Co-Synthesis of Energy-Efficient Multi-Mode Embedded Systems with Consideration of Mode Execution Probabilities

Marcus T. Schmitz, Bashir M. Al-Hashimi, *Senior Member, IEEE,* and Petru Eles, *Member, IEEE*

***Abstract*— In this paper, we present a novel co-design methodology for the synthesis of energy-efficient embedded systems. In particular we concentrate on distributed embedded systems that accommodate several different applications within a single device, i.e., multi-mode embedded systems. Based on the key observation that operational modes are executed with different probabilities, that is, the system spends uneven amounts of time in the different modes, we develop a new co-design technique that exploits this property to significantly reduce energy dissipation. Energy and cost savings are achieved through a suitable synthesis process that yields better hardware resource sharing opportunities. We conduct several experiments, including a realistic smart phone example, that demonstrate the effectiveness of our approach. Reductions in power consumption of up to 64% are reported.**

***Index Terms*— Energy efficiency, power minimization, embedded systems, system-level co-synthesis, multi-mode systems.**

I. 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*. One key characteristic of many current and emerging embedded systems is their need to work across a set of different interacting applications and operational modes. For instance, modern mobile phones often contain not solely the functionality required for communication purpose (e.g., voice coding and protocol handling), but additionally integrate applications like digital cameras, games, and complex multimedia functions (MP3 players and video decoders) into the same single device. Throughout this article, such embedded systems are referred to as *multi-mode embedded systems*. This paper introduces a novel co-synthesis methodology particular suitable for the design of energy-efficient multi-mode embedded systems. Starting from a specification model that captures both mode interaction and functionality, the developed co-synthesis technique maps the application under consideration of *mode execution probabilities* to a heterogeneous, distributed architecture with the aim to reduce the energy consumption through an appropriate resource sharing between tasks. Mode execution probabilities refer to the activation time of operational modes that are user typical. Consider for instance the

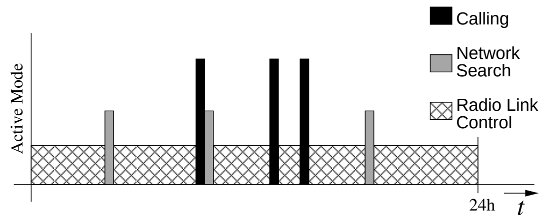


Fig. 1. Typical activation profile of a cellular phone

typical activation profile of a mobile phone, which is shown in Figure 1. According to this profile, the phone stays most of the time in a Radio Link Control mode, in order to maintain network connectivity. While the Network Search mode and the Calling mode are only active for small periods of the overall time. The main principle by which the proposed co-synthesis process achieves energy-efficiency is an implementation trade-off between the different operational modes. In general, modes with high execution probability should be implemented more energy efficient (e.g., by moving more tasks to hardware) than modes with a low execution probability. Nonetheless, the implementation of modes is heavily interrelated, due to the fact that different modes share the same resources (architecture). For example, mapping an energy-critical task of a highly active mode into energy-efficient hardware might prohibit to implement a timing-critical task into fast hardware due to the restricted area (see motivational example in Section IV). Clearly, a well balanced implementation of the operational modes is vital for a good system design.

In addition, the co-synthesis approach further reduces the energy dissipation by adapting the system performance to the particular needs of the active mode, using dynamic voltage scaling (DVS) as well as component shutdown. That is, instead of wasting energy through over-performance, the computational power is adapted according to the individual performance requirements of each mode and each task. Furthermore, we introduce a transformation-based method to extend existing DVS approaches in order to allow the scaling of hardware processing elements that are capable of executing tasks in

Manuscript received May 08, 2003; revised October 14, 2003. This work was supported in part by the UK Engineering and Physical Sciences Research Council (EPSRC), grant number GR/S95770/01. This paper was recommended by Associate Editor Margarida F. Jacome.

M. T. Schmitz and P. Eles are with the Department of Computer and Information Science, Linköping University, S-58333 Linköping, Sweden (email: g-marsc@ida.liu.se; petel@ida.liu.se).

Bashir M. Al-Hashimi is with the School of Electronics and Computer Science, University of Southampton, SO17 1BJ, United Kingdom (email: bmah@ecs.soton.ac.uk).

parallel, however, which rely on a single scalable supply voltage source.

The remainder of this paper is organized as follows. Section II introduces relevant previous work. Preliminaries, outlining a new multi-mode specification and an architectural model, are given in Section III. Motivational examples exemplify the need for a suitable multi-mode synthesis approach in Section IV. The problem at hand is formulated in Section V. Section VI describes our multi-mode co-synthesis approach, and Section VII presents experimental results. Finally, in Section VIII we draw some conclusions.

II. Previous Work

In the last decade, numerous methodologies for the design of low power consuming embedded systems have been proposed, including approaches that leverage power management techniques, such as dynamic power management (DPM) and dynamic voltage scaling (DVS). Nevertheless, a crucial feature of many modern embedded systems is their capability to execute several different applications (multi-modes), which are integrated into a single device.

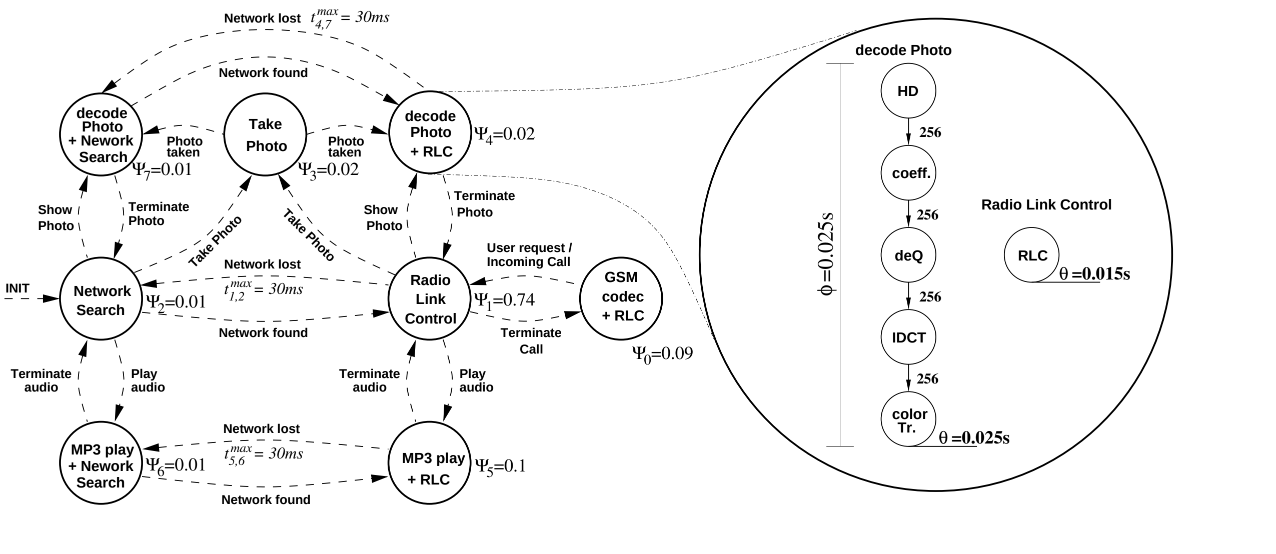
Approaches for the schedulability analysis of systems with several modes of operations can be found in the real-time research community [24], [29]. However, these approaches solely concentrate on scheduling aspects (i.e., they investigate if the mode change events fulfil the imposed timing constraints) and do not address implementation aspects. Three recent approaches have addressed various problems involved in the design of multi-mode embedded systems [19], [23], [30]. Shin *et al.* [30] proposed a schedulability-driven performance analysis technique for real-time multi-mode systems. They show that it is possible, through a sophisticated performance estimation, to identify timing-critical tasks, which are active in different operational modes. This identification allows to improve the execution times of the most crucial tasks, in order to achieve system schedulabilitly. In their work, the optimization of the identified tasks is up to the designer. For example, reductions in the execution times can be made by handcrafted code tuning and outsourcing of core routines into hardware. Kalavade and Subrahmanyam [19] have introduced a hardware/software partitioning approach for systems that perform multiple functions. Their technique classifies tasks, found within similar applications, into groups of task types. The implementation of frequently appearing task types is biased towards hardware. This can be intuitively justified by the fact that costly hardware implementations are shared across a set of applications, hence, exploiting the allocated hardware more cost effective. Oh and Ha [23] address the problem in a slightly different way. Their co-synthesis framework for multi-mode systems is based on a combined scheduling and mapping technique for heterogeneous multiprocessor systems (HMP [22]). Taking a processor utilization criterion into account, an allocation-controller selects the required processing elements (PEs) such that the schedulability constraint is satisfied and the system cost is minimized. The main principle behind all three approaches is to consider the possibility of resource sharing, i.e., computational tasks of the same type, which can be found in different modes, utilize the same implementations. Thereby, multiple hardware implementations of the same task type are avoided, which, in turn, reduces the hardware cost. Other noticeable approaches are the works by Chung *et al.* [12] and Yang *et al.* [32]. In [12], energy efficiency is achieved by leveraging information regarding the execution time variations, which is supplied to the mobile terminal by the contents provider. That is, the performance of the mobile terminal can be influenced directly by the contents provider, in accordance to the processing requirements of the sent content. The approach presented in [32] uses a two-phase scheduling method. In the first stage, which is performed off-line (during design time), a Pareto-optimal set of schedules is generated. These schedule provide different execution time/energy trade-offs. During run-time, a run-time scheduler selects points along the Pareto set, in order to account for the dynamic behavior of the application. As opposed to these approaches, the work presented in this paper addresses the design of low energy consuming multi-mode systems that exhibit variations in the mode activation profile; hence, it differs in several aspects from the previous works. To the authors' knowledge, there has been no prior work investigating the co-design problem of energy minimization taking into account mode execution probabilities. This paper makes the following contributions:
(a) The consideration of *mode execution probabilities* and their effect on the energy-efficiency of multi-mode embedded systems is analysed and demonstrated.
(b) A co-design methodology for the design of energy-efficient multi-mode systems is presented. The proposed co-synthesis maps and schedules a system specification that captures both mode interaction and mode functionality onto a distributed heterogeneous architecture. Four mutation strategies are introduced that aid the genetic algorithm-based optimization process in finding solutions of high quality by pushing the search into promising design space regions.
(c) Dynamic voltage scaling is investigated in the context of multi-mode embedded systems. A transformation-based approach is used to tackle the problem of DVS on processing elements that execute different tasks in parallel, but offer only a single scalable supply voltage source.

III. Preliminaries

This section introduces the functional specification model (Section III-A) and the architectural model (Section III-B), which are fundamental to the proposed co-synthesis framework.

