Synthesising Energy-Efficient Embedded Systems with LOPOCOS

Marcus T. Schmitz[1], Bashir M. Al-Hashimi[1], Petru Eles[2]

[1]Electronic Systems Design Group

Department of Electronics and Computer Science

University of Southampton, Southampton, UK

{m.schmitz, bmah}@ecs.soton.ac.uk

[2]Department of Computer and Information Science

Linköping University

S-58183 Linköping, Sweden

petel@ida.liu.se

Abstract

In this paper, we introduce the LOPOCOS (Low Power Co-synthesis) system, a prototype CAD tool for system level co-design. LOPOCOS targets the design of energy-efficient embedded systems, implemented as heterogeneous distributed architectures. In particular, it is designed to solve the specific problems involved in architectures that include dynamic voltage scalable (DVS) processors. The aim of this paper is to demonstrate how LOPOCOS can support the system designer in identifying energy-efficient hardware/software implementations for the desired embedded systems. Hence, highlighting the necessary optimisation steps during design space exploration for DVS enable architectures. The optimisation steps carried out in LOPOCOS involve component allocation and task/communication mapping as well as scheduling and dynamic voltage scaling. LOPOCOS has the following key features, which contribute to this energy efficiency. During the voltage scaling valuable power profile information of task execution is taken into account, hence, the accuracy of the energy estimation is improved. A combined optimisation for scheduling and communication mapping based on genetic algorithm, optimises simultaneously execution order and communication mapping

towards the utilisation of the DVS processors and timing behaviour. Furthermore, a separation of task and communication mapping allows a more effective implementation of both task and communication mapping optimisation steps. Extensive experiments are conducted to demonstrate the efficiency of LOPOCOS. We report up to 38% higher energy reductions compared to previous co-synthesis techniques for DVS systems. The investigations include a real-life example of an optical flow detection algorithm.

1 Introduction and Previous Work

Embedded systems have become omnipresent in wide variety of applications, such as telecommunication systems, consumer electronics, and other mass products. These computing sub-systems are responsible for control and data operations and are commonly implemented as an architectural mix of several processing elements (PEs), such as programmable microprocessors, ASIPs, ASICs, and FPGAs. The processing elements are connected through communication links (CLs) and form a heterogeneous distributed system. Today, the designers of these modern embedded systems are facing numerous challenges, mainly arising from two important facts. Firstly, the ever increasing demand for functionality is continuously enlarging the product complexity. Secondly, the intense burdens imposed by a highly competitive market, which has led to tight time-to-market windows and put rigorous constraints on the production cost. To overcome such difficulties, the automated design of these, mostly, mixed software and hardware systems is an inevitable necessity. System level co-design is a methodology helping the system designer to identify most suitable architectures by supporting a rapid design space exploration. This is done without actually implementing the application, but by means of estimating the design quality and its properties, using a CAD tool. Co-synthesis can therefore be viewed as the computer aided process to design embedded computing systems which consist of software executing on an underlying hardware architecture. The co-ordinated optimisation of both system parts (hardware and software) is the primary goal of this synthesis process. The optimisation itself can be driven by important design objectives, such as performance, cost, and power consumption. However, most previous co-design approaches have neglected issues related to power [15, 19, 23, 34, 42] or focused on distributed systems that exclude DVS processing elements [9, 13, 24], hence, leaving a major source of power reduction unexploited.

Nevertheless, during the last decade power has become a main design issue of concern, due to the proliferation of battery powered embedded systems which demand a cautious use of the available energy resources. One

way to tackle the problem of reducing the energy dissipation is the usage of system level power management techniques to trade-off system performance against power dissipation. One of the most promising techniques is dynamic voltage scaling (DVS). By conjointly reducing supply voltage and operational frequency, DVS is able to reduce the energy consumption significantly. Since the reduction of the operational frequency is equivalent to the reduction of the system performance, this technique is applicable to applications where the system schedule shows periods of idleness or periods where a reduced performance can be tolerated. The field of DVS finds it roots in [41], where its usability was demonstrated considering a desktop computer environment. The dynamical and conjoint adjustment of supply voltage and operational frequency, to satisfy the application needs, was shown to lead to superior power savings compared to dynamic power management (switching off of idle components), when both techniques are applicable [20]. This fact explains why DVS is attracting considerable attention from both academia and industry. Most major digital processor vendors have recently introduced DVS enabled processor types [25, 3, 2]. Many research has focused on the scheduling aspects for DVS, however, often making the assumption of single processor systems [22, 20, 38, 27]. The trends towards co-design methodologies and dynamic voltage scalable processors indicate the need for co-design techniques which support the consideration of DVS processors to synthesise energy-efficient embedded systems. Such a co-design framework will be presented in this paper.

Three research groups have addressed issues which have a close relationship to the problems solved in the LOPOCOS system. Luo and Jha [29] have extended an existing co-synthesis approach [11] to account for DVS processing elements. Their approach is based on a DVS algorithm which keeps communication events fixed, i.e., they reduce the global optimisation problem to smaller local problems which can be solved easier and faster. Furthermore, they take battery characteristic into consideration [30] to increase the battery life time of the system. The approach is very efficient in terms of optimisation time and achieved energy reduction, when the used processing elements show similar power consumption. However, in the case that the architecture is built out of highly heterogeneous PEs, this approach is likely to find sub-optimal solutions, from the DVS point of view, since the fixing of communication events neglects the different power profiles during the schedule and mapping optimisation.

Gruian and Kuchcinski [18] presented an approach based on an energy conscious list scheduling technique. The dynamically re-calculated task priorities are based on the energy gain inducted by scheduling decisions and the critical path of the task graph. However, since task priorities based on energy gain might lead the list scheduler to infeasible (timing violating) schedules, the task priorities are calculated as a weighted sum of energy gain and critical path priorities. A good trade-off factor is found by increasing the weight of the tasks on the critical path and re-scheduling the tasks until a feasible schedule has been found.

Bambha et al. [5] introduced a hybrid global/local search strategy based on *simulated heating* to find settings for the DVS processing elements. Their approach is split into two optimisations. The global optimisation is carried out using a genetic algorithm, while the local optimisation is based on a hill climbing and a Monte Carlo search. This work mainly focuses on the management of computational resources when using hybrid algorithms, and its iterative nature (up to 20 minutes for the voltage scaling of a task graph with 21 nodes) makes it too time consuming to be used in the innermost loop of a co-synthesis system. As opposed to the presented LOPOCOS system, the approaches above are based on constructive scheduling techniques and neglect the power profile information during the optimisation process.

One of the aims of this paper is to comprehensively introduce LOPOCOS, an experimental low power co-synthesis system, and to show how it can be used by system designers to synthesise energy-efficient distributed embedded systems containing DVS-PEs. The energy reduction in LOPOCOS is based on the utilisation of a generalised DVS algorithm, which is used to guide the optimisation of the system schedule and the task/communication mapping. However, in this paper, we will particularly focus our attention on an energy-efficient mapping approach for DVS. We demonstrate that significant energy savings can be achieved by appropriately choosing the mapping for the task executions, such that DVS can be exploited effectively. Additionally, it is shown that the component allocation plays an important role during the optimisation of DVS enabled systems, not only in terms of performance and production cost, but also in terms of energy dissipation. We conducted extensive experiments to demonstrate the ability of the proposed synthesis approach to find solutions of high quality in terms of cost, feasibility, and energy consumption.

The paper is organised as follows. Section 2 introduces our co-synthesis system (LOPOCOS) and highlights the important synthesis steps. Experimental results, with emphasis on the optimisation of mapping and allocation, are presented in Section 3. Finally, in Section 4, we draw the conclusions.

2 The LOPOCOS System

In this section, we will introduce the co-design flow as used in LOPOCOS (see Fig. 1). We will mainly focus here on the mapping and component allocation steps (indicated as Step 1, 2, and partly 3, in Fig. 1), while detailed descriptions of the used algorithms for voltage scaling and scheduling can be found elsewhere [35, 37].

