, for many fruitful discussions.

We would like to thank Christian Schmitz, who has contributed in deriving the smart phone benchmark during a visit at the University of Southampton.

We would also like to acknowledge Neal K. Bambha (University of Maryland, USA) and Flavius Gruian (Lund University, Sweden) for kindly providing their benchmark sets, which have been used to conduct some of the presented experimental results.

Chapter 1

INTRODUCTION

Over the last several years, the popularity of portable applications has explosively increased. Millions of people use battery-powered mobile phones, digital cameras, MP3 players, and personal digital assistants (PDAs). To perform major parts of the system's functionality, these mass products rely, to a great extent, on sophisticated embedded computing systems with *high performance* and *low power dissipation*. The complexity of such devices, caused by an ever-increasing demand for functionality and feature richness, has made the design of modern embedded systems a time-consuming and error-prone task. To be commercially successful in a highly competitive market segment with tight time-to-market and cost constraints, computer-based systems in mobile applications should be cheap and quick to realise, while, at the same time, consume only a small amount of electrical power, in order to extend the battery-lifetime. Designing such embedded systems is a challenging task.

This book addresses this problem by providing techniques and algorithms for the automated design of energy-efficient distributed embedded systems which have the potential to overcome traditional design techniques that neglect important *energy management* issues. In this context, special attention is drawn to *dynamic voltage scaling* (DVS) — an energy management technique. The main idea behind DVS is to dynamically scale the supply voltage and operational frequency of digital circuits during run-time, in accordance to the temporal performance requirements of the application. Thereby, the energy dissipation of the circuit can be reduced by adjusting the system performance to an appropriate level. Furthermore, the proposed synthesis techniques target the coordinated design (co-design) of *mixed hardware/software* applications towards the effective exploitation of DVS, in order to achieve substantial reductions in energy.

The main aims of this chapter are to introduce the fundamental problems that are involved in designing distributed embedded systems and to provide

and common design flow is introduced in Figure 1.2. It is characterised by three important design steps: *system specification* (Step A), *co-synthesis* (Step B), as well as concurrent *hardware and software synthesis* (Step C). The remainder of this section briefly outlines this design flow.

Starting from a new product idea, the first step towards a final realisation is *system specification*. At this stage, the functionality of the system is captured using different conceptual models [61] such as natural language, annotated-graphic representations (finite state machines, data-flow graphs), or high-level languages (VHDL, C/C++, SystemC). This design step is indicated as Step A in Figure 1.2. Having specified the system's functionality, the next stage in the design flow is the *co-synthesis*, shown as Step B in Figure 1.2. The goal of co-synthesis is threefold:

Architecture allocation: Firstly, an adequate target architecture needs to be allocated, i.e., it is necessary to determine the quantity and the types of different interconnected components that form the distributed embedded system. Components that can be allocated are given in a predefined technology library.

Application mapping: Secondly, all parts of the system specification have to be distributed among the allocated components, that is, tasks (function fragments) and communications (data transfers between tasks) are uniquely mapped to processing elements and communication links, respectively.

Activity scheduling: Thirdly, a correct execution order of tasks and communications has to be determined, i.e., the activities have to be scheduled under the consideration of interdependencies.

These three co-synthesis stages aim to optimise the design according to objectives set by the designer, such as power consumption, performance, and cost. In order to reduce the power consumption, emerging co-synthesis approaches (as the one proposed in this work) tightly integrate the consideration of *energy management techniques* within the design process [67, 76, 99, 100].

Energy management Energy management techniques utilise existing idle times to reduce the power consumption by either shutting down the idle components or by reducing the performance of the components.

The consideration of energy management techniques during the co-synthesis allows the optimisation of allocation, mapping, and scheduling towards their effective exploitation. After the co-synthesis has allocated an architecture as well as mapped and scheduled the system activities (tasks and communications), the next stage in the design flow is the concurrent *hardware and software synthesis*, indicated as Step C in Figure 1.2. These separated design steps transform the system specification, which has been split between hardware and software, into