A. Functional Specification of Multi-Mode Systems

The abstract specification model used for multi-mode embedded systems consists of two parts. In précis, it is based on a combination of finite state machines and task graphs, capturing both the interaction between different operational modes as well as the functionality of each individual mode. Structurally, each node in the finite state machine represents an operational mode and further contains the task graphs which are active during this mode. The following two sections introduce this model, which is henceforth referred to as operational mode



(a) Operational modes

(b) Task graph of a single mode

Fig. 2. Relation between OMSM and individual task graph specifications

state machine (OMSM). A similar abstract model was mentioned in [16]. However, here this model is extended towards system-level design and includes transition time limits as well as mode execution probabilities. The following introduces this model, using the smart phone example shown in Figure 2.

Top-level Finite State Machine: In this work, it is considered that an application is given as a directed cyclic graph $\Upsilon(\Omega, \Theta)$, which represents a finite state machine. Within this top-level model, each node $O \in \Omega$ refers to an operational mode and each edge $T \in \Theta$ specifies a possible transition between two different modes. If the system undergoes a change from mode O_x to mode O_y, where $x \neq y$, the transition time t_T^{max} associated with the transition edge $T = (O_x, O_y)$ has to be met. For instance, as indicated in Figure 2(a), upon losing the network connection the system needs to activate the Network Search mode within $30ms$. Such transition overheads can originate from the reconfiguration of FPGAs as well as from loading the application software of the particular mode into the local processing element's memory. At any given time there is only one active mode, i.e., the modes execute mutually exclusive. To exemplify the proposed model consider Figure 2(a). This figure shows the operational mode state machine for a smart phone example with eight different modes. A possible activation scenario could look like this: When switched on, the phone initialises into Network Search mode. The system stays in this mode until a suitable network has been found. Upon finding a network the phone undergoes a mode change to Radio Link Control (RLC). In this mode it maintains the connection to the network by handling cell hand-overs, radio link failure responses, and adaptive RF power control. An incoming phone call necessitates to switch the system into GSM codec + RLC mode. This mode is responsible for speech encoding and decoding, while simultaneously maintaining network connectivity. Similarly, the remaining modes have different functionalities and are activated upon mode change events. Such events originate upon user requests (e.g., MP3-player activation) or are initiated by the system itself (e.g., loss of network connection necessitates to switch the system into network search mode). Furthermore, based on the key observation that many multi-mode systems spend their operational time *unevenly* in each of the modes, an execution probability Ψ_O is associated with each operational mode O, i.e., it is known what percentage of the operational time the device spends in each mode. For instance, in accordance to Figure 2(a), the smart-phone stays 74% of this operational time in Radio Link Control (RLC) mode, 9% in GSM codec + RLC mode, and 1% in Network Search mode. The remaining 16% of the operation time are associated with the remaining modes. In practice the mode probabilities vary from user to user, depending on the personal usage behavior. Nevertheless, it is possible to derive an average activation profile based on statistical information collected from several different users. Taking this information into account will prove to be important when designing systems with a prolonged battery lifetime. It is interesting to note that different operational modes do not necessarily correspond to different functionalities of the system. It is possible to use alternative modes to model the same functionality under different working conditions (such as different workloads). For instance, in order to account for variations in the wireless channel quality, we could exchange the GSM voice transcoder mode O_0 in Figure 2(a) with three transcoder schemes, each responsible for the coding at a specific signal-to-interference ratio (SIR) on the channel. During run-time the appropriate transcoder scheme would be selectively activated, depending on the actual channel quality.

Functional Specification of Individual Modes: The functional specification of each operational mode $O \in \Omega$ in the top-level finite state machine is expressed by a task graph $G_S^O(\mathcal{T}, \mathcal{C})$. This relation is shown in Figure 2. Each node $\tau \in \mathcal{T}_O$ in a task graph represents a task, i.e., a fragment of functionality that needs to be executed without preemption. The level of granularity is coarse, i.e., tasks refer to func-

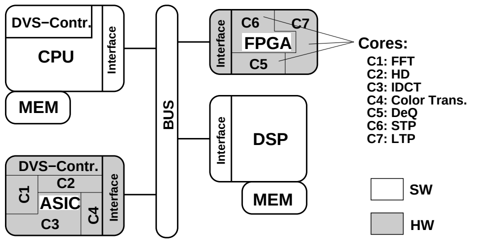


Fig. 3. Distributed architecture model

tions such as Huffman decoder, de-quantizer, FFT, IDCT, etc. Therefore, each task is further associated with a task type $\eta \in \Gamma = \{HD, deQ, FFT, IDCT, ...\}$. A distinctive feature of multi-mode systems is that task type sets $\Gamma^O \subseteq \Gamma$ of different modes $O \in \Omega$ can intersect, i.e., tasks of same type are executed in different modes. Such modes can share the same hardware resource (inter-mode sharing). Resource sharing is also possible for multiple tasks of identical type that are found in a single mode (intra-mode sharing); however, due to task communalities among different modes, the chances to share resources are increased. Further, tasks might be annotated with deadlines θ_τ (with $\tau \in \mathcal{T}_O$) by which the execution has to be finished, in order to guarantee correct functioning. Similarly, the whole task graph has to be successively repeated according to a period ϕ_O. Edges $\gamma \in \mathcal{C}$ in the task graph refer to precedence constraints and data dependencies between the computational tasks, i.e., if two tasks, τ_i and τ_j, are connected via an edge, then task τ_i must be finished and transfer data to task τ_j before τ_j can be executed. A feasible implementation of a certain mode O needs to respect all task deadlines θ, task graph period ϕ, and precedence relations.

B. Architectural Model and System Implementation

The proposed system-level synthesis approach targets distributed architectures that possibly consist of several heterogeneous processing elements (PEs), such as general-purpose processors (GPPs), application-specific instruction set processors (ASIPs), ASICs, and FPGAs. These components are connected through an infrastructure of communication links (CLs). A directed graph $G_A((\mathcal{P} \cup \mathcal{L}), \mathcal{E})$ captures such an architecture, where nodes $\pi \in (\mathcal{P} \cup \mathcal{L})$ denote PEs and CLs, while edges $\lambda \in \mathcal{E}$ impose the connections between those components. Figure 3 shows an architecture example. Since each task might have multiple implementation alternatives, it can be potentially mapped onto several different PEs that are capable of performing this type of task. Tasks mapped to software-programmable components (i.e., GPP or ASIP) are placed into local memory. However, if a task is mapped to a hardware component (i.e., ASIC or FPGA), a core for this task type needs to be allocated. A feasible solution needs to obey the imposed area constraints, i.e., only a restricted number of cores can be implemented on hardware components. The subdivision of hardware components (ASICs and FPGAs) into hardware cores is shown in Figure 3. Each core is capable of performing a single task of type $\eta \in \Gamma$ at a time. Tasks assigned to GPPs or ASIPs (software tasks) need to be sequenced, whilst the tasks mapped onto FPGAs and ASICs (hardware tasks) can be performed in parallel if the necessary resources (cores) are not already engaged. However, contention between two or more tasks assigned to the same hardware core requires a sequential execution order, similar to software tasks. Cores implemented on FPGAs can be dynamically reconfigured during a mode change, involving a time overhead, which needs to respect the imposed maximal mode transition times t_T^{max}. Further, PEs might feature dynamic voltage scaling to enable a trade-off between power consumption and performance that can be exploited during run-time. The relation between the dynamic power dissipation P_{dyn} and the circuit delay d (inverse proportional to performance) can be expressed using the following two equations [8], [11]:

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

$$d \propto 1/f = k \cdot \frac{V_{dd}}{(V_{dd} - V_t)^\alpha}$$

where C_{eff} is the effectively switched capacitance, V_{dd} denotes the circuit supply voltage, f represents the clock frequency, k and α are a circuit dependent constants, and V_t denotes the threshold voltage. As we can see from these equations, by varying the circuit supply voltage V_{dd}, it is possible to trade off between power consumption and performance. In reality, DVS processors are often restricted to run at discrete voltage levels [5], [6]. Therefore, a set $\mathcal{V}_\pi$ specifies the available discrete voltages of DVS-PE π. For such PEs a voltage schedule needs to be derived, in addition to a timing schedule. To implement a multi-mode application captured as OMSM, the tasks and communications of all operational modes need to be mapped onto the architecture, and a valid schedule for these activities $\epsilon \in \mathcal{A}$, where $\mathcal{A} = \mathcal{T} \cup \mathcal{C}$, needs to be constructed. As mentioned above, for tasks mapped to DVS-enabled components an energy reducing voltage schedule has to be determined. According to these aspects, an implementation candidate can be expressed through four functions, which need to be derived for each operational mode $O \in \Omega$:

Task mapping: $M_\tau^O : \mathcal{T} \rightarrow \mathcal{P}$

Communication mapping: $M_\gamma^O : \mathcal{C} \rightarrow \mathcal{L}$

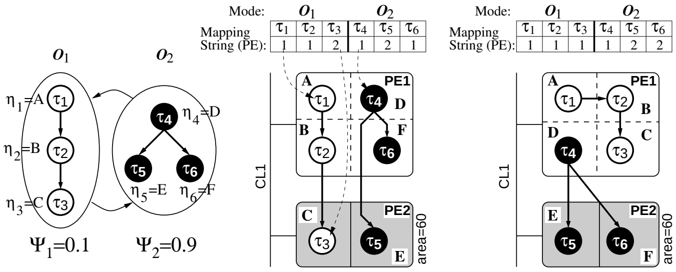
Timing schedule: $S_\epsilon^O : \mathcal{A} \rightarrow \mathbb{R}_0^+$

Voltage schedule: $V_\tau^O : \mathcal{T}_{DVS} \rightarrow \mathcal{V}_\pi$

where M_τ^O and M_γ^O denote task and communication mapping, respectively, assigning tasks to PEs and communications to CLs. Activity start times are specified by the scheduling function S_ϵ^O, while V_τ^O defines the voltage schedule for all tasks $\tau \in \mathcal{T}_{DVS}$ mapped to DVS-PEs, where $\mathcal{V}_\pi$ is the set of the possible discrete supply voltages of PE π. Clearly, the mappings as well as the corresponding schedules are defined for every mode separately, i.e., during the change from mode O_x to mode O_y, the execution of activities found in mode O_x are finished and the activities of mode O_y are activated.

IV. Motivational Examples

The aim of this section is to motivate the key ideas behind the new multi-mode co-synthesis, that is, the consideration of



(a) Application specified by two interacting modes
(b) Optimised without mode consideration
(c) Optimised with mode consideration

Fig. 4. Example of mode execution probabilities

mode execution probabilities and multiple task type implementations. First, the influence of mapping in the context of multi-mode embedded systems with different mode execution probabilities is demonstrated. Second, it is illustrated that multiple task implementations can further reduce the energy dissipation of multi-mode embedded systems.