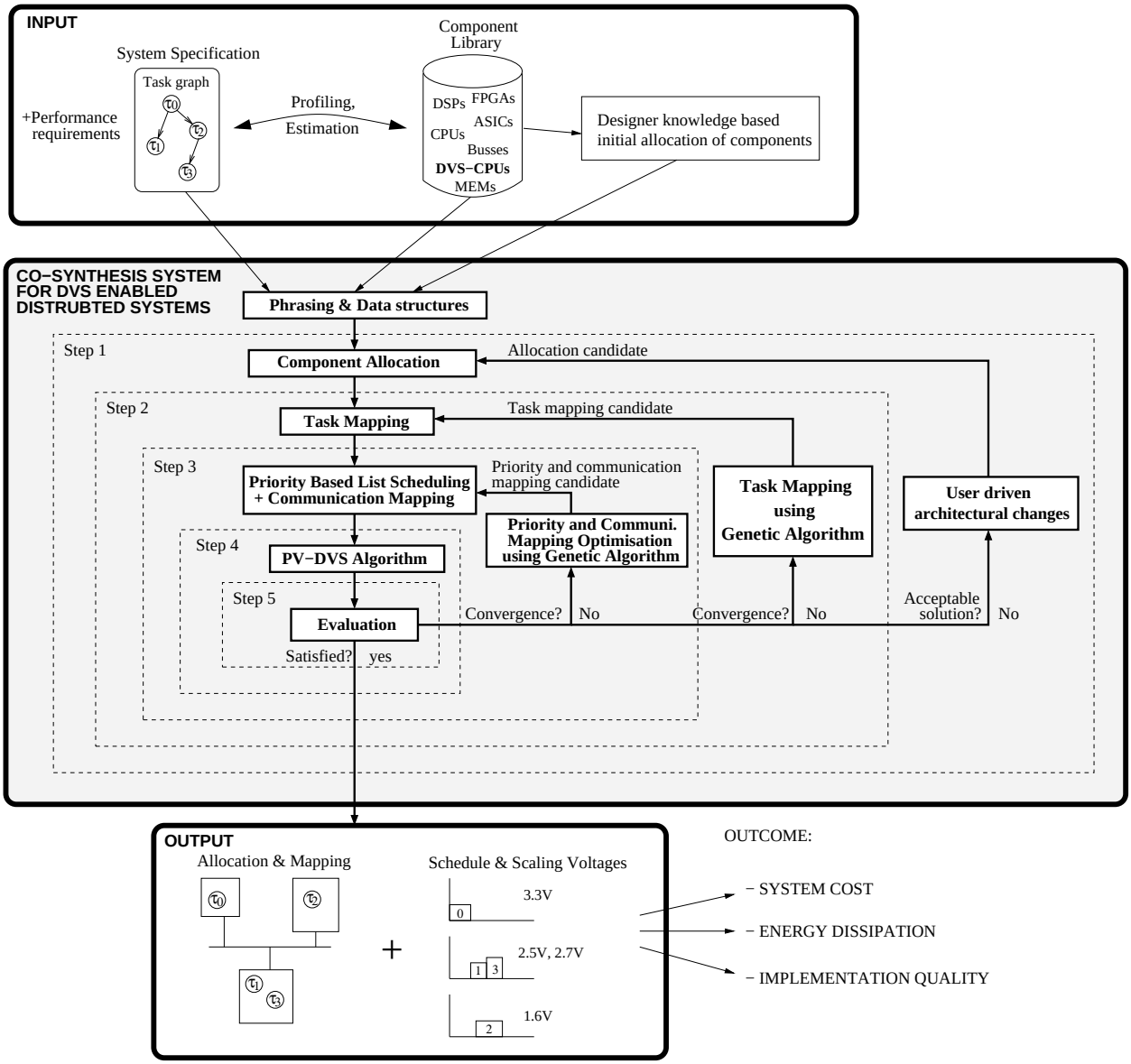


Figure 1: Co-synthesis flow in LOPOCOS

2.1 System Specification and Target Architecture

The synthesis process starts from a system specification given as multi-rate hyper task graph $G_S(\mathcal{T}, \mathcal{C})$, a combination of several smaller task graphs, capturing all task activations for the hyper-period (LCM of all graph periods). Such a task graph example is shown in Fig. 2(a). Each node $\tau \in \mathcal{T}$ in these graphs represents a task, an atomic unit of functionality to be executed without preemption. A node might inherit an earliest possible start time η which needs to be exceed before the task can be executed, and a hard deadline θ which must be met at run-time in order to ensure correct functionality. Edges $\gamma \in \mathcal{C}$ in the task graph denote precedence constraints and data dependencies between tasks. If two tasks, τ_i and τ_j, are connected by an edge, then the execution of task τ_i must be finished

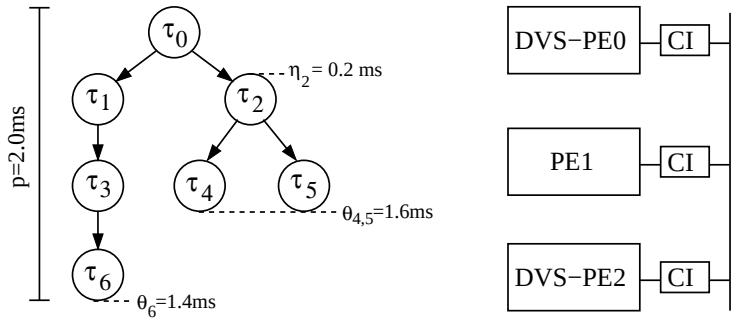


Figure 2: Task graph and target architecture example

before task τ_j can be started. Data dependencies inherit a data value, reflecting the quantity of information to be exchanged by two tasks. Further, each task graph has a specific period p, representing the time limit between two successive invocations. An implementation is only feasible when all timing and precedence constraints are fulfilled. Such a specification model is mostly suitable for data-flow intensive applications.

The architectures we consider here consist of heterogeneous processing elements (PEs), like general purpose processors, ASIPs, FPGAs, and ASICs. These components *include* state-of-the-art DVS-PEs. Furthermore, the PEs might employ lower level power management techniques, like gated clocks [6], to avoid switching of unused circuitry during periods of idleness. An infrastructure of communication links, like buses and point-to-point connections, connects these PEs. In addition, DVS-PEs are connected through communication interfaces (CIs), able to adapt to the different operational frequencies caused by scaling the DVS-PEs. An example architecture is shown in Fig. 2(b). The architecture is captured using a directed graph $G_A(\mathcal{P}, \mathcal{L})$ where nodes $\pi \in \mathcal{P}$ represent PEs and edges $\lambda \in \mathcal{L}$ denote CLs.

Each task of the system specification might have multiple implementation alternatives, therefore, it can be potentially mapped to several PEs able to execute this task. If two communicating tasks are accommodated on different PEs, π_n and π_m with $n \neq m$, then the communication takes place over a CL, involving a communication time and power overhead. For each possible task mapping certain implementation properties, like e.g. execution time, dynamic power dissipation, memory, and area requirements, are given in a technology library. These values are either based on previous design experience or on estimation techniques, such as those presented in [40, 28, 7].

2.2 Co-Synthesis Problem

The co-synthesis problem for heterogeneous distributed systems including DVS components can be decomposed into five synthesis steps (see also Fig. 1):

(1) *Allocation:* This step determines the quantities and types of the used system components (processing elements and communication links). In our particular case this includes also PEs that employ DVS technology.

(2) *Mapping:* This step defines the spatial assignment of computational tasks onto PEs and communications onto CLs. Mapping implicitly decides if a task is implemented as software running on a programmable processor (general purpose CPU or DSP), or as hardware on an ASIC or FPGA.

(3) *Scheduling:* According to the data dependencies and timing constraints, the execution of the system tasks needs to be scheduled, hence, the start time of each task and communication needs to be specified. Tasks executing on programmable processors have to be sequentialised, while tasks implemented in hardware can be performed in parallel.

(4) *Voltage scaling:* In order to reduce the energy consumption, the tasks mapped to DVS-PEs can be executed at a lower speed (lower supply voltage), whenever the schedule allows such an extension of the execution time without violating any precedence and timing constraints.

(5) *Evaluation:* In order to judge the quality of the implementation alternative, which is given by the first four steps, it is necessary to estimate certain implementation properties, such as timing behaviour, power consumption, production cost, etc. According to these estimates, which should be as accurate as possible, it is possible to guide an optimisation process.

The system designer is interested in deriving the most suitable system implementation for a given specification, i.e., he needs to find a system architecture, a mapping of tasks and communications onto the architecture, a schedule for the mapped tasks, and a dynamic voltage setting for the DVS-PEs. Of course, the most suitable implementation shows low system cost, low energy dissipation, and must be feasible in terms of performance.