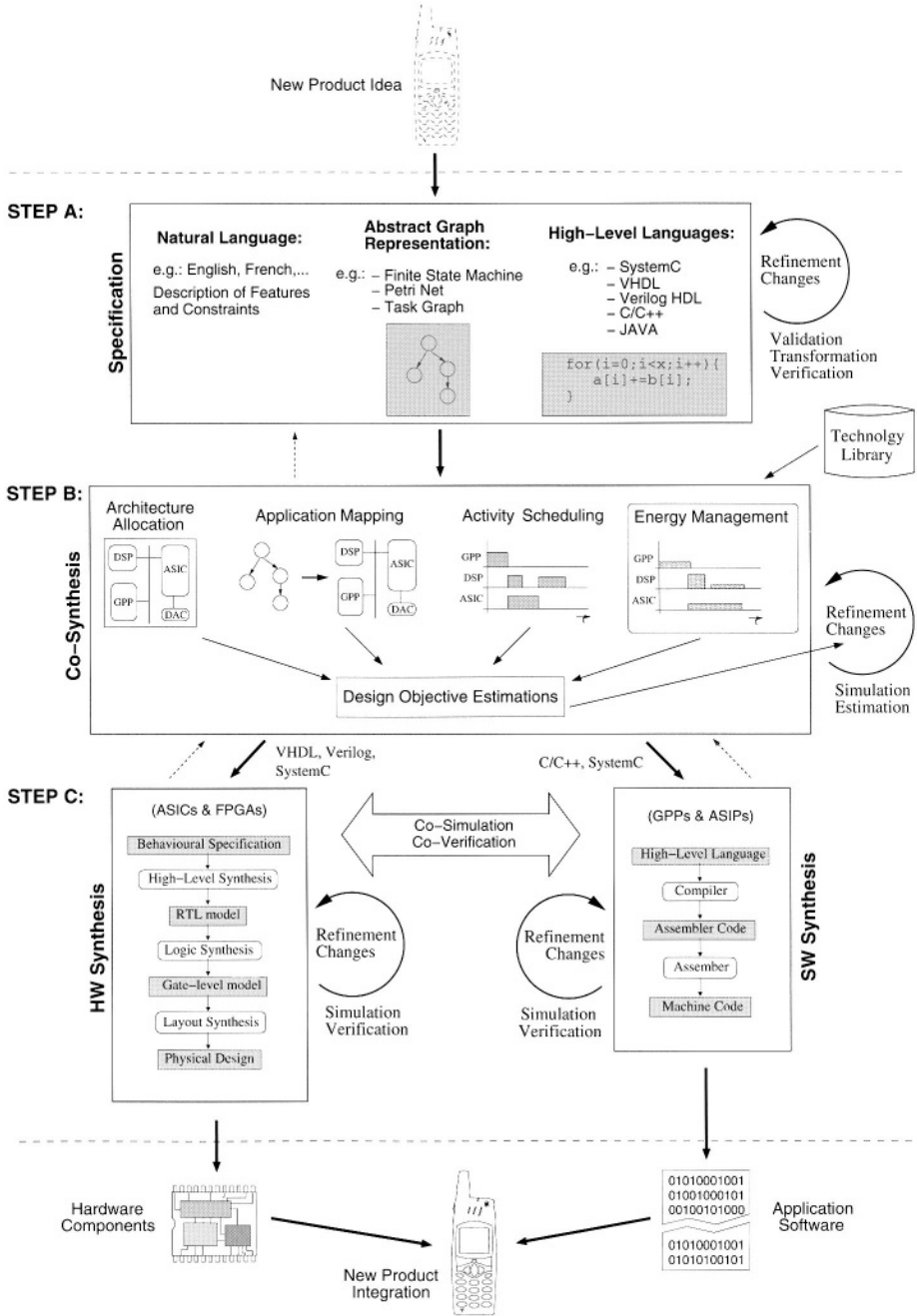


Figure 1.2. Typical design flow of a new embedded computing system

physical implementations. System parts that are mapped onto customised hardware are designed using high-level [8, 19, 60, 134, 154], logic [9, 42, 110, 131], and layout [56] synthesis tools. While system parts that have been mapped onto software programmable processors (CPUs, DSPs) are compiled into assembler and machine code, using either standard or specialised compilers and assemblers [1, 93]. The main advantage of a concurrent hardware (HW) and software (SW) synthesis is the possibility to co-simulate both system parts, with the aim of finding errors in the design as early as possible to avoid expensive re-designs. The following section describes the whole design process shown in Figure 1.2 in more detail and introduces the terminology used throughout this book.