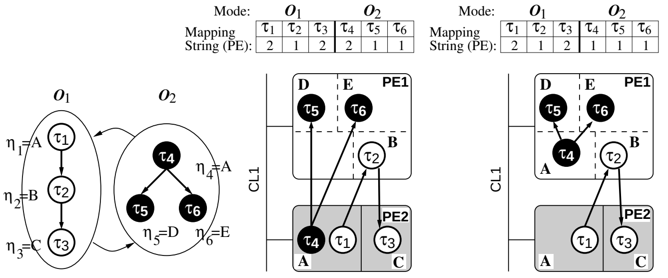
Example – Mode Execution Probabilities: For simplicity, timing and communication issues are neglected in the following example. Consider the application shown in Figure 4(a), which consists of two operational modes, O_1 and O_2, each specified by a task graph with three tasks. The system spends 10% of its operational time in mode O_1 and the remaining 90% in mode O_2, i.e., the execution probabilities are given by $\Psi_1 = 0.1$ and $\Psi_2 = 0.9$. The specification needs to be mapped onto a target architecture built of one general-purpose processor (PE1) and one ASIC (PE2), linked by a bus (CL1). Depending on the task mapping to either of the components, the execution properties of each task are shown in Table I. In

task type	PE1 (SW) execution time (ms)	PE1 (SW) dynamic energy (mJ)	PE2 (HW) execution time (ms)	PE2 (HW) dynamic energy (mJ)	PE2 (HW) area (mm^2)
A	20	10	2	0.010	24.0
B	28	14	2.2	0.012	30.0
C	32	16	1.6	0.023	27.5
D	26	13	3.1	0.047	24.5
E	30	15	1.8	0.015	21.0
F	24	14	2.2	0.032	28.0

TABLE I
TASK EXECUTION AND IMPLEMENTATION PROPERTIES

general, hardware implementations of tasks achieve a higher performance and are more energy efficient [9]. It can be observed that all tasks are of different type, therefore, if a task is mapped to HW, a suitable core needs to be allocated explicitly for that task. Hence, in this particular example, no hardware sharing is considered. Each allocated core uses area on the hardware component that offers $60mm^2$, i.e., at most 2 cores can be allocated at the same time without violating the area constraint (see Table I, Column 6). Note that although the two modes execute mutually exclusive, the task types implemented in hardware (HW cores) cannot be changed during run-time, since their implementation is static (non-reconfigurable ASIC); as opposed to software-programmable components. Consider the mapping shown in Figure 4(b) in which the highest energy consuming tasks (τ_3 and τ_5, when implemented in software) are executed using a more energy-efficient hardware implementation. According to the task characteristics given in Table I, the energy dissipation during modes O_1 and O_2 are $E_1 = 10mJ + 14mJ + 0.023mJ = 24.023mJ$ and $E_2 = 13mJ+0.015mJ+14mJ = 27.015mJ$. Neglecting the mode execution probabilities by assuming that both modes are active for even amounts of time (50% mode 0_1 and 50% mode O_2), the energy consumption can be calculated as $E_e = 0.5 \cdot 24.023mJ+0.5\cdot 27.015mJ = 25.519mJ$. Nevertheless, taking the real behavior into account, mode O_1 is active for 10% of the operational time, i.e., its energy dissipation can then be calculated as $E_{r1} = 0.1 \cdot 24.023mJ = 2.4023mJ$. Similarly, mode O_2 is active 90% of the operational time, hence, its energy is given by $E_{r2} = 0.9 \cdot 27.015mJ = 24.3135mJ$. Thus, the real energy dissipation results in $E_r = E_{r1} + E_{r2} = 26.7158mJ$. Now consider an alternative mapping, shown in Figure 4(c), for the same task graphs. In this configuration tasks τ_5 and τ_6, i.e., the most energy dissipating tasks of the highly active mode O_2, use energy-efficient hardware implementations on PE2, while task τ_3 of the less active model O_1 is shifted into the software-programmable processor (PE1). According to this solution, the energy consumptions of modes O_1 and O_2 are given by $E_1 = 10mJ+14mJ+16mJ = 40mJ$ and $E_2 = 13mJ + 0.015mJ + 0.032mJ = 13.047mJ$. Considering the even execution of each mode (neglecting the execution probabilities), the energy consumption can be calculated as $0.5 \cdot 40mJ + 0.5 \cdot 13.047mJ = 26.524mJ$. Note that this value is higher than the corresponding energy of the first mapping ($E_e = 25.519mJ$). Thus, a co-synthesis approach that neglects the mode execution probabilities would optimize the system towards the first mapping. However, in real-life the modes are active for different amount of time and hence the real energy dissipation is given by $E_r = 0.1\cdot 40mJ+0.9\cdot 13.047mJ = 15.7423mJ$. This is 41% lower compared to the first mapping ($E_r = 26.7158mJ$) shown in Figure 4(b), which is not optimized for an uneven task execution probability. Furthermore, the second task mapping allows to switch off PE2 and CL1 during mode O_1, since all tasks of this mode are assigned to PE1. This results in a significant reduction of the static power, additionally increasing the energy savings.

Example – Multiple Task Type Implementations: An important characteristic of multi-mode systems is that tasks of the same type might be found in different modes, i.e., resources can be shared among the different modes in a time-multiplexed fashion. To increase the possibility of component shutdown, it might be necessary to implement the same task type multiple times, however, on different components. The following example, shown in Figure 5, clarifies this aspect. Here tasks τ_1 and τ_4 are of type A (see Figure 5(a)), allowing resource sharing between these tasks. The sharing is possible without contention due to the mutual exclusive execution of these tasks (only one mode is active at a given time). In the first mapping, given in Figure 5(b), both tasks utilize the same HW core. However, implementing task τ_4 in software (additional task type A on PE1), as shown in Figure 5(c), allows to shut down PE2 and CL1 during the execution of mode O_2. Hence,



(a) Application with resource sharing possibility (b) Resource sharing, but no shut–down possible (c) No resource sharing, but component shut–down

Fig. 5. Multiple task type implementations

multiple implementations of task types can help to reduce power dissipation.

These two examples have demonstrated that it is essential to guide the synthesis process by: (a) an energy model that takes into account the mode execution probability as well as (b) allows multiple task implementations.

V. Problem Formulation

The goal of our co-synthesis approach is an energy-efficient implementation of application Υ, which is modeled as OMSM, such that timing and area constraints are satisfied. This involves the derivation of the mapping and schedule functions, M_τ^O, M_γ^O, S_ϵ^O, and V_τ^O (outlined in Section III-B), under the consideration of static and dynamic power as well as mode execution probabilities. Although static power consumption is often neglected in system-level design approaches, since until recently dynamic power has been the dominating power dissipation, emerging sub-micron technologies with reduced threshold voltage levels show increased leakage currents that are becoming comparable to the dynamic currents [10]. In multi-mode systems this static power consumption can have a significant impact on the overall energy efficiency. The reasons for this are the different performance requirements of the various operational modes. For instance, the minimal performance requirements of the hardware architecture are imposed by the most computational intensive mode, i.e., the minimal allocated architecture has to provide enough computational power to execute this performance critical mode. However, the allocated architecture might be far more powerful than actually needed for the execution of modes with low performance requirements. Furthermore, low performance modes, such as the standby-mode of mobile phones (i.e., Radio Link Control), often account for the greatest portion of the system time. During such circumstances, the static energy dissipation of unnecessarily switched-on PEs and CLs can outweigh the dynamic energy consumption caused by tasks of a "lightweight" mode. Thus, switching-off the unneeded components becomes an important aspect particularly in multi-mode embedded systems. In accordance, an accurate estimation of the average power consumption of an implementation alternative should consider both static and dynamic power, and further the mode execution probabilities. The average power consumption $\bar{p}$ can be expressed using the following equation:

$$\bar{p} = \sum_{O \in \Omega} (P_O^{stat} + P_O^{dyn}) \cdot \Psi_O \tag{1}$$

where P_O^{stat}, P_O^{dyn}, and Ψ_O refer to the static power dissipation, the dynamic power dissipation, and the execution probability of mode O, respectively. The static and dynamic power consumptions are given as:

$$P_O^{stat} = \sum_{\xi \in \mathcal{K}_O} P^{stat}(\xi) \tag{2}$$

and

$$P_O^{dyn} = \left(\sum_{\epsilon \in \mathcal{A}_O} E^{dyn}(\epsilon) \right) \cdot \frac{1}{hp_O} \tag{3}$$

where $P^{stat}(\xi)$ refers to the static power consumption of a component ξ, which is found in the set of all active components $\mathcal{K}_O \subseteq (\mathcal{P} \cup \mathcal{L})$ of mode O. Please note that this static power consumption also includes the additional power required for the DC/DC converter of voltage-scalable processors. Further, $\mathcal{A}_O$ and hp_O denote all activities and the hyper-period of mode O, respectively. With respect to the type of activities, the dynamic energy consumption $E^{dyn}(\epsilon)$ can be calculated as:

$$E^{dyn}(\epsilon) = \begin{cases} P_{max}(\epsilon) \cdot t_{min}(\epsilon) \cdot \frac{V_{dd}^2(\epsilon)}{V_{max}^2(\epsilon)} & \text{if } \epsilon \in \mathcal{T}_{\text{DVS}} \\ P_{max}(\epsilon) \cdot t_{min}(\epsilon) & \text{if } \epsilon \in \mathcal{T} \setminus \mathcal{T}_{\text{DVS}} \\ P_C(\epsilon) \cdot t_C(\epsilon) & \text{if } \epsilon \in \mathcal{C} \end{cases} \tag{4}$$

where P_{max} is the dynamic power consumption and t_{min} the execution time of tasks when executed at nominal supply voltage V_{max}. Tasks $\tau \in \mathcal{T}_{\text{DVS}}$ mapped to DVS-PEs can execute at a scaled supply voltage V_{dd}, resulting in a reduced energy consumption. Further, communications consume power P_C over a time t_C. If the DVS-enabled processors are restricted to a limited set of discrete voltages, the continuous selected supply voltage V_{dd} is split into its two neighboring discrete voltages V_{low} and V_{high}. The corresponding execution times in each voltage are calculated as given in [25]. The mode execution probabilities used in Equation (1) are either based on approximations or statistical information collected from several real users. In the case that statistical information is available from a set of different users U, the average execution probabilities $\overline{\Psi}_O$ of a single operational mode $O \in \Omega$ can be calculated.