The combined scheduling and mapping problem for energy reduction and timing feasibility can be formulated using the common triplet notation for scheduling problems [33]. Our problem is then described by $Q_m|prec|\theta_j, f_A, \sum E_j^s$, where Q_m specifies a multiprocessor environment, *prec* refers to a task model with precedence constraints, θ_j and f_A are objectives capturing the deadline and area constraints, respectively. $\sum E_j^s$ denotes the additional objective to minimise the energy dissipation based on DVS. Therefore, the synthesis problem for DVS is to find an arrangement of the task execution order and mapping, such that the energy reduction through DVS-PEs is maximised and all specification constraints (timing, precedence, area, etc.) are met. A more detailed description of the synthesis problem, including the DVS problem, can be found in [36].

In this work, we make the assumption that the specified tasks are of a coarse granularity and that the PEs can continue operation during the voltage scaling (as is the case for the DVS processor in [8]). This allows us to neglect the scaling overhead in terms of power consumption and time.

2.3 Evaluation

This section corresponds to Step 5 in Fig. 1. Each implementation candidate, resulting after a sequence of allocation, mapping, scheduling, and voltage scaling steps, can be characterised in terms of certain implementation properties, which reflect the design quality. In order to guide the optimisation process towards solutions of high quality, it is necessary to find a mathematical formulation in terms of an objective functions. This is of particular importance when using iterative improvement heuristics, such as genetic algorithms, simulated annealing, or tabu search.

LOPOCOS targets primarily four objectives: (a) system cost, (b) area, (c) timing, and (d) energy consumption. After each synthesis step one of these parameters can be derived. The dynamic voltage scaling allows to calculate the energy dissipation based on the found scaling voltages. The timing feasibility can be checked after the schedule has been determined. The mapping is responsible for the used area in terms of bytes and gates (whether implemented as software or hardware) and the allocation of components decides upon the system cost. We will elaborate each of these parameters in the following sections.

2.4 Voltage Scaling

This section briefly introduces the dynamic voltage scaling technique used in LOPOCOS, as carried out in Step 4 in Fig. 1. Dynamic voltage scaling reduces the dynamic power dissipation P_{dyn} of a digital circuit by reducing the supply voltage V_{dd} during the run-time of the application. Since the supply voltage influences the dynamic power quadratically, as shown in Equation (1), voltage scaling provides a significant potential for power reduction.

$$P_{dyn} = C_L \cdot N_{0\to1} \cdot f \cdot V_{dd}^2 \tag{1}$$

where C_L denotes the load capacitance of the circuit, $N_{0\to1}$ represents the zero-to-one switching activity, and f is the operational frequency. However, reducing the supply voltage does not come without a penalty, it increases the circuit delay d, which, in turn, necessitates the lowering of the operational frequency f to guarantee correct function of the circuit:

$$d = k \cdot \frac{V_{dd}^2}{(V_{dd} - V_t)^2} \tag{2}$$

where k is a circuit dependent constant and V_t denotes the circuit threshold voltage. Furthermore, the energy consumption E_{dyn} is given as the integral of the power dissipation P_{dyn} over time t, and since the time is inverse proportional to the operational frequency f, the energy can be expressed using the following equation:

$$E_{dyn} = N_{0\to1} \cdot C_L \cdot V_{dd}^2 \tag{3}$$

It is important to observe that the energy consumption solely depends on the switched load capacitance $N_{0\to1} \cdot C_L$ and the squared supply voltage. Therefore, considering the switched load capacitance as given by technology and performed computation, and that no switching occurs after the computation has been executed (gated clocks), the *only* way to reduce the consumed energy is to lower the supply voltage.

In LOPOCOS, the voltage scaling is performed for each implementation alternative to estimate its energy consumption for the DVS enabled architecture (Fig. 1, Step 4). The voltage scaling algorithm, called PV-DVS, utilises an energy gradient based method, which is explained in [35]. The produced voltage settings are static, hence, no calculations have to be performed at run-time, in order to determine the voltage levels. This avoids both timing and energy overhead due to such calculations. Of course, this makes the approach more suitable for applications where the actual execution time do not vary widely from the estimated worst case execution time, such as transformational computations on fixed size data sets. By taking into account the particular power dissipated by each task (the power profile of the tasks), this voltage scaling allows to increase the accuracy of the estimated energy dissipation, leading to an improved accuracy of the co-synthesis process.

Fig. 3 describes the functionality of the voltage scaler, considering the example task graph and the target architecture in Fig. 2. The voltage scaling algorithm gets as input a mapped task graph and a schedule corresponding to the task executions at nominal supply voltage V_{max} (Fig. 3(a)). After passing this information through the voltage scaling algorithm, the voltage settings for tasks τ_4 and τ_5 have been reduced, to exploit the available slack. The schedule in Fig. 3(b) therefore shows a reduced energy consumption. Of course, the task executions can only be extended, which corresponds to lowering the supply voltage, as long as deadlines (dashed lines in Fig. 3) are met. For clarity reasons we have disregarded the communications in this example, though LOPOCOS considers them during the co-synthesis process.

After identifying an effective setting for the supply voltages we can calculate the total energy consumption E_Σ of an implementation:

$$E_\Sigma = \sum_{\varepsilon \in \mathcal{A}} E(\varepsilon) \tag{4}$$

where $\mathcal{A} = \mathcal{T} \cup \mathcal{C}$ defines the set of all activities including tasks and communications. Based upon the type of

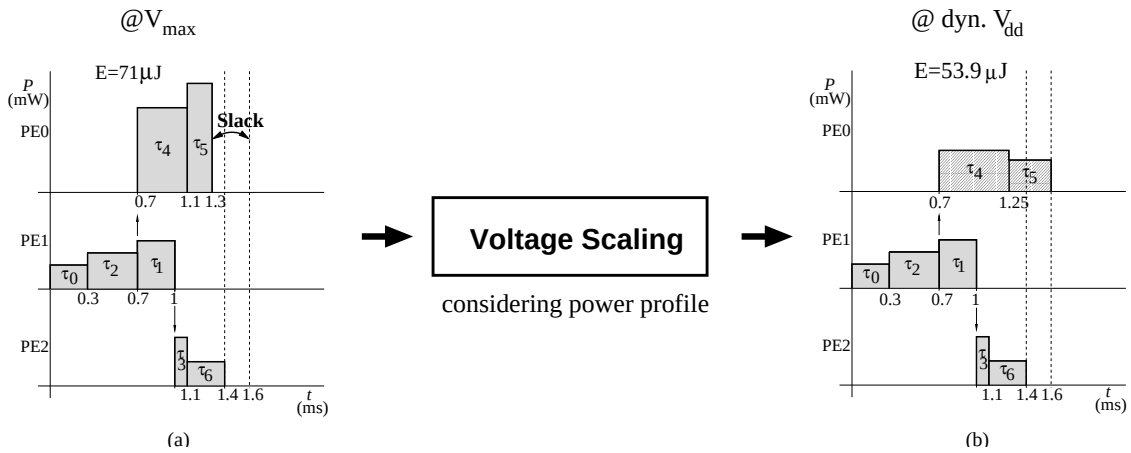


Figure 3: Voltage scaling: (a) system schedule at nominal supply voltage and (b) system schedule with dynamically changing supply voltages.

activity, the energy dissipation $E(\varepsilon)$ is calculated from:

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

where P_{max} and t_{min} refer to the power dissipation and execution time at nominal supply voltage, respectively, V_{dd} is the scaled supply voltage, $\mathcal{T}_{\text{DVS}}$ is the set of all tasks mapped to DVS-PEs, and P_C and t_C denote the power dissipation and execution time of communication activities. The total energy consumption is one of the objectives which needs to be minimised, hence, the remaining optimisation steps (scheduling, mapping, allocation) will be partially based on this value. For example, the scheduling step optimises both timing feasibility and energy reduction.

2.5 Energy-Efficient Scheduling

This section will focus on the schedule optimisation, indicated as Step 3 in Fig. 1. In traditional co-synthesis environments, scheduling optimisation is solely carried out to achieve timing feasibility. However, in the presence of system level power management techniques (PM), e.g., dynamic power management (DPM) and dynamic voltage scaling (DVS), scheduling becomes also important from the energy point of view, since it has an influence on the system idle and slack times which are utilised by the PM.