1.2 System Specification (Step A)

The functionality of a system can be captured using a variety of conceptual specification models [61]. Different modelling styles are, for example, high-level languages (hardware description and programming languages) such as SystemC, Verilog HDL, VHDL, C/C++, or JAVA, as well as more abstract models such as block diagrams, task graphs, finite state machines (FSMs), Petri nets, or control/dataflow graphs. Typical applications targeted by the presented work can be found in the audio and video processing domain (e.g. multi-media and communication devices with extensive data stream operations). Such applications fall into the category of data-flow dominated systems. An appropriate representation for these systems is the task graph model [84, 112, 157], which will be introduced in the following section.

1.2.1 Task Graph Representation

The functionality of a complex system with intensive data stream operations can be abstracted as a directed, acyclic graph (DAG) $G_S = (\mathcal{T}, \mathcal{C})$, where the set of nodes $\mathcal{T} = \{\tau_0, \tau_1, \dots, \tau_n\}$ denotes the set of tasks to be executed, and the set of directed edges \mathcal{C} refers to communications between tasks, with $\gamma_{ij} \in \mathcal{C}$ indicating a communication from task τ_i to task τ_j . A task can only start its execution after all its ingoing communications have finished. Each task can be annotated with a deadline θ , the time by which its execution has to be finished. Furthermore, the task graph inherits a repetition period ϕ which specifies the maximal delay between to invocations of the source tasks (tasks with no ingoing edges). Structurally, task graphs are similar to the data-flow graphs that are commonly used in high-level synthesis [60, 154]. However, while nodes in data-flow graphs represent single operations, such as multiplications and additions, the nodes in task graphs are associated with larger (coarse) fragments of functionality, such as whole functions and processes. The concept behind this model can be exemplified using a simple illustrative example.

Example 1: For the purpose of this example, consider an MP3 audio decoder. In order to reconstruct the “original” stereo audio signal from an encoded stream, the decoder reads the data stream and applies several transformations such as Huffman decoding, dequantisation, inverse discrete cosine transformation (IDCT), and antialiasing. A possible task graph specification along with a high-level language description in C of such an MP3 decoder is shown in Figure 1.3. The figure outlines the relation between task graph model and high-level description. In this particular example the granularity of each task in the task graph corresponds to a single sub-function of the C specification. For instance, the Huffman Decoder tasks (τ_3 and τ_4) in Figure 1.3(a) reflect the functionality that is performed by the third sub-function in Figure 1.3(b). The flow of data is expressed by edges between the individual tasks. The output data produced by the Huffman Decoder tasks, for example, is the input of the dequant tasks (τ_5 and τ_6), indicated by the communication edges $\gamma_{3,5}$ and $\gamma_{4,6}$. In order to decode the compressed data into a high quality audio signal, one execution of all tasks in the graph, starting from task τ_0 and finishing with τ_{16} , has to be performed in at most $25ms$ as expressed by the task deadline θ_{16} . However, to obtain real-time decompression of a continuous music stream, the execution of all tasks has to be performed 40 times per second, i.e., with a repetition rate of $\phi = 25ms$. Although in this particular example the deadline and the repetition rate are identical, they might vary in other applications. As opposed to the C specification, the task graph explicitly exhibits *application parallelism* as well as *communication between tasks (data flow)*, while the exact algorithmic implementation of each function is abstracted away. \square

Task graphs can be derived from given high-level specification either manually or using extraction tools, such as the one proposed in [148].

1.3 Co-Synthesis (Step B)

Once the system’s functionality has been specified as task graph, the system designers will start with the system-level co-synthesis. This is indicated as Step B in Figure 1.2. In addition, Figure 1.4 shows the co-synthesis flow in diagrammatic form. Co-synthesis is the process of deriving a mixed hardware/software implementation from an abstract functional specification of an embedded system. To achieve this goal, the co-synthesis needs to address four fundamental design problems: *architecture allocation*, *application mapping*, *activity scheduling*, and *energy management*. Figure 1.4 shows the order in which these problems have to be solved. In general, these co-synthesis steps are iteratively repeated until all design constraints and objectives are satisfied [52, 54, 70, 156]. An iterative design process has the advantage that valuable feedback can be provided to the different synthesis steps. This feedback, which