The co-synthesis goal is to find a task mapping M_τ^O, a communication mapping M_γ^O, a starting time schedule S_ϵ^O, as well as a voltage schedule V_τ^O for each operational mode O, such that the total average power $\bar{p}$, given in Equation (1), is minimized and the deadlines are satisfied. Furthermore, a feasible implementation candidate needs to fulfill the following requirements:

(a) The mapping of tasks M_τ^O does not violate area constraints in terms of memory and hardware area, i.e., $(\sum_{\eta \in \Gamma_\pi} a_\eta) \leq a_\pi^{max}, \quad \forall \pi \in \mathcal{P}$; where Γ_π is the set of all task types implemented on PE π, and a_η and a_π^{max} refer to the area used by task type η and the available area on PE π, respectively. Please note that for DVS-enabled

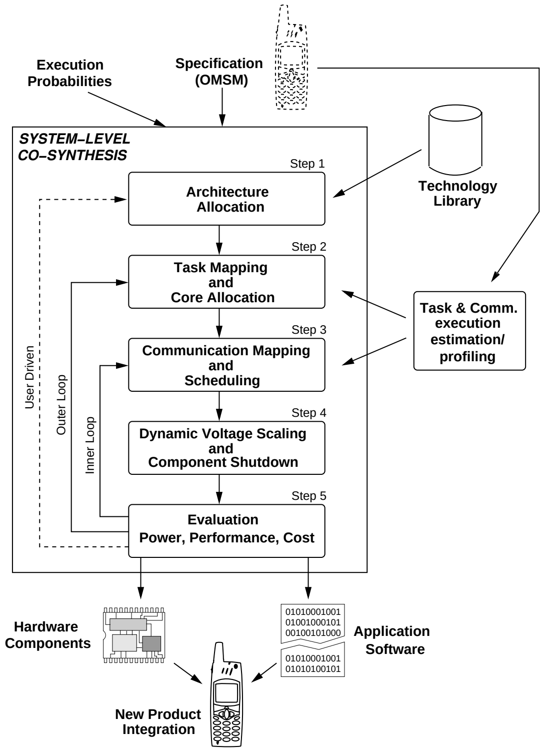


Fig. 6. Multi-mode embedded systems design flow

HW, a_{π}^{max} represents the available area including the area overhead required for the DC/DC converter.

(b) The timing schedule S_{ϵ}^{O} and the voltage schedule V_{τ}^{O}, based on task and communication mapping, do not exceed any task deadlines θ_{τ} or task graph repetition periods ϕ_{O}, therefore, $t_S(\tau)+t_{exe}(\tau) \leq \min(\theta_{\tau}, \phi), \ \forall \tau \in \mathcal{T}$; where $t_S(\tau)$ and t_{exe} refer to task start time and task execution time (potentially based on voltage scaling).

(c) The system reconfiguration time t_T between mode changes does not exceed the imposed maximal mode transition times t_T^{max}. Hence, $t_T \leq t_T^{max}, \ \forall T \in \Theta$ needs to be respected for all mode transitions.

VI. Co-Synthesis of Energy-Efficient Multi-Mode Systems

Figure 6 shows an overview of our co-synthesis flow for multi-mode embedded systems. This design flow is primarily based on two nested optimization loops. The outer loop optimizes task mapping and core allocation, while the inner loop is responsible for the combined optimization of communication mapping and scheduling. Although we concentrate in this paper on task mapping and core allocation, we will outline briefly this overall design flow. As we can observe from Figure 6, an initial system specification has to be translated into the final hardware and software implementations. In our approach the specification includes information regarding the execution probabilities. Along the design flow we can identify five major synthesis steps:

(1) An adequate target architecture needs to be allocated, i.e., it is necessary to determine the quantity and the types of the different interconnected components (processing elements and communication links). Available components are specified in the technology library.

(2) The tasks of the system specification and the required communications have to be uniquely mapped among the allocated components. Based on the component to which a task has been mapped, its execution properties are determined. This is done in accordance to previously established execution estimations and profile information. Furthermore, hardware cores are allocated based on the available hardware area and the application parallelism.

(3) Depending on the task mapping, the communications are mapped onto the allocated communication links, and the activities are scheduled with the aim to meet the imposed task deadlines, while, at the same time, achieve a good slack distribution in order to reduce energy via DVS.

(4) Dynamic voltage scaling and component shutdown possibilities are exploited to reduced the system energy consumption.

(5) The system implementation candidates, specified by the synthesis steps 1–4, are evaluated in terms of power consumption, performance (deadline satisfaction), and cost (architecture and hardware area). This step is used to provide feedback to the previous synthesis steps in order to refine the design.

For more information we refer the interested reader to [25]–[27]. A major goal of this design flow is to support the system designer with a methodology that aids to find suitable target architectures for a given system specification.

In this paper, we concentrate on the task mapping and core allocation step, since the scheduling and communication mapping can be carried with standard single mode techniques (e.g., [17], [26]). This is due to the fact that the modes are executing mutually exclusive. Thus, here we present new techniques and algorithms for task mapping, hardware core allocation, and dynamic voltage scaling that suit the particular problems of multi-mode embedded systems. However, since these approaches are targeted towards the exploitation of mode execution probabilities, we first discuss how such probabilities can be obtained in practice.

A. Estimation of Mode Execution Probabilities

As we have demonstrated in the motivational example of Section IV, the consideration of mode execution probabilities during design time can help to significantly reduce the energy consumption of the embedded system. Certainly, to achieve a good design it is necessary that the execution probabilities (estimations) used during design time reflect the real usage probabilities (in-field) accurately. In the following, we outline how to obtain adequate execution probabilities using two different design scenarios:

(a) *The new design is an upgrade of an existing product which is connected to a service provider* (e.g., a new version of a mobile phone). For such product types it is possible to use information regarding the activation profile that has been collected on the provider side during the operation of the previous product generation. For instance, the cellular network base stations can record the activation profile of the mobile terminals (e.g., phones with radio link control and

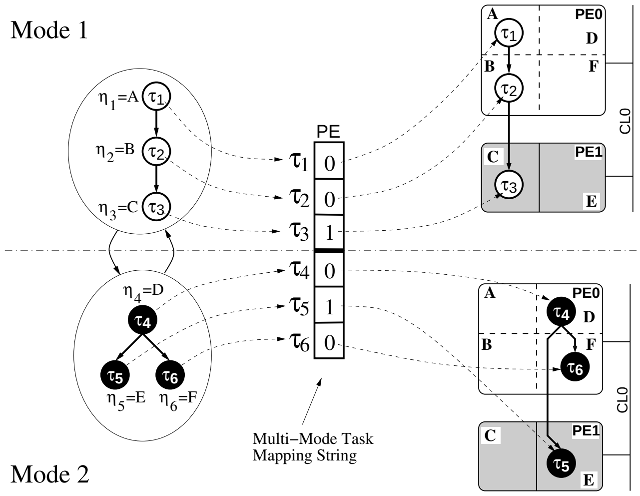


Fig. 7. Task mapping string for multi-mode systems

calling mode) directly in-field. This information could then be evaluated and used during the design of the new product. (b) *The product is a completely new design*. In this situation, it is common practice to evaluate the market acceptance before the final product is introduced, using a limited number of prototypes that are distributed among a set of evaluation users. During this evaluation phase, the prototypes can gather information regarding the activation profile. This information could then be used during the final design of the product to optimize the energy consumption. Of course, it is also possible to use application-specific insight of the designer to estimate the execution probabilities. As we will show in the experiments given in Section VII-A, even if the estimated execution probabilities do not reflect the user activation with absolute accuracy, but are sufficiently close to the real values, energy savings can be still achieved.

B. Multi-Mode Co-Synthesis Algorithm

The task mapping approach, which simultaneously determines M_τ for all modes of application Υ, is driven by a genetic algorithm (GA). GAs optimize a population of individuals over several generations by imitating and applying the principles of natural selection. That is, the GA iteratively evolves new populations by mating (crossover) the fittest individuals (highest quality) of the current population pool until a certain convergence criterion is met. In addition to mating, mutation, i.e., the random change of genes in the genome (string), provides the opportunity to push the optimization into unexplored search space regions. GA-driven task mapping approaches have already been shown to provide a powerful tool in deriving mappings for single modes systems [15], [25]. Here we enhance such approaches towards multi-mode aspects. These enhancements include the consideration of resource sharing, component shutdown, and mode transition issues. As opposed to the single mode task mapping strings, such strings for multi-mode specifications combine the mapping strings of each operational mode into one larger task mapping string, as shown in Figure 7. Within this string each number represents the PE to which the corresponding task is assigned. This encoding enables the usage of a genetic algorithm to optimize the placement of tasks across the processing elements that form the distributed architecture. Please note that this representation supports the implementation of multiple task types. For instance, if two tasks of the same type are mapped onto different PEs, these tasks are implemented on both PEs. Thereby, the possibility of multiple task implementations is mainly inherited into the genetic mapping algorithm which is guided by a cost function that accounts for the multiple task implementations, i.e., the GA trades off between the savings in static power consumption against the increase dynamic power.

The goal of the co-synthesis is to find a mapping of tasks that minimizes the total power consumption and obeys the performance constraints. Figure 8 outlines the pseudocode of our co-synthesis algorithm. Starting from an initial random population of multi-mode task mapping strings (line 1), the optimization runs until the convergency criterion is met (line 2). The used criterion is based on the diversity in the current population and the number of elapsed iterations without producing any improved individual. To judge the quality of mapping candidates, i.e., the fitness which guides the genetic algorithm, it is necessary to estimate important design objectives, including static and dynamic power dissipation,

```
Algorithm: MULTI-MODE-SYN
--------------------------------------------------------------
Input:  - OMSM (FSM+TGs), technology lib,
          allocated architecture
Output (Outer loop): - Core allocation
                     - Task mapping
Output (Inner loop): - Communication mapping
                     - Scheduling
                     - Scaled supply voltages
--------------------------------------------------------------
01: Pop ← InitRandPop
02: while (NotConv(Pop)) {  //outer loop
03:   for all map ∈ Pop {
04:     mob ← CompMobil(map)  //ASAP&ALAP based
05:     cores ← ImplementHWcores(map,mob)
06:     ap ← CalcAreaPenalty(map,cores)
07:     P^stat_PE ← CalcStaticPowPE
08:     trp ← CalcTransPenalty(cores)
09:     for all mode ∈ Ω {  //inner loop
10:       CommMapping_Scheduling(mode)
11:       tp(mode) ← CalcTimingPenalty(mode)
12:       P^dyn(mode) ← CalcDynPow(mode) //+ DVS
13:       P^stat_CL ← CalcStaticPowCL
14:     } // end for
15:     F_M=MappingFitness(P^dyn, P^stat_CL, P^stat_PE,
                           tp, ap, trp)
16:   } // end for
17:   RankIndividuals(F_M, Pop)
18:   mat ← SelMatIndividuals(Pop)
19:   TwoPointCrossover(mat)
20:   OffspringInseration(Pop)
21:   ShutdownImproveMut(Pop)
22:   AreaImproveMut(Pop)
23:   TimingImproveMut(Pop)
24:   TransImprovetMut(Pop)
25: } // end while
```