The scheduling optimisation in LOPOCOS is carried out by a genetic list scheduling approach, similar to [10, 17], and it produces a static schedule. Unlike commonly used list scheduling techniques [4], which rely

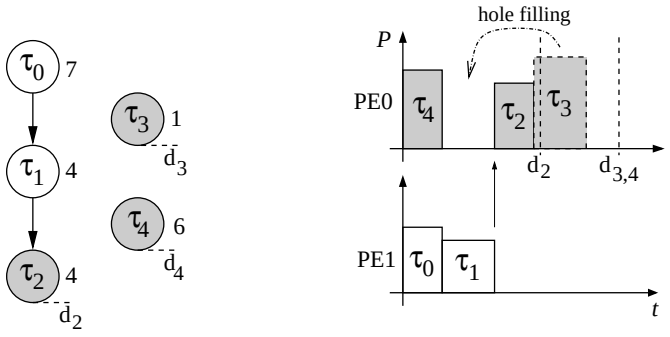


Figure 4: Hole filling problem

on sophisticated algorithms to calculate the task priorities, genetic list scheduling algorithms (GLSA) employ a genetic algorithm (GA) to iteratively improve these priorities. Literature about the functionality of genetic algorithms can be found in [16, 31]. The main advantage of such a strategy is the fact that the optimisation can be based on any arbitrary objectives, capturing different synthesis goals (timing, area, power, etc.). Although the GLSA presented in [10, 17] provide a valuable basis for our approach, they use list schedulers which are rather unsuitable for the specific problem we address here, namely reducing the energy consumption while, at the same time, finding feasible schedules. Advanced LS algorithm, like the one in [17] use a so called "hole filling" technique to minimise delays. Such an approach can lead to bad quality solutions when applied in the contents of DVS. The following example explains the reason for this.

Consider the task set given in Fig. 4, mapped onto an architecture build out of two DVS-PEs. The task priorities are given on the right side of each task. According to these priorities a feasible schedule, as shown in Fig. 4, can be generated by a list scheduler. We can observe that the tasks τ_0, τ_1, and τ_2 can be scaled only a little until deadline d_2 is met. However, task τ_4 can utilise the idleness before task τ_2 starts execution, and task τ_3 can be scaled until the deadline $d_{3,4}$ is met. Let us consider now the employment of a hole filling technique. In this case the list scheduler would try in its last scheduling step to place task τ_3 into the idle period between task τ_4 and task τ_2. This decision is fatally wrong, from an energy reduction point of view, since the only available slack for all tasks would be the time between the execution of task τ_2 and the deadline d_2. However, if the deadline $d_{3,4}$ would be identical with deadline d_2, then the schedule displayed in Fig. 4 would become infeasible and the schedule produced with hole filling would be the better one. To avoid such a dilemma our list scheduler solely makes scheduling decisions based on the task priorities, which are iteratively optimised. Therefore, it is capable of producing both schedule variants discussed above and evaluates their suitability.

The objective of the scheduling optimisation is to yield low energy consumption, while, at the same time, achieve timing feasibility. In order to calculate the objective that guides the optimisation towards timing feasibility,

we employ the following penalty function:

$$p_t = 1 + \frac{\sum_{\tau \in \mathcal{T}_d} DV_\tau^2}{T_{HP}^2} \tag{5}$$

where $\mathcal{T}_d$ denotes the set of all deadline tasks, $t_F(\tau)$ and $t_d(\tau)$ represent the finishing time and deadline of task τ, respectively. The deadline violations are captured by $DV_\tau = \max\big(0, t_F(\tau) - t_d(\tau)\big)$. The hyper task graph period T_{HP} is used to relate the deadline violations and squaring has been applied, in order to assign a higher penalty to larger violations of imposed deadlines. The fitness of each schedule F_S is then simply calculated by multiplying the energy dissipation E_Σ (see Section 2.4, Equation (4)) with the timing penalty p_t. More details on our scheduling algorithm, called energy-efficient genetic list scheduling (EE-GLSA), can be found in [37].

2.6 Task and Communication Mapping for Energy Reduction

The previously introduced DVS technique (Section 2.4) and the schedule optimisation (Section 2.5) are an incremental part of LOPOCOS, and are carried out for each task mapping candidate. This is shown in Fig. 1. Nevertheless, task and communication mapping are two separate optimisation steps in our synthesis approach. For clarity reasons we first show how the task mapping is optimised and then introduce the communication mapping.

2.6.1 Task Mapping

The task mapping step determines which PE carries out which task. Thereby, it determines the execution time and power dissipation at nominal supply voltage and further the area requirement in terms of bytes or gates, depending on whether a task is implemented as software or hardware. The goal of the mapping optimisation step is to distribute the tasks among the available PEs, including DVS enabled PEs, such that the energy dissipation is minimised and a feasible design in terms of timing behaviour and area constraints is achieved. Since the schedule has a significant impact on the DVS usability, also the mapping decisions influence how well DVS can exploit the idle times. Clearly, different mappings result in different schedules, due to the effects on task execution times, inter task communication times, and exploration of task parallelism. Simply spoken, scheduling and mapping are interrelated.

We have extended a GA based task mapping algorithm similar to the one given in [13], such that it solves the specific problems identified in Step 2 of the design flow shown in Fig. 1. This extension is based on our PV-DVS and the GA list scheduling algorithm [35, 37], which are used to calculate parts of the mapping fitness function. Thereby, it is possible to integrate the proposed DVS optimised scheduling algorithm into the mapping step of the

design flow, in order to guide the optimisation.

In GA based mapping approaches, solution candidates are encoded into mapping strings. Each gene in these strings describes a mapping of a task to a PE. A genetic algorithm evolves an initial mapping population (several mapping candidates) by imitating and applying the principles of natural selection. The evolution is based on mating the fittest individuals of the current population through the usage of crossover operations to generate offsprings with a potentially higher quality. Additionally, mutation provides the opportunity to enter unexplored regions of the search space by applying random changes to the genes of an individual. A description of the presented energy-efficient genetic mapping algorithm (EE-GMA) is given in Fig. 5.

The fitness F_A of mapping candidates (Equation (6)) is calculated with respect to an additional objective, namely area. The fitness is expressed by:

$$F_A = F_S \cdot \prod_{\pi \in \mathcal{P}} AP_\pi \tag{6}$$

$$AP_\pi = \begin{cases} 1 & \text{if} \quad AA_\pi \geq SA_\pi \\ k \cdot \left(\frac{SA_\pi}{AA_\pi} - 1 \right) + 1 & \text{otherwise} \end{cases} \tag{7}$$

where F_S is the schedule fitness based on the DVS reduced energy dissipation including a time penalty, as briefly outlined in Section 2.5, and AP_π assigns an area penalty for each PE exceeding its area constraints, as given in Equation (7). The used area is denoted as SA_π, and the maximal available area is represented by AA_π (either as memory or silicon area depending on the implementation in SW or HW). If the available area AA_π is not exceeded, we do not need to assign an area violation penalty for the particular processing element π, hence, F_S is multiplied by one. On the other hand, if the area constraint is exceeded, the used area SA_π and the available area AA_π are related and multiplied by a constant k. This constant allows to adjust the aggressiveness of the penalty. We set k to 0.02 (empirically found to be a good value) in our experiments, which was sufficiently high to avoid infeasible results at the end of the mapping optimisation. However, it is still low enough to allow infeasible solutions to survive sometimes in order to increase the population diversity and to avoid a premature convergence of the GA. In this way, it is possible to stimulate the placement of functionality onto the distributed PEs such that energy is minimised, while timing and area constraints are respected. The parameters of the GA for the task mapping were set as follows: The population size was set to 50, the minimal dynamic mutation probability was 5%, the generational gap was 20% and the initial population pool was filled with random mappings.

EE-GMA

Input: - task graph TG
- technology library (execution times, power dissipations)
- allocated architecture

Output: - timing, area, and energy optimised mapping

01: **Initialisation:** Create random initial population pool P_m of mapping strings.
02: **Perform Mapping:** Generates, for each member of the solution pool, a mapping based on the corresponding mapping string. Specifices the task properties such as execution time, power dissipation, etc.
03: **Invoke EE-GLSA:** Invoke the schedule and communication mapping optimisation to determine a suitable and energy efficient schedule for the current task mapping, for each individual of the population.
04: **Assign Fitness:** Compute fitness of each individual in the population pool.
a) Calculate area penalty
b) Derive fitness based on area penalty and the schedule fitness.
05: **Termination:** If no improved individual (improvement $> 1\%$) has been produces for 10 generations, then terminate. Otherwise, continue.
06: **Ranking:** Individuals are ranked according to their fitness.
07: **Selection:** According to the size of the generational overlap select individuals for mating. High ranked individuals have a high probability to be selected.
08: **Mating:** Produce two-point crossover between a pair of selected individuals.
09: **Mutation:** Randomly change genes of individuals using a dynamic mutation probability scheme, with exponential decreasing probability during run-time.
10: **Offspring insertion:** Exchange low ranked individuals by newly produced individuals with respect to the size of the generational overlap.
11: Invoke step 02.

Figure 5: The proposed EE-GMA approach for energy-efficient task mappings

2.6.2 Communication Mapping

Communication issues have great impact on the timing behaviour of the application and, therefore, should be considered carefully during the design space exploration [26, 14]. One important decision we have taken in this regard, was to separate the mapping of communication activities from the task mapping. The following example illustrates the reasons behind this decision. The tasks and communications of the task graph shown in Fig. 6(a) need to be mapped onto a target architecture consisting of three PEs, connected by four CLs. A possible mapping

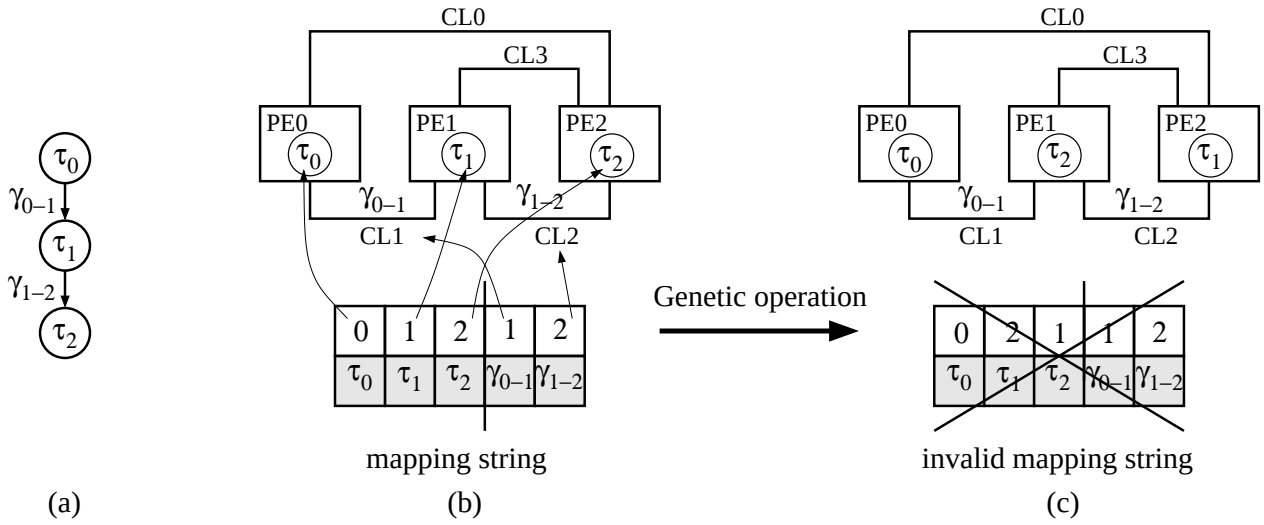


Figure 6: Combined optimisation of task and communication mapping

is shown in Fig. 6(b). Let us consider a certain genetic operation which transforms the mapping string shown in Fig. 6(b) to the one in Fig. 6(c). A quick check upon this mapping indicates that this assignment of activities represents in invalid solution. Consider for example the communication γ_{0-1} between task τ_0 and τ_1. Although the tasks are mapped onto PE0 and PE2, which are solely connected through CL0, the communication is mapped to CL1. Hence, this mapping is invalid (if we consider that only direct communications are allowed without routing over intermediate PEs).

Needless to say, a combined task and communication optimisation would lead to a high number of invalid solutions during the optimisation, which, in turn, would have a negative effect on the convergence of the population towards high quality solutions. To overcome this problem we propose a combined optimisation of scheduling and communication mapping, which is carried out for each task mapping candidate. Or, in other words, the communication mapping is carried out in Step 3 of the design flow shown in see Fig. 1. In this way it is possible to avoid invalid solutions, since all possible mappings of communication activities onto the communication links are statically known for a particular task mapping. If the tasks τ_0 and τ_1, for example, are mapped to PE1 and PE2, respectively, then the communication γ_{0-1} can only be mapped onto CL2 or CL3. Our communication optimisation, described next, takes advantage of this information to ensure that only valid solutions are produced.

The presented communication mapping optimisation is carried out in parallel with the schedule optimisation. To explain this strategy consider the extended string representation, shown in Fig. 7, which encodes both a possible schedule and a communication mapping candidate. It can be observed that this string is divided into priority and communication mapping genes. A list scheduler determines an execution order based on the encoded priorities, while the mapping of communication activities onto the communication links is given by the communication

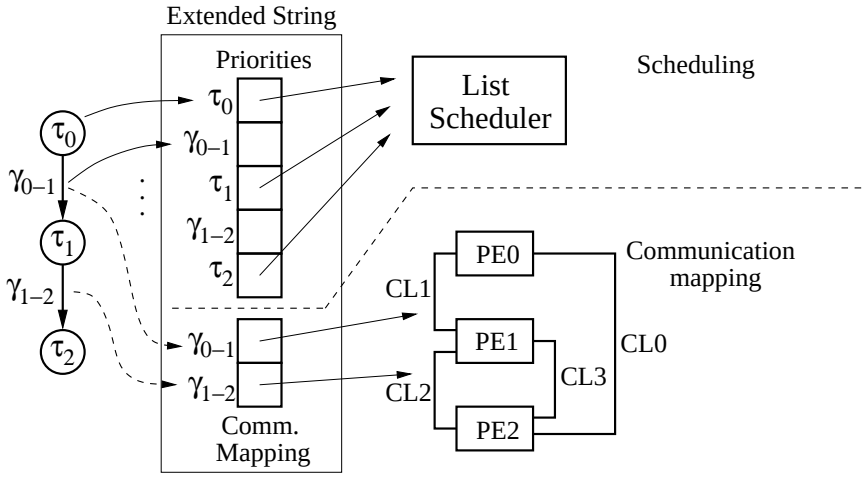


Figure 7: A combined priority and communication mapping string

mapping genes. Here we concentrate on the communication optimisation, while details about the schedule optimisation can be found in [37]. This string representation allows the concurrent optimisation of priorities and communication mappings, using a single genetic algorithm. However, it necessitates specialised genetic operators, like crossover and mutation, which operate on the two string parts in parallel but without interference. This is done to avoid a mixtures of genes that would result in invalid off-springs.

For each of the mapping genes only valid values, which result in feasible solutions, are allowed. This is not hard to achieve due to the fact that the task mapping precedes the communication mapping. Thereby, for every communicating pair of tasks the possible CLs are unambiguously specified. Therefore, it is possible to generate random initial chromosomes (random in the sense that a random choice is taken among the possible CLs) that assure proper communication mappings. Similarly, the mutation operator chooses randomly among the valid possibilities. This validity is further maintained by the standard (single or two point mating) crossover operations, due to the fact that genes maintain their spatial position in the chromosome of Fig. 7.

In order to keep the optimisation time low, the communication mapping string is dynamically adapted to the particular task mapping, as the number of inter-PE communications changes. Of course, the valid values of each gene change also dynamically in accordance to the task mapping. Note, that the presented communication mapping optimisation improves both the timing behaviour as well as the power consumption, due to the fitness calculation based on equations (4) and (5). The mapping experiments, given in Section 3, indicate the importance of this optimisation step to achieve high quality solutions in terms of feasibility and energy savings.