Fig. 8. Pseudocode: Multi-mode co-synthesis

area usage, and timing behavior (lines 03–14). The following explains each of the required estimations. The hardware area depends on the allocated cores on each hardware component (ASIC or FPGA). Of course, for each task type mapped to hardware at least one core of this type needs to be allocated. However, if too many cores are placed onto a single ASIC or FPGA, the available area is exceeded and an area penalty is introduced (line 6). On the other hand, if multiple tasks of the same type are mapped to the same hardware component and the hardware area is not violated, it is possible to implement cores multiple times (if helpful for the energy reduction). In the proposed approach, additional cores (line 5) are allocated for parallel tasks with low mobility (line 4), therefore, the chance to exploit application parallelism is increased. The mobility of a tasks is the difference between its earliest possible start time and its latest possible start time [31]. Clearly, from an energy point of view this is also preferable, especially in the presence of DVS, where a decreased execution time results in more slack that can be exploited. Section VI-C describes the core allocation in more detail. At this point it is possible to compute the static power consumption of the implementation (line 7), taking into account component shutdown between different modes. Components can be shut down during the execution of a certain mode whenever no tasks belonging to that mode are mapped onto these components, i.e., the component is vacant (for instance, PE0 during execution of mode O_2 in Figure 5). Another important aspect is the reconfigurability of FPGAs, which allows to exchange the implemented cores to suit the active mode. However, this reconfiguration during a mode change takes time, hence, a transition penalty is introduced if the maximal transition times are exceeded (line 8). Having determined the cores to be implemented (line 5), it is now also possible to schedule each mode of the application and to derive a feasible communication mapping (line 10). Since the modes are mutually exclusive, we can employ a communication mapping and scheduling optimization for a single mode system. In our approach we utilize the technique described in [25] for this step. If timing constraints are violated by the found schedule, a timing penalty is introduced (line 11). Furthermore, based on the communication mapping and scheduling, the dynamic power consumption of the application can be computed, taking into account DVS (line 12) if voltage-scalable components are present. Similarly to the shutdown of PEs, it is also possible to switch off a CL when no communications are mapped to that link (line 13), therefore, further reducing the static power consumption of the system. Based upon all estimated power consumptions and penalties, a fitness is calculated (line 15) as,

$$F_M = \bar{p} \cdot tp \cdot \left(1 + w_A \cdot \sum_{\pi \in \mathcal{P}_v} \frac{a_\pi^U - a_\pi^{max}}{a_\pi^{max} \cdot 0.01}\right) \cdot w_R \cdot \prod_{T \in \Theta_v} \frac{t_T}{t_T^{max}} \tag{5}$$

where the average power dissipation $\bar{p}$ is given by Equation (1) and tp introduces a timing penalty if the schedule exceeds task deadlines or the repetition period. Further, an area penalty is applied for all PEs with area violation $\mathcal{P}_v$ by relating used area a_π^U and area constraint a_π^{max}. Similarly, a transition time penalty is applied for all transitions Θ_v that exceed their maximal transition time limit t_T^{max}. Both area and transition penalties are weighted (w_A and w_R), which allows to adjust the aggressiveness of the penalties. Having assigned a fitness to all individuals of the population, they are ranked using linear scaling (line 17). A tournament selection scheme is used to pick individuals (line 18) for mating (line 19). The produced offsprings are inserted into the population (line 20).

In order to improve the performance of the genetic algorithm, we apply four genetic mutation strategies that add problem-specific knowledge into the optimization process (lines 21–24). This is achieved by introducing a small number of mutated individuals into the current population whenever the optimization process occurs to be trapped. These newly injected solution candidates provide the potential to turn into high quality solution by mating with other solution. The mutation strategies are introduced next:

Shutdown Improvement: To increase the chances of component shutdown, which leads to a reduction of static power consumption, the genetic task mapping algorithm employs a simple yet effective strategy during the optimization. Out of the current population randomly picked individuals (probability 2% was found to lead to good results) are modified as follows. A single mode O_x and a non-essential PE π_a are selected. Non-essential PEs are considered to be PEs that implement task types that have alternative implementations on other PEs, hence, they are not fundamental for a feasible solution. Our goal is to switch off PE π_a during the execution of mode O_x. Therefore, all tasks of mode O_x which are mapped to π_a are randomly remapped to the remaining PEs ($\mathcal{P} \setminus \pi_a$), hence, PE π_a can be shut down during mode O_x. Of course, only feasible mappings are allowed, i.e., tasks are always mapped randomly to the PEs that are capable of executing this kind of task type.

Area Improvement: To avoid convergence towards area infeasible solutions, a second strategy is employed. If only infeasible area mappings have been produced for a certain number of generations, the search is pushed away from this region by randomly re-mapping hardware tasks onto software-programmable PEs.

Timing Improvement: In contrast to the area improvement strategy, if a certain amount of timing infeasible solutions have been produced, software tasks are randomly mapped to faster hardware implementations. Thereby, the chance to find timing feasible implementations is increased.

Transition Improvement: Cores implemented in FPGAs can be dynamically reconfigured. However, this involves a time overhead. If this overhead exceeds the imposed transition time limits, the mapping is infeasible. Hence, after generating for a certain number of generations solely solutions that violate the transition times, tasks are randomly re-mapped away from the FPGAs that cause the violations.

Although some of the produced genomes (strings) might be infeasible in terms of area and timing behavior, all these strategies have been found to improve the search process significantly by introducing individuals that evolve into high quality solutions. For instance, running the synthesis process (on examples of moderate size) without the shutdown improvement strategy often results in implementations which do not exploit this energy reduction possibility.

C. Hardware Core Allocation

For tasks mapped to ASICs and FPGAs it is necessary to allocate hardware cores that are capable of executing the required task types. This is a trivial job as long as only tasks of different types are mapped to the same hardware component, i.e., when a single core for each task needs to be allocated. Nevertheless, if tasks of the *same* type η are assigned to the *same* PE more than once, it is necessary to make a decision upon how many cores of type η need to be implemented. This is important because hardware cores are able to execute tasks in parallel, i.e., the right quantitative choice of cores can efficiently help to exploit application parallelism, hence, improve the timing behavior as well as energy dissipation. In the proposed co-synthesis, the following approach is employed during the schedule optimization. Initially, each task type assigned to hardware is implemented only once, even if multiple tasks of this type are mapped onto the same PE. This ensures that all hardware tasks have at least one executable core implementation. If the hardware area constraints are not violated through the initial allocation, additional cores are implemented as follows. The tasks are analysed to identify possibly parallel executing task, taking into account task dependencies. These tasks are then ordered according to their mobility. Clearly, tasks with low mobility are more likely to improve the timing behavior and therefore should be the preferred choice when implementing additional hardware cores. Cores for tasks with low mobility are implemented as long as the area constraints of the hardware components are not violated. Note that this strategy potentially improves the dynamic energy dissipation, since it is probable to result in more slack time, which, in turn, can be exploited through DVS.

D. Dynamic Voltage Scaling for Multiple Parallel Executing Tasks

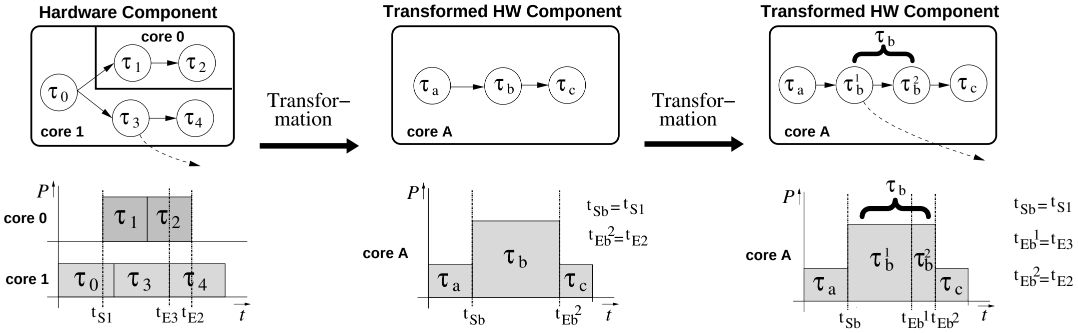
Dynamic voltage scaling is a powerful technique to reduce energy consumption by exploiting temporal performance requirements through dynamically adapting processing speed and supply voltage of PEs. The applicability of DVS to distributed embedded systems was demonstrated in [7], [17], [20], [28]. However, these works concentrate on dynamically changing the performance of software PEs only, while parallel execution of tasks on hardware resources has been neglected. Nevertheless, in the context of energy-efficient multi-mode systems, where performance requirements of each operational mode can vary significantly, DVS needs to be considered carefully. Consider, for instance, an inverse discrete cosine transformation (IDCT) algorithm implemented in fast hardware which is used during two modes: MP3 decoding and JPEG image decoding. Clearly, the JPEG decoder should restore images as quickly as possible, i.e., the IDCT hardware is required to execute at maximal supply voltage (equivalent to peak performance). On the other hand, the MP3 decoder works at a fixed repetition rate of $25ms$ for which the hardware implementation operates faster than necessary, i.e, the IDCT performance can be reduced such that this repetition rate is adequately met. By using DVS it is possible to adapt the execution speed to suit both needs and to reduce the energy consumption to a minimum. Here we consider that hardware components might employ DVS. However, due to the area and power overhead involved in additional DVS circuitry (DC/DC-converter [14], [21]) it is assumed that all cores allocated to the same hardware component are fed by a single voltage supply, i.e., dynamically scaling the supply voltage simultaneously affects the performance of all cores on that hardware component.

The proposed technique enables an efficient usage of existing DVS approaches [7], [17], [20], [28] to handle the case of dynamic voltage scaling on hardware components that execute tasks in parallel. To cope with this problem, the potentially parallel executing tasks on a single voltage-scalable hardware resource are transformed into an equivalent set of sequentially executing tasks, taking into account the dynamic power dissipation on each core. Note that this is done to calculate the scaled supply voltages only, i.e., this virtual transformation does not affect the real implementation. In this section we highlight solely our transformation-based approach, while we refer the interested reader to [7], [17], [20], [28], where different voltage scaling techniques are described in detail.