2.7 Component Allocation with DVS capability

The last step (outermost loop, shown in Fig. 1 as Step 1) in the system design process is the allocation of components, like processing elements and communication links, and their interconnection. In LOPOCOS, this step is user driven and thereby based on the knowledge and experience of the designer. We assume that the designer has predefined an architecture and our LOPOCOS tool helps him to evaluate its quality in terms of energy dissipation, cost, and feasibility. If an architecture does not prove to be satisfactory, the designer makes the necessary changes and evaluates again. In this way it is also possible to trade-off the different design goals and hence achieve multiple design alternatives. Similarly to the scheduling and mapping steps, the allocation of components has an influence on the usability of DVS. For example, it might be beneficial to reduce the workload on the system PEs by introducing a new PE. Such a decision can lead to increased deadline slacks in the system schedules. These slacks are exploited by DVS, resulting in higher dynamic energy reductions, while, at the same time, increase the product cost and the static power consumption. Therefore, a specifically "over-designed" system might be the preferable choice of the designer. Clearly, this optimisation is based on the astuteness of the designer.

3 Experimental Results

LOPOCOS was tested on several benchmark examples to demonstrate its capability to produce high quality solutions in terms of energy, timing, and area requirements. Experimental results for the DVS algorithm and the genetic list scheduling can be found in [35, 37]. In particular, we will focus here on the optimisation of mapping and allocation, as outlined in Sections 2.6 and 2.7. The design flow in Fig. 1 has been implemented on a Pentium-III/750MHz Linux PC with 128MB RAM. The benchmarks consist of four sets: 1) We have used TGFF [12] to generate 25 hypothetical examples (`tgff1` – `tgff25`)[1]. These specifications include *power managed* DVS-PEs and non-DVS-PEs. Accordingly, the power dissipation varies among the executed tasks (with maximal variations of 2.6 times). 2) The `Hou` examples are taken from [21]. The PEs of these benchmarks are characterised by non uniform power profiles. Since the initial PEs (taken from [21]) are not DVS enabled, we extended the same PEs with DVS capabilities, such that $V_t = 0.8V$ and $V_{max} = 3.3V$. 3) Furthermore, we have taken 5 examples from [5]. These benchmarks represent two different implementations of Fast Fourier Transforms (`fft1` and `fft3`), a Karplus-Strong music synthesis algorithm (`Karp10`), a quadrature mirror filter bank (`qmf4`), and a measurement application (`meas`). The architectures are composed out of 2 to 6 identical DVS-PEs, assuming constant power consumption. The supply voltage of these processors can be dynamically varied between 0.8 and 7 volts. The

[1] Available at: http://www.ecs.soton.ac.uk/~ms99r/benchmarks.html

throughput constraints and initial average power are calculated at a reference voltage of 5 volts. 4) The final benchmarks represents a real-life example, a traffic monitoring system based on an optical flow detection (OFD) algorithm. This application is a sub-system of an autonomous model helicopter [1, 18], specified by 32 tasks.

The remainder of this section is split into experiments concerning the hypothetical benchmarks (Section 3.1) and experiments carried out on the OFD real-life example (Section 3.2).

3.1 Hypothetical Examples

To give insight into the energy efficiency achieved by LOPOCOS, we have conducted several experiments. The first experiment shows an comparison between two different mapping approaches. The first one is based on a constructive list scheduling technique and a power profile neglecting DVS approach [29]. In the following we will refer to this approach as EVEN-DVS. The second mapping approach corresponds to the technique used in LOPOCOS. It is based on a genetic list scheduling algorithm (EE-GLSA, see Section 2.5) and a DVS technique which considers the power profile information during the voltage scaling (see Section 2.4). Table 1 shows this comparison for the benchmark sets `tgff` and `Hou`. All presented results were obtained by running the optimisation process ten times and averaging the outcomes. It can be observed that the proposed mapping technique was able to reduce the energy dissipation when compared to the results of the EVEN-DVS approach, with improvements of up to 38.8% (`tgff3`: 24.6% compared to 63.4%). The optimisation times for the EVEN-DVS based task mapping varied between 1.91*s* and 172.38*s* for task graphs with up to 100 nodes. Our approach optimised the same examples in 3.42*s* to 14050*s*. These increased execution times are due to two reasons: a) The search space for EVEN-DVS is smaller, since it is based on a constructive list scheduling, and b) the generalised DVS approach [35] shows a higher computational complexity than the voltage scaling used in EVEN-DVS. This results in the classical trade-off between optimisation time and accuracy (solution quality).

The next experiment is concerned with the benchmark examples taken from [5]. We had to re-calculate the throughput constraints at nominal supply voltage $V_{dd} = 5V$ for the same scheduling and mapping as given in [5], since we employ a different communication model (contention, requests for the bus, etc.). Unfortunately this makes a direct comparison to the results given in [5] impossible. Nevertheless, the re-calculation of the throughput was carried out for the same task mappings and execution orders as used in [5], which are based on a dynamic level scheduling (DLS) approach [39]. Due to the highly serialised structure of the `meas` example, we could further calculate the theoretically optimal supply voltage settings, which resulted in an energy reduction of 13%, with respect to a task execution at nominal supply voltage. Our synthesis approach found a near optimal solution, with an energy dissipation only 4% higher than the theoretical bound, in 8.3*s* (Tab. 2). For the remaining

Example	NO-DVS		EVEN-DVS approach [29]			LOPOCOS			
	Energy Dissip.	*CPU time (s)*	*Energy Dissip.*	*CPU time (s)*	*Reduction (%)*	*Energy Dissip.*	*CPU time (s)*	*Reduction (%)*	*Reduction Factor*
tgff1*	333	3.11	116	1.91	65.23	92	12.14	72.41	1.11
tgff2	709747	24.10	625970	13.34	11.80	445532	47.86	37.23	3.15
tgff3	298991	69.46	225433	39.68	24.60	109351	2437.98	63.43	2.58
tgff4	63924	24.15	15743	12.15	75.37	10817	290.10	83.08	1.10
tgff4_t	49807	22.93	20275	11.83	59.29	18487	226.93	62.88	1.06
tgff4_fixed	59294	20.32	18860	11.66	68.19	10621	299.45	82.09	1.20
tgff5	568210	64.42	426614	41.23	24.92	233063	904.99	58.98	2.37
tgff6	24685	19.97	7298	11.66	70.44	3799	221.99	84.61	1.2
tgff7	1491203	10.29	1169258	5.55	21.59	1058346	41.75	29.03	1.34
tgff8	525250	15.52	182894	8.49	65.18	136057	46.92	74.1	1.35
tgff9*	600428	9.01	358087	4.63	40.36	323158	45.28	46.18	1.14
tgff10	9417	7.45	8531	3.98	9.41	7193	17.58	23.62	2.51
tgff11	2858919	26.87	2400940	14.48	16.02	2229397	97.6	22.02	1.37
tgff12	174440	56.36	90087	37.61	48.36	58404	1328.52	66.52	1.38
tgff13	927704	60.97	511019	32.56	44.92	328377	853.42	64.6	1.44
tgff14	7723	23.29	7578	14.39	1.88	6693	69.01	13.34	7.11
tgff15	20017	86.85	17948	54.39	10.34	16938	916.98	15.38	1.49
tgff16	2984716	34.66	2177495	24.64	27.05	2141352	197.41	28.26	1.04
tgff17	16237	41.97	11417	26.80	29.69	8220	308.54	49.38	1.66
tgff18	1518517	4.17	1248236	2.80	17.80	1066350	9.71	29.78	1.67
tgff19	3431	5.91	2176	4.12	36.59	1907	18.31	44.41	1.21
tgff20*	18621	12.41	7286	7.18	60.87	4646	92.24	75.05	1.23
tgff21	2182722	121.95	1543090	59.71	29.30	1352422	1665.04	38.04	1.3
tgff22	894765	301.48	691269	172.38	22.74	456021	2240.57	49.03	2.16
tgff23*	5519226	147.33	3261600	87.85	40.90	2129198	14050.26	61.42	1.5
tgff24	720861	151.80	302288	98.07	58.07	200328	2199.39	72.21	1.24
tgff25	3232360	74.20	2555077	44.36	20.95	2328983	1664.63	27.95	1.33
Hou*	11816	10.57	10704	11.43	9.41	6708	163.78	43.23	4.59
Hou_clustered*	12766	1.58	10145	1.97	20.53	7879	3.42	38.28	1.86