The following example illustratively outlines the proposed transformation approach. Figure 9 shows the transformation of five hardware tasks, executing on two cores (both cores are implemented within the same hardware component), to four sequential tasks on a single core. The given schedules do not only reveal the activation times of the individual tasks, but further indicate their power dissipation over time (given by the height of the tasks). Such a power annotated schedule is referred to as power-profile. The main advantage of the shown transformation lies within the fact that it results in sequentially executing tasks on a single components, which is equivalent to the behavior of software tasks. Hence, a voltage scaling technique for software processors can be applied after the transformation, in order to exploit system idle times. The transformation is carried out in two main steps:

(a) A single power profile is derived by adding the power profiles of both hardware cores and by splitting this power profile into individual tasks whenever the power values change. In Figure 9(b) these points are t_{Sb} (the start time of task τ_1) and t_{Eb^2} (the end time of task τ_2). The power dissipation of tasks τ_a and τ_c are equivalent to the power of core 1, while task τ_b dissipates a power which is the sum of the power consumptions on core 0 and core 1.

(b) Further, for each outside data dependency (indicated as dashed arrow in Figure 9), the virtual power profile is split and additional tasks are introduced. For the given example, task τ_3 communicates to an outside task. Since the execution of this task lies within the virtual task τ_b, task τ_b is split into τ_b^1 and τ_b^2. In this way the outside communication can be correctly included. Tasks with deadlines are handled in a similar fashion, in order to avoid that tasks are extended beyond their timing constraints.



(a) Physical implementation (b) Virtual implementation (c) Virtual implementation

Fig. 9. DVS transformation for HW cores considering inter-PE communication

VII. Experimental Results

Based on the techniques and algorithms presented in this work, a multi-mode synthesis approach has been implemented on a Pentium III/1.2GHz Linux PC. In order to evaluate its capability to produce high quality solutions in terms of energy consumption, timing behavior, and hardware area requirements, a set of experiments has been carried out on 15 automatically generated multi-mode examples (`mul1`–`mul15`[1]) and one real-life benchmark example (`smart-phone`)[2]. All reported results were obtained by running the optimization processes 40 times and averaging the outcomes. The average power dissipations as well as the energy consumptions have been calculated according to Equation (1) to (4).

A. Automatically Generated Examples

Each of the 15 generated examples (`mul1`–`mul15`) is specified by 3 to 5 operational modes, each consisting of 8 to 32 tasks (required execution cycles vary between 500–350000). The used target architectures contain 2 to 4 heterogeneous PEs (clock frequencies are given in the range from 25 to 50MHz), some of which are DVS-enabled. These PEs are interconnected through 1 to 3 communication links. The active power consumption of programmable processors was randomly chosen between 5mW and 500mW, depending on the executed task. The power dissipation of hardware components are selected to be 1 to 2 orders of magnitude lower. Further, the static power dissipation was set to be 5 to 15% of the maximal active power. The execution probabilities of individual modes were randomly chosen and vary between 1% and 85%. Timing constraints have been assigned in the form of individual task deadlines as well as repetition periods to the modes (hyper-periods). The timing constraints were varied between 15ms and 500ms, such that schedulable implementations with up to 50% deadline slack could be found.

To illustrate the importance of taking mode execution probabilities into account during the synthesis process, an execution probability neglecting approach is compared with the proposed synthesis technique, which considers the mode probabilities. The first two sets of experiments demonstrate the energy savings achievable through the consideration of mode execution probabilities, either with or without the exploration of DVS. The third set examines the influence of the actual activation profile on the energy savings.

Comparisons excluding Dynamic Voltage Scaling: To highlight the influence of mode execution probabilities on the achievable energy saving, consider Table II which shows the multi-mode co-synthesis results for the 15 automatically generated benchmarks. The first three columns give the benchmark names, the hyper-period (repetition period) of each mode, and the mode execution probabilities. The fourth and the fifth column present the dissipated average power and optimization time for the execution probability neglecting synthesis approach. Note that the execution probabilities are neglected during the synthesis only, while the computed power dissipations at the end of the synthesis incorporate the execution probabilities, in order to ensure a meaningful comparison. The sixth and the seventh column show the same for the proposed approach, which considers the execution probabilities throughout the synthesis process. Take, for instance, example `mul6`. When ignoring the execution probabilities during the optimization, an average power dissipation of $1.677mW$ is achieved. However, optimizing the same benchmark example under the consideration that modes execute with *uneven* probabilities (e.g., 15:10:10:65 — i.e., mode 1 is active for 15%, mode 2 is active of 10%, and so on), the average power can be reduced by an appropriate task mapping and core allocation to $1.301mW$. This is a significant reduction of 22.46%. Furthermore, it can be observed that the proposed technique was able to reduce the energy consumption of all examples with up to 62.18% (`mul7`). Note that these reductions are achieved without any modification of the underlying hardware architectures, i.e., the system costs are not increased. It is also important to note that the achieved energy reductions are solely introduced by taking the mode execution probabilities into account during the co-synthesis process, i.e., both compared approaches allow the same resource sharing and rely on the same scheduling technique. When comparing the optimization times for both approaches, it can be observed that the proposed technique shows a slightly increased CPU time for most examples, which is mainly due to the more complex design space structure.

Comparisons including Dynamic Voltage Scaling: The next experiments were conducted to see how the proposed tech-

[1]The examples were generated with the publicly available tool TGFF [13].

[2]The used benchmarks, including the realistic smart phone example, can be found at: http://www.ida.liu.se/˜g-marsc/benchmarks/.

			w/o probabilities		with probabilities		
Example (No. of modes)	Hyper-period (ms)	Mode Execution Probabilities	Average power (mW)	CPU time (s)	Average power (mW)	CPU time (s)	Reduction (%)
mul1 (4)	70,60,90,20	5:10:75:10	8.131	20.7	7.529	24.7	7.29
mul2 (4)	50,70,40,80	10:5:80:5	3.404	15.5	2.771	18.2	18.61
mul3 (5)	20,24,60,40,30	7:3:80:5:5	10.923	23.4	10.430	23.0	4.17
mul4 (5)	60,30,70,40,50	1:4:5:40:50	7.975	21.0	6.726	25.2	15.50
mul5 (4)	20,60,60,40	7:13:35:45	5.186	18.4	4.668	22.1	10.01
mul6 (4)	65,40,40,100	15:10:10:65	1.677	20.6	1.301	19.9	22.46
mul7 (4)	200,160,190,100	5:5:5:85	3.306	11.6	1.250	21.4	62.18
mul8 (4)	400,70,40,80	75:5:15:5	1.565	32.1	1.329	28.0	15.06
mul9 (4)	40,40,100,40	7:3:80:10	3.081	6.0	1.901	5.8	38.28
mul10 (5)	500,70,500,80,70	45:5:40:5:5	1.105	28.3	0.941	32.1	14.83
mul11 (3)	100,120,200	80:10:10	2.199	9.3	1.304	16.6	40.70
mul12 (4)	80,80,90,150	15:10:50:25	7.006	25.4	5.975	34.2	14.69
mul13 (3)	80,60,100	5:15:80	4.090	15.8	2.816	15.8	31.04
mul14 (5)	60,30,60,100,190	5:5:10:10:70	8.195	28.6	6.466	33.0	21.13
mul15 (5)	220,150,60,250,200	5:10:75:7:3	2.188	41.5	1.222	55.4	44.16

TABLE II

CONSIDERING MODE EXECUTION PROBABILITIES (EXCLUDING DVS)

			w/o probabilities		with probabilities		
Example (No. of modes)	Hyper-period (ms)	Mode Execution Probabilities	Average power (mW)	CPU time (s)	Average power (mW)	CPU time (s)	Reduction (%)
mul1 (4)	70,60,90,20	5:10:75:10	4.271	526.6	3.964	768.6	10.92
mul2 (4)	50,70,40,80	10:5:80:5	1.568	860.4	1.273	687.4	18.82
mul3 (5)	20,24,60,40,30	7:3:80:5:5	4.012	1053.5	3.344	1192.2	16.66
mul4 (5)	60,30,70,40,50	1:4:5:40:50	2.914	1135.2	2.320	1125.4	20.39
mul5 (3)	20,60,60,40	7:13:35:45	1.394	967.7	1.315	932.1	5.68
mul6 (4)	65,40,40,100	15:10:10:65	0.689	472.9	0.465	593.7	32.53
mul7 (4)	200,160,190,100	5:5:5:85	1.331	540.3	0.479	820.7	64.02
mul8 (4)	400,70,40,80	75:5:15:5	0.564	1262.1	0.436	1412.0	22.64
mul9 (4)	40,40,100,40	7:3:80:10	0.942	161.2	0.648	177.1	34.66
mul10 (5)	500,70,500,80,70	45:5:40:5:5	0.480	1456.3	0.394	1361.9	17.88
mul11 (3)	100,120,200	80:10:10	0.396	318.1	0.255	403.2	35.53
mul12 (4)	80,80,90,150	15:10:50:25	2.857	1384.7	2.460	1450.7	13.91
mul13 (3)	80,60,100	5:15:80	1.185	498.3	0.953	576.6	19.56
mul14 (5)	60,30,60,100,190	5:5:10:10:70	2.320	1512.3	1.797	1556.4	22.55
mul15 (5)	220,150,60,250,200	5:10:75:7:3	0.801	1316.7	0.324	1836.3	59.56

TABLE III

CONSIDERING MODE EXECUTION PROBABILITIES (INCLUDING DVS)

nique compares to DVS and if further savings can be achieved by taking the mode probabilities and DVS simultaneously into account. Table III reports on the findings. The DVS technique that was used here is based on PV-DVS [28], which has been extended to enable the consideration of DVS not only for software processors, but also for parallel executing cores on hardware PEs (see Section VI-D). As in the first experiments, two approaches are compared here. The first approach disregards the mode execution probabilities during optimization, while the second takes them into account throughout the co-synthesis. Similar to Table II, the second and the third column of Table III show the results without consideration of execution probabilities, whilst the fourth and the fifth column present the results achieved by taking execution probabilities into account. Let us consider again benchmark `mul6`. Although the execution probabilities are neglected in the fourth column, a reduced average power consumption ($0.689mW$) can be observed, when compared to the results given in Table II. This clearly demonstrates the high energy reduction capabilities of DVS. Nevertheless, it is possible to further minimize the power consumption to $0.465mW$ by considering the execution probabilities together with DVS. This is an improvement of 32.53%, solely due to the synthesis for the particular execution probabilities. For all other benchmarks savings of up to 64.02% (`mul7`) were achieved. Due to the computation of scaled supply voltages and the influence of scheduling on the energy consumption, the optimization times are higher when DVS is considered.

Influence of Real Activation Probabilities: The next experiment is conducted to highlight the influence of the real user behavior on the energy efficiency of a system that has been synthesized under the consideration of certain mode execu-

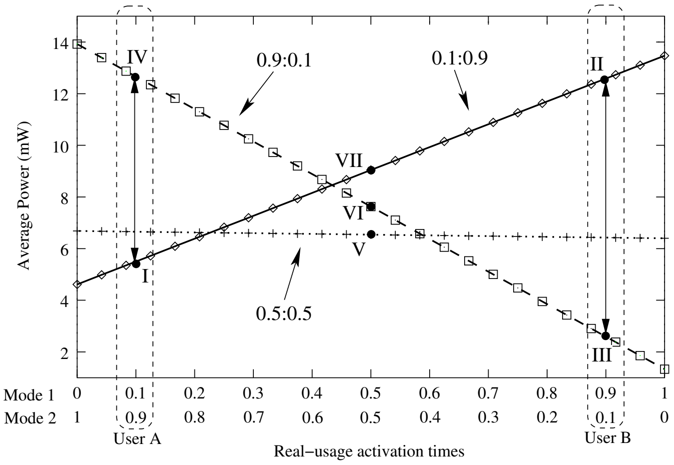


Fig. 10. System specification consisting of two operational modes optimized for different execution probabilities
(solid line–0.1:0.9, dashed–0.9:0.1, dotted–0.5:0.5)

tion probabilities. Certainly, the mode execution probabilities which are used during the synthesis represent an "imaginative" user and the activation probabilities of a *real* user will differ from those. In accordance, our experiments try to answer the following question: How is the energy efficiency affected by the different activation profiles during application run-time? For experimental purpose a simple specification with two modes is used, which contains 14 and 24 tasks (mode 1 and mode 2). The underlying architecture consists of two programmable processors and a single ASIC, all connected via a shared bus. This configuration was synthesised for three different pairs of execution probabilities (0.1:0.9, 0.9:0.1, and 0.5:0.5). These three implementation possibilities correspond to the three lines shown in Figure 10. All implementations are based on the same hardware architecture; yet, each has a different task and communication mapping, core allocation, as well as schedule. The first solution (solid line) was synthesized under the consideration of an execution probabilities 0.1:0.9, that is, it is assumed that mode 1 and mode 2 are active for 10% and 90% of the operational time, respectively. Similarly, the second (dashed line) and the third (dotted line) line represent solutions that have been synthesized using execution probabilities 0.9:0.1 and 0.5:0.5, respectively. According to the real execution probabilities during run-time, i.e., the activation behavior of the user, the average power dissipations of the implemented systems vary. Consider the system optimized for execution probabilities 0.1:0.9 (solid line). If the user behavior corresponds to these probabilities (User A), the system dissipates an average power of approximately $5.5mW$ (point I). However, if a different user (User B), for instance, uses mode 1 for 90% and mode 2 for 10% of the time (0.9:0.1), the system will dissipate approximately $12.5mW$ (point II). Nevertheless, if the system would be optimized for this activation profile (0.9:0.1), as indicated by the dashed line in Figure 10, a lower power dissipation of around $2.6mW$ (point III) can be achieved. Similarly, if the system is optimized for execution probabilities 0.9:0.1 (dashed line) and the user runs the application 10% in mode 1 and 90% in mode 2 (User A), then a power dissipation of $12.5mW$ (point IV) is given. While an optimization towards this usage profile can achieve a system implementation which dissipates only $5.5mW$ (point I), i.e., extending the battery-lifetime by a factor of 2.83 times. The dotted plot in Figure 10 represents the solution when the execution probabilities are neglected during the optimization, that is, the execution probabilities are considered to be equal for both modes. Of course, if the modes 1 and 2 are active for equal amounts of time, this solution achieves a lower power dissipation ($6.5mW$, point V) than the systems optimized for execution probabilities 0.1:0.9 ($9mW$, point VI) and 0.9:0.1 ($7.6mW$, point VII). The figure reveals that the design for 0.1:0.9 (solid line) achieves the lowest power dissipation when the user complies to an activation profile between 0:1 and 0.21:0.79. While the designs for 0.5:0.5 (dotted line) and for 0.9:0.1 (dashed line) lead to the lowest energy dissipation in the ranges from 0.21:0.79 to 0.57:0.43 and 0.57:0.43 to 1:0, respectively. In summary, Figure 10 clearly shows that the execution probabilities substantially influence the energy dissipation of the system. Certainly, the system should be optimized as close as possible towards the real behavior in order to achieve low energy consumptions, which, in turn, result in longer battery-lifetimes.

B. Smart Phone Benchmark

To further validate the co-synthesis technique in terms of real-world applicability, the introduced approach was applied to a smart-phone example. This benchmark is based on three publicly available applications: a GSM codec [3], a JPEG codec [4], and an MP3 decoder [18]. Accordingly, the smart-phone offers three different services to the user, namely, a GSM cellular phone, a digital camera, and an MP3-player. Of course, the used applications do not specify the whole smart-phone device, however, a major digital part of it. The specification for this example, given as operational mode state machine (OMSM), has already been introduced in Figure 2. For each of the eight operational modes, the corresponding task graphs have been extracted from the above given references. The individual applications have been software profiled to gather the necessary execution characteristics of each task. This was carried out by compiling profile information into the application [1], [2] and running the produced software on real-life input streams. On the other hand, the hardware estimations are not based on direct measurements, but have been based on typical values, such that hardware tasks execute 1 to 2 orders of magnitude faster and dissipated 1 to 2 orders of magnitude less power than their software counterparts [9]. Depending on the operational mode, the number of tasks and communications varies between 5–88 nodes and 0–137 edges, respectively. The hardware architecture of the embedded system within the smart phone consists of one DVS-enabled processor (execution properties are based on values given for the ARM8 developed in [8]) and two ASICs. These components are connected via a single bus. Tables IV and V give the results of the conducted experiments, distinguishing between optimizations without and with the consideration of DVS.

Similar to the previous experiments, approaches which neglect the execution probabilities are compared with the

				without probabilities		with probabilities	
Mode	No. of Tasks/Comm.	Exec. Prob.	Hyper-period (s)	E_O/HP_O (mJ)	$\bar{p}_O$ (mW)	E_O/HP_O (mJ)	$\bar{p}_O$ (mW)
0	88/137	0.09	0.020	0.2637	1.1868	0.1272	0.5723
1	12/0	0.74	1.000	0.8263	0.6115	0.8210	0.6075
2	12/0	0.01	1.000	1.7176	0.0172	1.7110	0.0171
3	5/4	0.02	0.250	1.3004	0.1040	0.9545	0.0764
4	12/5	0.02	0.500	1.6650	0.0666	1.3761	0.0550
5	17/16	0.10	0.025	0.1231	0.4922	0.0719	0.2874
6	17/16	0.01	0.025	0.2203	0.0881	0.3971	0.1588
7	12/5	0.01	0.500	1.7884	0.0358	1.3245	0.0265
Overall					2.6022		1.8011

TABLE IV
SMART PHONE EXPERIMENTS WITHOUT DVS

				without probabilities		with probabilities	
Mode	No. of Tasks/Comm.	Exec. Prob.	Hyper-period (s)	E_O/HP_O (mJ)	$\bar{p}_O$ (mW)	E_O/HP_O (mJ)	$\bar{p}_O$ (mW)
0	88/137	0.09	0.020	0.0746	0.3355	0.0786	0.3539
1	12/0	0.74	1.000	0.8190	0.6061	0.0180	0.0133
2	12/0	0.01	1.000	0.0280	0.0003	0.8110	0.0081
3	5/4	0.02	0.250	0.3355	0.0268	0.3545	0.0284
4	12/5	0.02	0.500	0.3556	0.0142	0.8412	0.0336
5	17/16	0.10	0.025	0.0513	0.2052	0.0813	0.3250
6	17/16	0.01	0.025	0.0492	0.0197	0.1975	0.0791
7	12/5	0.01	0.500	0.4917	0.0098	0.8671	0.0173
Overall					1.2176		0.8587

TABLE V
SMART PHONE EXPERIMENTS WITH DVS

introduced co-synthesis technique that considers the uneven activation times of different modes. Table IV shows this comparison for a fixed voltage system, i.e., no DVS is applied. The table provides information regarding all 8 modes of the smart phone. This mode information includes benchmark properties such as complexity, execution probability, and hyper-period. Furthermore, the table gives the achieve energy dissipation for the mode hyper-period and average power consumption of each mode. The average power consumption can be calculated from the energy values by dividing the energy by the hyper-period and multiplying the result with the execution probability. Synthesizing the system without consideration of execution probabilities results in an overall average power consumption of $2.6022mW$, when running the system after the synthesis according to the activation profile. Nevertheless, taking into account the mode usage profile during the co-synthesis this can be reduced by 30.76% to $1.8011mW$. Please note that the given overall average power consumption is calculated based on Equations (1)–(4); hence, these values are directly proportional to the battery-lifetime. The saving is achieved without the modification of the allocated hardware architecture, therefore, the system cost is the same for both solutions.

Also DVS has been applied to this benchmark, considering that the GPP of the given architecture supports DVS functionality. The results are shown in Table V. It can be observed that the overall average power consumption of the smart phone drops to $1.2176mW$, even when neglecting mode execution probabilities. However, the combination of applying DVS and taking execution probabilities into account results in the lowest power consumption of $0.8587mW$, a 29.5% reduction, when compared to the activation profile neglecting approach. That is, solely by considering the activation profile during the synthesis, the battery-lifetime could be extended by one third, even when using a system that employs DVS components. Overall, the average power is decreased from $2.602mW$ to $0.859mW$, which represents a significant reduction of nearly 67%. Regarding the required co-synthesis times, the four implementations could be found in $80.1s$ (without probabilities and DVS) to $4344.8s$ (with probabilities and DVS). Clearly, considering DVS requires longer optimization times due to the voltage scaling problem that needs to be solved repetitively within the innermost optimization loop of the co-synthesis algorithm. For instance, the optimization for DVS increases the run-time from $80.1s$ to $3754.1s$ for the case without consideration of execution probabilities, and from $96.9s$ to $4344.8s$ when execution probabilities are taken into account. On the other hand, the consideration of mode execution probabilities increases the optimization time only moderately form $80.1s$ to $96.9s$ in the case of no DVS, and from $3754.5s$ to $4344.8s$ if DVS is considered.

VIII. CONCLUDING REMARKS

We have introduced new techniques and algorithms for the energy minimization of multi-mode embedded systems. An abstract specification model called operational mode state machine has been proposed. This model allows for the specification of mode interaction (top-level finite state machine) as well as mode functionality (task graph). The advantage of such a representation is the capability to express the complete functionality of the system within a single model, containing both control and data flow.