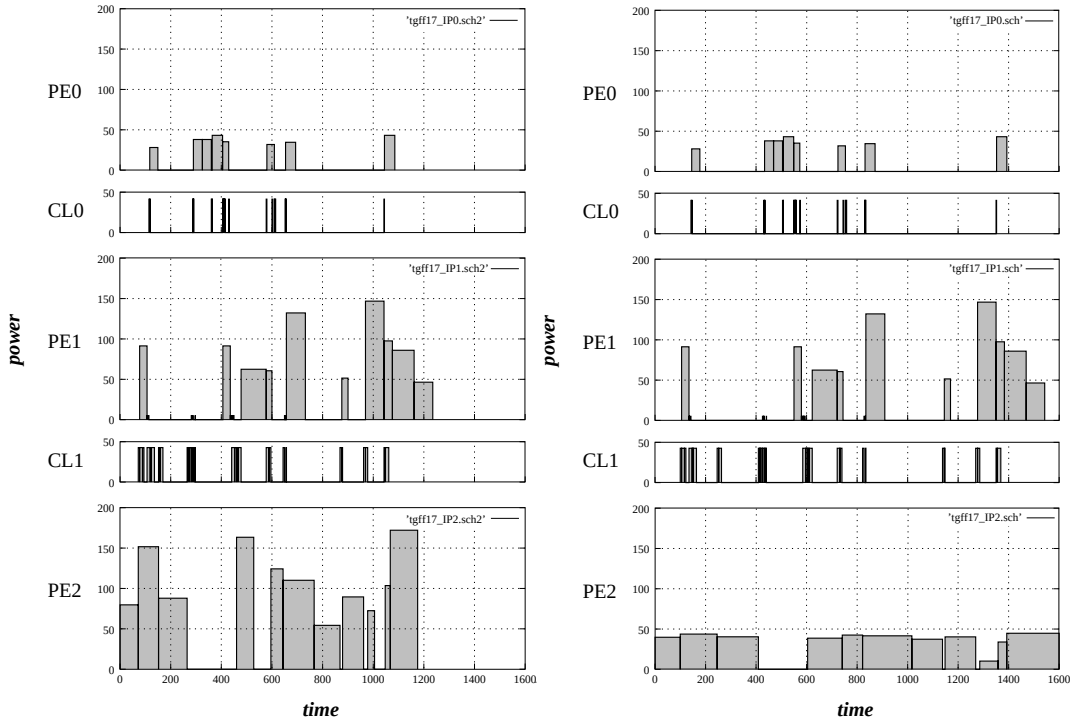
Table 1: Comparison between the mapping optimisation for EVEN-DVS [29] and the approach used in LOPOCOS (*the architectures of these benchmarks consist of DVS-PEs only)

Example	*No.of Tasks/ Comms.*	NO-DVS	LOPOCOS			DLS Mapping
		Energy Dissip.	*Energy Dissip.*	*CPU time (s)*	*Reduction (%)*	*Reduction (%)*
fft1	28/32	29600	14019	591.2	52.63	38.66
fft3	28/32	48000	21452	1144.7	55.31	23.39
karp10	21/20	59400	24055	755.1	59.50	19.63
meas	12/12	28300	25732	8.3	9.07	9.07
qmf4	14/21	16000	11097	202.3	30.64	20.38

Table 2: Comparison between dynamic level scheduling [39] and the proposed approach

benchmarks given in Tab. 2, up to 39.9% (karp10: 19.6% compared to 59.5%) higher energy savings could be achieved when compared to a constructive scheduling and mapping based on DLS.

In the next experiment, we illustrate an architecture refinement process, carried out under the supervision of the designer. Based on the feedback provided by the optimisation steps 1 to 3 (Fig. 1), the designer allocates



(a) Task execution at nominal supply voltage (**no** DVS is exploited)

(b) Scaled task execution using the proposed DVS approach, which **considers** the power profiles

Figure 8: Tow identical execution orders of the `tgff17` benchmark: (a) unscaled (NO-DVS) and using (b) the DVS techniques using in LOPOCOS

necessary and/or deallocates unnecessary PEs and CLs, in order to achieve the intended trade-offs between system cost, energy dissipation, and quality. To demonstrate this, we have carried out this optimisation step on the task graph example `tgff17`. The allocated architecture consists of three PEs (PE0–PE2) including one DVS-PE (PE2). These components are connected via two buses (CL0 and CL1). A possible mapping and scheduling for this example is given in Fig. 8, showing both the schedule at nominal voltage and at a dynamically changing voltage. Using this allocation of components, the total energy dissipation $E_{total} = E_{stat} + E_{dyn} = 48870$ is achieved when utilising the presented PV-DVS mapping and schedule optimisation. The cost of this system is 1656. However, the budget for the system design might be 1800 and so the designer can change the design in order to find a different trade-off between energy dissipation and cost. For example, it seems to be a good idea to exchange PE1 with a DVS enabled PE, since the system power profile of the current allocation (as given in Fig. 8(b)) shows a high power consumptions for PE1. Certainly, allocating a DVS enabled version of PE1 will increase the static power dissipation and the cost of the system, due to the hardware overhead of the dynamic supply voltage hardware, but it might also enable a further reduction of the dynamic power consumption. To clarify this, we have carried out the following experiment. For the DVS enabled version of PE1 it is considered that its static power dissipation is

Architecture	*Static Power (W)*	*Dyn. Power (W)*	*Total Power (W)*	*CPU Time (s)*	*Reduction (%)*
2 DSPs	0.383	2.137	2.520	n/a	n/a
3 DVS-DSPs	0.574	1.371	1.945	148.3	22.8%
4 DVS-DSPs	0.736	1.163	1.899	303.6	24.6%
5 DVS-DSPs	0.898	1.132	2.030	381.9	19.4%

Table 3: Increasing architectural parallelism to allow voltage scaling of the OFD algorithm

10% higher than for its no-DVS version (a realistic assumption based on the system described in [32]) and that its cost is increased by 100. The changed system configuration results in a total energy dissipation of 46798 and a implementation cost of 1756. Whether this energy reduction justifies the increased system cost strongly depends on the application domain. For example, if the system is going to be a unique implementation (e.g. satellite sub-system) higher cost might be acceptable, while in the case of mass products, cost constraints could be more stringent. In a similar manner, the additional allocation of components might relax the schedule and introduce more available slack time to be used by the DVS technique. Again, it is necessary to compromise between the achieved dynamic energy reduction, the increased static power consumption and the cost.

3.2 Real-life Example: OFD Algorithm

The final experiments are concerned with an energy efficient implementation of an optical flow detection (OFD) algorithm on board of an autonomous helicopter. In its current implementation the OFD algorithm runs on two ADSP-21061L DSPs, with an average current of $760mA$ at $3.3V$, hence, an average power dissipation of approximately $2.5W$. Due to the stringent power budget on board of the helicopter, including application critical sub-systems, it is necessary to keep the overall power dissipation under a certain limit. To reduce the power consumption to a minimal amount, DVS seems predestined, since the OFD algorithm shows an unnecessary high performance (12.5 frames of 78x120 pixels per second). However, a repetition rate of 6.25 frames per second is sufficient to ensure a correct detection, allowing to relax the system specification. For experimental purpose we consider a hypothetical extension of the DSPs towards DVS capability. We take into account that such an extension has an influence on the static power dissipated by the digital circuits and, therefore, increase the static current by 10% [32].

In the first part of this experiment we keep the application constraints fixed, i.e., the OFD algorithm needs to perform with a repetition rate of $12.5Hz$ (equivalent to the current implementation). In order to increase the usage of the application parallelism, we use three different architectures build out of three to five DVS-DSPs, connected via a shared bus. In this way the OFD algorithm can be performed faster. Table 3 reports on our findings. The

Architecture	*Static Power (W)*	*Dyn. Power (W)*	*Total Power (W)*	*CPU Time (s)*	*Reduction (%)*
2 DSPs	0.383	1.069	1.452	n/a	n/a
2 DVS-DSPs	0.413	0.394	0.807	783.5	44.4%
3 DVS-DSPs	0.574	0.277	0.851	1107.2	41.4%
4 DVS-DSPs	0.736	0.253	0.989	1393.4	31.9%
5 DVS-DSPs	0.898	0.241	1.139	1634.7	21.6%

Table 4: Relaxing the performance constraints of the OFD algorithm

first row represents the current implementation of the OFD algorithm, i.e., running on an architecture without DVS technology. This implementation shows a total power consumption of $2.52W$. Now, consider the DVS enabled architectures with three to five DSPs. In accordance with the number of allocated PEs, the static power consumption has increased as well. However, the PEs are capable to exploit the application parallelism more effectively, which, in turn, allows a fast execution of the OFD algorithm. This results in a slack time, usable by the DVS-PEs to lower the dynamic power dissipation. As it can be observed from Table 3, all implementations using DVS-DSPs show a reduced total power consumption (sum of static and dynamic power consumption) of up to 24.6%. Please note that this reduction does not necessitate any performance degradation, while the cost of the system increases.

As we have mentioned before, the current implementation of the OFD algorithm shows an unnecessary high performance and it is therefore possible to relax the specified system constraints. Hence, in the following experiment, we reduced to the repetition rate from $12.5Hz$ to $6.25Hz$, i.e., an execution at half speed. This performance is still high enough to allow a correct flow detection. The results of this investigation are shown in Table 4. We can observe that for all given architectures the energy consumption can be significantly reduced by up to 44.4%, when compared to a non DVS implementation (first row in Tab. 4). However, among all implementation alternatives the architecture composed out of 2 DVS-PEs seems to be the favourite, since it achieves the highest energy savings at a low cost. This is due to the fact that with each additionally allocated PE the static power consumption increases, while the achievable dynamic energy reductions decrease (caused by limited parallelism of the application). Again, this shows how important an accurate design space exploration is, when synthesising DVS enabled embedded systems.

4 Conclusions and Future Work

In this paper, we have comprehensively introduced LOPOCOS, an experimental co-design tool for distributed embedded architecture including DVS processors. Extensive experiments, carried out on several hypothetical and

real-life examples, show very encouraging results in terms of energy efficiency and timing behaviour. It was shown that with the usage of a GA based synthesis approach for DVS enable architectures, it is possible to find better solutions when compared to constructive scheduling and mapping techniques, in reasonable amounts of time. The energy efficiency is achieved not only through the schedule and mapping optimisation towards DVS, but under the additional consideration of the PE power profiles during these optimisation steps. Furthermore, it was shown that a combined scheduling and communication mapping optimisation can help to overcome the specific problem of a combined task and communication mapping optimisation.

Current work extends the presented co-synthesis system towards conditional process graphs, in order to increase its specification flexibility. This further necessitates the consideration of on-line scheduling and voltage scaling techniques to increase the possible energy savings by taking into consideration dynamic execution times.

References

[1] WITAS: The Wallenberg laboratory for research on Information Technology and Autonomous System. http://www.ida.liu.se/ext/witas/.

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

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

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

[5] N. Bambha, S. Bhattacharyya, J. Teich, and E. Zitzler. Hybrid Global/Local Search Strategies for Dynamic Voltage Scaling in Embedded Multiprocessors. In *Proc. 1st Int. Symp. Hardware/Software Co-Design (CODES'01)*, pages 243–248, April 2001.

[6] Luca Benini, Giovanni De Micheli, Enrico Macii, Massimo Poncino, and R. Scarsi. Symbolic Synthesis of Clock-gating Logic for Power Optimization of Synchronous Controllers. *Trans. on Design Automation of Electronic Systems*, 4(4):351–375, 1999.

[7] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto. Energy Estimation for 32 bit Microprocessors. In *Proc. 8th Int. Workshop Hardware/Software Co-Design (CODES'00)*, pages 24–28, May 2000.

[8] Thomas D. Burd, Trevor A. Pering, Anthony J. Stratakos, and Robert W. Brodersen. A Dynamic Voltage Scaled Microprocessor System. *IEEE J. Solid-State Circuits*, 35(11):1571–1580, November 2000.

[9] Bharat P. Dave, Ganesh Lakshminarayana, and Niraj K. Jha. COSYN: Hardware-Software Co-Synthesis of Embedded Systems. In *Proc. DAC*, pages 703–708, 1997.

[10] Muhammad K. Dhodhi, Imtiaz Ahmad, and Robert Storer. SHEMUS: Synthesis of Heterogeneous Multiprocessor Systems. *J. Microprocessors and Microsystems*, 19(6):311–319, August 1995.

[11] R. Dick and N. K. Jha. MOCSYN: Multiobjective core-based single-chip system synthesis. In *Proc. Design, Automation and Test in Europe Conf. (DATE99)*, pages 263–270, March 1999.

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

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

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

[15] R. Ernst, J. Henkel, and Th. Brenner. Hardware-Software Co-synthesis for Mirco-Controllers. *IEEE Design & Test of Comp.*, 10(4):64–75, Dec 1993.

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

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

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

[19] R.K. Gupta. *Co-Synthesis of Hardware and Software for Digital Embedded Systems*. PhD thesis, Stanford University, December 1993.

[20] Inki Hong, Darko Kirovski, Gang Qu, Miodrag Potkonjak, and Mani B. Srivastava. Power Optimization of Variable-Voltage Core-Based Systems. *IEEE Trans. Computer-Aided Design*, 18(12):1702–1714, Dec 1999.

[21] Junwei Hou and Wayne Wolf. Process Partitioning for Distributed Embedded Systems. In *Proc. CODES*, pages 70 – 76, March 1996.

[22] Tohru Ishihara and Hiroto Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proc. Int. Symp. Low Power Electronics and Design (ISLPED'98)*, pages 197–202, 1998.

[23] Asawaree Kalavade and Edward A. Lee. A Global Criticality/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem. In *Proc. CODES*, pages 42–48, Sept. 1994.

[24] Darko Kirovski and Miodrag Potkonjak. System-level Synthesis of Low-Power Hard Real-Time Systems. In *Proc. IEEE 34th Design Automation Conf. (DAC97)*, pages 697–702, 1997.

[25] Alexander Klaiber. *The Technology Behind Crusoe Processors*. Transmeta, Jan 2000.

[26] P. V. Knudsen and J. Madsen. Integrating Communication Protocol Selection with Hardware/Software Codesign. *IEEE Trans. Computer-Aided Design*, 18(9):1077–1095, Aug 1999.

[27] Seongsoo Lee and Takayasu Sakurai. Run-time Voltage Hopping for Low-power Real-time Systems. In *Proc. IEEE 37th Design Automation Conf. (DAC00)*, pages 806–809, 2000.

[28] Y.-T. S. Li, S. Malik, and A. Wolfe. Performance Estimation of Embedded Software with Instruction Cache Modeling. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD-95)*, pages 380–387, November 1995.

[29] Jiong Luo and Niraj K. Jha. Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD-00)*, pages 357–364, Nov 2000.

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

[31] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1996.

[32] Trevor Pering, Thomas D. Burd, and Robert B. Brodersen. The Simulation and Evaluation for Dynamic Voltage Scaling Algorithms. In *Proc. Int. Symp. Low Power Electronics and Design (ISLPED'98)*, pages 76–81, August 1998.

[33] Michael Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, 1995.

[34] S. Prakash and A. Parker. SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems. *J. Parallel & Distributed Computing*, pages 338–351, Dec 1992.

[35] Marcus T. Schmitz and Bashir M. Al-Hashimi. Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems. In *Proc. Int. Symp. System Synthesis (ISSS'01)*, pages 250–255, October 2001.

[36] Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles. Co-Synthesis with Energy Minimisation for Heterogeneous Distributed Systems containing Power Managed Processing Elements. Technical Report UOS-TR-MTS01, University of Southampton, UK, Department of Electronics and Computer Science, September 2001.

[37] Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles. Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems. In *Proc. Design, Automation and Test in Europe Conf. (DATE2002)*, March 2002.

[38] Youngsoo Shin and Kiyoung Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proc. IEEE 36th Design Automation Conf. (DAC99)*, pages 134–139, 1999.

[39] Gilbert C. Sih and Edward A. Lee. A Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel and Distributed Systems*, 4(2):175–187, February 1993.

[40] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power Analysis of Embedded Software: A First Step Towards Software Power Minimization. *IEEE Trans. VLSI Systems*, Dec 1994.

[41] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 13–23, 1994.

[42] Wayne H. Wolf. An Architectural Co-Synthesis Algorithm for Distributed, Embedded Computing Systems. *IEEE Trans. VLSI Systems*, 5(2):218–229, June 1997.