The presented co-synthesis technique not only optimizes mapping and scheduling towards hardware cost and timing behavior, but also aims at the reduction of power consumption at the same time. A key contribution has been the development of an effective mapping strategy that considers uneven mode execution probabilities as well as important power reduction aspects, such as multiple task implementations and core allocation. For this purpose a GA-based mapping approach has been proposed along with four improvement strategies to effectively handle the optimization of component shutdown, transition time, area usage, and timing behavior. These improvement strategies guide the mapping optimization of multi-mode specifications towards high quality solutions in terms of power consumption, timing feasibility, and area usage. A newly introduced transformation-based algorithm for DVS-enabled hardware components, which handles parallel task execution, allows to leverage the efficiency of existing voltage scaling algorithms. This algorithm transforms a set of potentially parallel executing tasks on a single HW component into a set of sequential executing tasks, taking into account imposed deadlines and inter-PE communications.

The proposed techniques and algorithms have been validated through extensive experiments including a smart phone real-life example. These experiments have demonstrated that taking into account mode execution probabilities throughout the system synthesis leads to substantial energy savings compared to conventional approaches which neglect this issue. Furthermore, DVS has been considered in the context of multi-mode embedded systems and it was shown that considerably high energy reductions can be achieved by combining both the consideration of execution probabilities and dynamic voltage scaling.

ACKNOWLEDGMENTS

The authors are thankful to the anonymous reviewers for their valuable comments and suggestions that helped to improve this manuscript.

REFERENCES

[1] GNU CC Manual.
available at: http://gcc.gnu.org/.

[2] GNU gprof Manual.
available at: http://www.gnu.org/manual/gprof-2.9.1/gprof.html.

[3] GSM 06.10, Technical University of Berlin.
Source code available at http://kbs.cs.tu-berlin.de/∼jutta/toast.html.

[4] Independent JPEG Group: jpeg-6b.
Source code available at ftp://ftp.uu.net/graphics/jpeg/jpegsrc.v6b.tar.gz.

[5] Intel® XScale™ Core, Developer's Manual, December 2000. Order Number 273473-001.

[6] Mobile AMD Athlon™4, Processor Model 6 CPGA Data Sheet, November 2000. Publication No 24319 Rev E.

[7] Alexandru Andrei, Marcus T. Schmitz, Petru Eles, Zebo Peng, and Bashir M. Al-Hashimi. Overhead-Conscious Voltage Selection for Dynamic and Leakage Power Reduction of Time-Constraint Systems. In *Proceedings Design, Automation and Test in Europe Conference (DATE2004)*, pages 518–523, February 2004.

[8] Thomas D. Burd. *Energy-Efficient Processor System Design*. PhD thesis, University of California at Berkeley, 2001.

[9] Thomas D. Burd and Robert W. Brodersen. Processor Design for Portable Systems. *Journal on VLSI Signal Processing*, 13(2):203–222, August 1996.

[10] Anantha Chandrakasan, William J. Bowhill, and Frank Fox. *Design of High-Performance Microprocessor Circuits*. Wiley-IEEE Press, 2001.

[11] Anantha P. Chandrakasan and Robert W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publisher, 1995.

[12] Eui-Young Chung, Luca Benini, and Giovanni De Micheli. Contents Provider-Assisted Dynamic Voltage Scaling for Low Energy Multimedia Applications. In *Proceedings International Symposium Low Power Electronics and Design (ISLPED'02)*, pages 42–47, August 2002.

[13] R. Dick, D. Rhodes, and W. Wolf. TGFF: Task Graphs for free. In *Proceedings 5th International Workshop Hardware/Software Co-Design (Codes/CASHE'97)*, pages 97–101, March 1998.

[14] James Goodman, Anantha Chandrakasan, and Abram P. Dancy. Design and Implementation of a Scalable Encryption Procesoor with Embedded Variable DC/DC Converter. In *Proceedings IEEE 36th Design Automation Conference (DAC99)*, pages 855–860, 1999.

[15] Martin Grajcar. Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System. In *Proceedings IEEE 36th Design Automation Conference (DAC99)*, pages 280–285, 1999.

[16] T. Grötker, R. Schoenen, and H. Meyr. PCC: A Modeling Technique for Mixed Control/Data Flow Systems. In *IEEE European Design and Test Conference*, March 1997.

[17] Flavius Gruian and Krzysztof Kuchcinski. LEneS: Task Scheduling for Low-Energy Systems Using Variable Supply Voltage Processors. In *Proceedings Asia South Pacific - Design Automation Conference (ASP-DAC'01)*, pages 449–455, Jan 2001.

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

[19] Asawaree Kalavade and P. A. Subrahmanyam. Hardware/Software Partitiong for Multifunction Systems. *IEEE Transactions on Computer-Aided Design*, 17(9):819–836, Sep 1998.

[20] Jiong Luo and Niraj K. Jha. Battery-aware Static Scheduling for Distributed Real-Time Embedded Systems. In *Proceedings IEEE 38th Design Automation Conference (DAC01)*, pages 444–449, 2001.

[21] W. Namgoong, M. Yu, and T. Meng. A High-Efficiency Variable-Voltage CMOS Dynamic dc-dc Switching Regulator. In *Proceedings International Solid-State Circuits Conference*, pages 380–381, 1997.

[22] Hyunok Oh and Soonhoi Ha. A Static Scheduling Heuristic for Heterogeneous Processors. In *2nd International EuroPar Conference Vol. II*, August 1996.

[23] Hyunok Oh and Soonhoi Ha. Hardware-Software Cosynthesis of Multi-Mode Multi-Task Embedded Systems with Real-Time Constraints. In *Proceedings 2nd International Symposium Hardware/Software Co-Design (CODES'02)*, pages 133–138, May 2002.

[24] P. Pedro and Alan Burns. Schedulability Anaysis for Mode Changes in Flexible Real-time Systems. In *Proceddings Euromicro Workshop on Real-Time Systems*, pages 17–19, June 1998.

[25] Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles. Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems. In *Proceedings Design, Automation and Test in Europe Conference (DATE2002)*, pages 514–521, March 2002.

[26] Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles. Synthesizing Energy-Efficient Embedded Systems with LOPOCOS. *Kluwer Journal on Design Automation for Embedded Systems*, 6(4):401–424, July 2002.

[27] Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles. Iterative Schedule Optimisation for Voltage Scalable Distributed Embedded Systems. *ACM Transactions on Embedded Computing Systems*, 3(1):182–217, February 2004.

[28] Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles. Iterative Schedule Optimisation for Voltage Scalable Distributed Embedded Systems. *ACM Transactions on Embedded System Design*, 3(1):182–217, February 2004.

[29] L. Sha, R. Rajkumar, J. Lehoczky, and K Ramamritham. Mode Change Protocols for Priority-driven Preemptive Scheduling. 1:243–265, December 1989.

[30] Y. Shin, D. Kim, and K. Choi. Schedulability-Driven Performance Analysis of Multiple Mode Embedded Real-Time Systems. In *Proceedings IEEE 37th Design Automation Conference (DAC00)*, pages 495–500, June 2000.

[31] 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.

[32] Peng Yang, Paul Marchal, Chun Wong, Stefaan Himpe, Francky Catthoor, Patrick David, Johan Vounckx, and Rudy Lauwereins. Managing Dynamic Concurrent Tasks in Embedded Real-Time Multimedia Systems. In *Proceedings International Symposium System Synthesis (ISSS'02)*, pages 112–119, October 2002.

Marcus T. Schmitz received the Dipl.-Ing. (FH) degree in electrical engineering from the University of Applied Science Koblenz, Germany, in 1999, and the Ph.D. degree in electronics from the University of Southampton, United Kingdom, in 2003.

Since April 2003, he is a postdoctoral researcher in the Embedded Systems Laboratory, Department of Computer and Information Science, Linköping University, Linköping, Sweden, where he is working on the development of off-line and on-line voltage scaling techniques for embedded systems. His research interests include system-level co-design, application-driven design methodologies, energy-efficient system design, and reconfigurable architectures. Dr. Schmitz has published several papers in the area of dynamic voltage scaling for heterogeneous embedded systems and he has co-authored the book "System-Level Design Techniques for Energy-Efficient Embedded Systems" (Kluwer Academic, 2004).

Bashir M. Al-Hashimi (M' 99-SM' 01) received the B.Sc. degree with 1st-class classification in Electrical and Electronics Engineering from the University of Bath, UK, in 1984, and the Ph.D. degree from York University, UK, in 1989. Following this he worked in industry for six years designing high performance chips for analog and digital signal processing applications, developing CAD tools for simulation and synthesis of analogue and digital circuits. In 1995, he joined Staffordshire University, UK, where he formed the VLSI Signal Processing research group with funding from industry and government. In 1999, he joined the Department of Electronics and Computer Science, Southampton University, UK, where he is currently a Professor of Computer Engineering. He has authored one book on SPICE simulation, (CRC Press,1995), and coauthored two books, "Power Constrained Testing of VLSI circuits" (Kluwer Academic, 2002), and "System-Level Design Techniques for Energy-Efficient Embedded Systems" (Kluwer Academic, 2004). He has authored and coauthored over 125 technical papers. His research and teaching interests include low-power hardware-software co-design, System-on-chip test, and VLSI CAD.

Dr Al-Hashimi is the Editor-in-Chief of the IEE Proceedings: Computers and Digital Techniques, and a Member of the Editorial Board of the Embedded Computing Journal. He has served on the IEEE/ACM Design Automation and Test in Europe (DATE) conference in various capacities including Topic Chair (Simulation and Emulation, 2001), and for 2005 as a Member of the Executive Committee. He is a Member of the Technical Programe Committee of the IEEE European Test Symposium and the IEEE VLSI Test Symposium. He has served as the Guest Editor for a number of journal special issues; the most recent are "UK Embedded System Design", (Design Automation for Embedded Systems Journal, Kluwer, 2002), and "Low Power SoC", (IEE Proceedings: Computers and Digital Techniques, 2002). Dr Al-Hashimi was the co-author of a Best Paper Award at the 2000 IEEE International Test Conference relating to low power BIST for RTL data paths.

Petru Eles (M'99) received the M.Sc. degree in computer science from the "Politehnica" University of Timisoara, Romania, in 1979, and the Ph.D. degree in computer science from the "Politehnica" University of Bucharest, Romania, in 1993.

He is currently a Professor with the Department of Computer and Information Science at Linköping University, Sweden. His research interests include design of embedded systems, hardware/software co-design, real-time systems, system specification and testing, CAD for digital systems. He has published extensively in these areas and has coauthored several books among which "System Synthesis with VHDL" (Kluwer Academic, 1997) and "System-Level Design Techniques for Energy-Efficient Embedded Systems" (Kluwer Academic, 2004).

Prof. Eles was a corecipient of best paper awards at the 1992 and 1994 European Design Automation Conference (EURO-DAC), and of the best presentation award at the 2003 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis.

Petru Eles is an Associate Editor of the IEE Proceedings - Computers and Digital Techniques. He has served as a Program Committee member for numerous International Conferences such as DATE, ICCAD, ECRTS, CASES, EMSOFT, RTSS, and as Technical Program Chair of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS).