Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems

Marcus T. Schmitz and Bashir M. Al-Hashimi
Dept. of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, United Kingdom
{m.schmitz,bmah}@ecs.soton.ac.uk

Petru Eles
Dept. of Computer and Information Science
Linköping University
S-58183 Linköping, Sweden
petel@ida.liu.se

Abstract

In this paper, we present an efficient two-step iterative synthesis approach for distributed embedded systems containing dynamic voltage scalable processing elements (DVS-PEs), based on genetic algorithms. The approach partitions, schedules, and voltage scales multi-rate specifications given as task graphs with multiple deadlines. A distinguishing feature of the proposed synthesis is the utilisation of a generalised DVS method. In contrast to previous techniques, which "simply" exploit available slack time, this generalised technique additionally considers the PE power profile during a refined voltage selection to further increase the energy savings. Extensive experiments are conducted to demonstrate the efficiency of the proposed approach. We report up to 43.2% higher energy reductions compared to previous DVS scheduling approaches based on constructive techniques and total energy savings of up to 82.9% for mapping and scheduling optimised DVS systems.

1. Introduction and Related Work

Modern embedded systems are often implemented as distributed systems consisting of several processing elements (PEs), like programmable microprocessors, ASIPs, FPGAs, and ASICs. In reality, such embedded systems have to concurrently perform a multitude of complex tasks under a strict timing behaviour, given in the system specification. However, due to the various degrees of application parallelism, the PEs experience non-uniform workloads, resulting in idle intervals. Furthermore, the performance of the allocated architecture can often not be adapted perfectly to the application needs, turning out as slack between deadline and the real finishing time.

Dynamic voltage scaling (DVS) exploits such idle and slack times to reduce the power consumption [12, 14, 21]. This is done by conjointly changing the supply voltage and the operational frequency during run-time, with respect to temporal performance requirements. Recent implementations of DVS processors have shown that voltage scaling can reduce the power consumption by up to 10 times when running real-life applications [5]. Nevertheless, modern microprocessors make often use of gated clocks to switch off unused circuit parts during idle times. Hence, the power consumption depends on the function carried out, resulting in non-uniform PE power profiles; this holds also for DVS-PEs [5]. It was shown in [14, 19, 23] that the consideration of the PE power profile during the voltage selection leads to further energy savings.

System level co-design is a methodology aiming to aid the system designers/architects to solve the difficult problem of finding the "best" suitable implementation for a system specification. Three important co-synthesis steps are: a) *Mapping:* Determining the assignment of computational tasks to PEs and data transfers to communication links (CLs), b) *Scheduling:* Determining the execution order (sequencing) of tasks mapped to PEs and communications to CLs, and c) *Evaluation:* Determining the quality of the implementation candidate (timing feasibility, cost, power, area, etc.). Previous research in co-synthesis is extensive but has mainly focused on traditional architectures *excluding* issues related to power [15, 20, 26] or considered energy optimisation with components that are *not* DVS enabled [8, 16, 22]. This research will provide a valuable basis for the presented work. However, three recently proposed synthesis approaches for distributed systems have a close relationship to the problems we address in this paper. In [11], a DVS optimised schedule is derived using a constructive list scheduling technique with a dynamic re-calculation of task priorities based on average energy dissipation. If the found schedule does not meet the specified deadline, priorities of tasks on the critical path are increased and all tasks are re-scheduled. In [18], a mobility based list schedule is modified towards DVS utilisation by distributing slack time more evenly among the tasks of the system. A method for the identification of scaled supply voltages for distributed system was introduced in [3]. However, it focuses mainly on the voltage selection and its iterative nature results in undesirable high execution times, which can not be tolerated in the inner most loop of an iterative schedule and mapping optimisation. All these approaches are based on constructive scheduling heuristics and neglect the power profile information during the supply voltage selection.

This paper presents an iterative list scheduling heuristic to simultaneously optimise the schedule towards feasible timing behaviour and utilisation of DVS-PEs, and hence the minimisation of the dissipated energy. Due to the potentially larger

search space of iterative optimisation methods, compared to constructive techniques, schedules with reduced energy dissipation are likely be found. Furthermore, the optimisation process is guided by a generalised DVS algorithm [23]. This voltage scaling technique takes into account the PE power profiles during a refined voltage selection, leading to further energy reductions. In addition to the schedule optimisation, we employ a task mapping based on GA to push the distribution of tasks among the architecture towards energy-efficiency by abetting the exploitation of DVS. Overall, we concentrate here on the task mapping and scheduling aspects rather than on the voltage selction, which is explained in [23].

The presented work makes the following contributions: a) It is shown how iterative improvement mapping and scheduling algorithms can be effectively adapted to optimise system implementations towards an efficient utilisation of the DVS-PEs while meeting, at the same time, hard deadlines. This is done using a new two-step approach for scheduling and mapping based on genetic algorithms (GAs). b) The outlined schedule optimisation is based on a DVS algorithm, which takes into account the PE power profiles, hence, leading to further energy savings. c) To illustrate the efficiency of the proposed approach, a comparative study is presented, comparing our results with two recently published synthesis approaches [11, 18], which are based on constructive list scheduling heuristics and neglect the PE power profiles. This further includes a quantitative comparison between a variable-voltage system and a multi-voltage system, which demonstrates the efficiency of the proposed technique also for multi-voltage processors.

The remainder of this paper is organised as follows: Preliminary aspects are introduced in Section 2. Section 3 describes our synthesis approach in detail which then, in Section 4, is extended to multi-voltage systems. In Section 5 we present extensive experiments and comparisons with the results produced by other approaches. We conclude in Section 6.

2. Preliminaries

2.1. Specification and Architectural Model

In this work, we consider that a multi-rate application is specified as a set of communicating tasks, represented by a task graph $G_S(\mathcal{T}, \mathcal{C})$. This (hyper) task graph might be the combination of several smaller task graphs, capturing all task activations for the hyper-period (LCM of all graph periods). Fig. 1(a) shows a task graph example. 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 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 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.

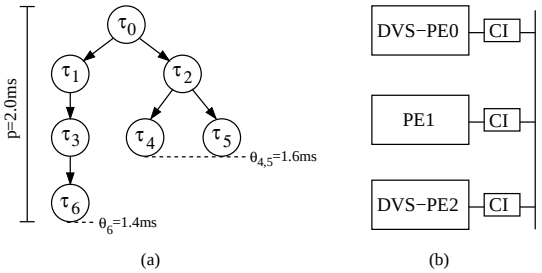


Figure 1. Task graph and DVS architecture

The architectures we consider here consist of heterogeneous 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. An infrastructure of communication links, like buses and point-to-point connections, connects these PEs 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. 1(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 [4, 17, 25]. This is not a trivial task and influenced by various parameters, e.g. the input data of the application. However, such techniques are essential to enable an effective co-synthesis, including the presented approach.

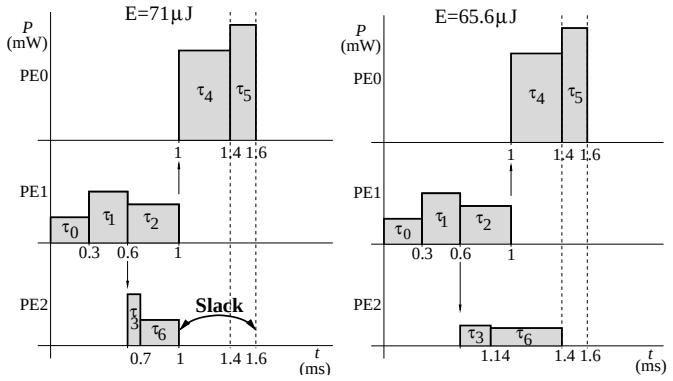
2.2. Task Execution Order and DVS

The relation between dynamic power dissipation P_{dyn}, operational frequency f, and supply voltage V_{dd} is expressed by,

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

$$f = k \cdot (V_{dd} - V_t)^2 / V_{dd} \quad (2)$$

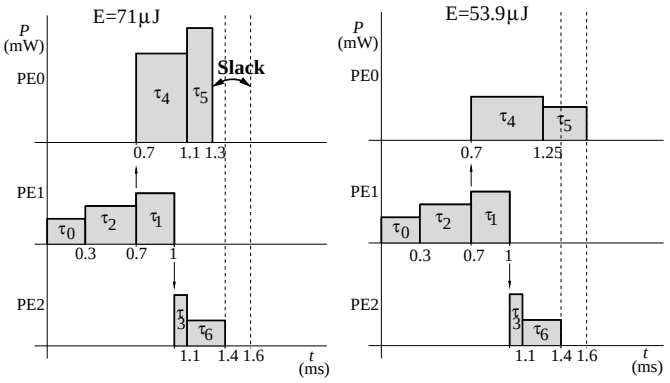
where C_L denotes the load capacitance of the digital circuit, $N_{0\to1}$ represents the zero-to-one switching activity, k is a circuit dependent constant, and V_t is the threshold voltage. It can be observed from Equation (2) that the operational frequency f decreases with decreasing supply voltage V_{dd}, and hence the task execution time increases. Thereby, DVS is applicable in schedules where idle and slack times can be found, allowing to slow down certain tasks while meeting hard deadlines. Since the execution order of tasks influences the idle and slack times in a schedule it should be optimised for the utilisation by DVS. To clarify this, consider the following illustrative example.



(a) Execution at nominal supply voltage V_{max}

(b) Scaled execution with $V_{dd3} = 2.08V$ and $V_{dd6} = 2.34V$

Figure 2. Possible schedule not optimised for DVS



(a) Execution at nominal supply voltage V_{max}

(b) Scaled execution with $V_{dd4} = 2.74V$ and $V_{dd5} = 2.41V$

Figure 3. Schedule optimised for generalised DVS

Fig. 2(a) shows a possible schedule for the tasks given in Fig. 1(a) executing at nominal supply voltage. The underlying architecture consists of two DVS-PEs (PE0, PE2) and one non-DVS-PE (PE1) connected through a bus, as given in Fig. 1(b). The nominal supply voltage V_{max} and the threshold voltage V_t of PE0 and PE2 are $V_{max} = 3.3V$ and $V_t = 0.8V$, respectively, while PE1 runs all tasks at V_{max}. For the sack of simplicity, the communications are neglected when discussing this particular example. The task execution times t_{min} and power dissipations P_{max} at nominal supply voltage are given in Table 1, which also shows the task mapping. According to these values, the energy

Task	t_{min} *(ms)*	P_{max} *(mW)*	*mapping*
τ_0	0.3	10	PE1
τ_1	0.3	20	PE1
τ_2	0.4	15	PE1
τ_3	0.1	40	PE2
τ_4	0.4	70	PE0
τ_5	0.2	90	PE0
τ_6	0.3	20	PE2

Table 1. Execution times, power dissipations, and mappings for the example task graph

dissipation corresponding to the given schedule can be calculated as $E = \sum_{\tau \in \mathcal{T}} P_{max}(\tau) \cdot t_{min}(\tau) = 71\mu J$. Considering the deadlines given in Fig. 1(a), it can be observed from Fig. 2(a) that the tasks τ_3 and τ_6 are eligible for scaling, since τ_6 finishes at $1ms$ and it has a deadline $\theta_6 = 1.4ms$, resulting in a slack of $0.4ms$. An extension of any other task can not be tolerated, since task τ_5 has a finishing time equal to its deadline. By scaling the schedule, using our implementation of the generalised DVS technique (taking the PE power profile into account) presented in [23], the voltage schedule shown in Fig. 2(b) can be produced, with tasks τ_3 and τ_6 executing at $2.08V$ and $2.34V$, respectively. Thereby, the energy is reduced to $65.6\mu J$ (using Equations (1) and (2)), a 7.6% reduction.

Now, consider a second feasible schedule at nominal supply voltage, as shown in Fig. 3(a), where the order of τ_1 and τ_2 has been exchanged. Since the mapping of the tasks has not been modified the dissipated energy remains $E = 71\mu J$. Observing this schedule shows that only tasks τ_4 and τ_5 can be extended. This is due to the slack time of task τ_5, which finishes execution at $1.3ms$ while its deadline is $\theta_5 = 1.6ms$. Generating a voltage schedule for this execution order of tasks results in the supply voltages $V_{dd4} = 2.74V$ and $V_{dd5} = 2.41V$, using the same generalised DVS technique as for the previous alternative. Hence, the energy is reduced to $E = 53.9\mu J$, an improvement of 24.1% compared to the 7.6% of the first schedule.

Although the schedule in Fig. 3(a) shows less slack than the one in Fig. 2(a), its energy reduction is significantly higher (with 16.5%). This is due to the particular power consumptions when executing the different tasks. The example demonstrates how important it is to take into consideration the power profiles during scheduling, in order to produce energy-efficient implementations with DVS-PEs.

2.3. Genetic List Scheduling Algorithm

List scheduling algorithms (LS) make scheduling decisions based on task priorities. They maintain one or more ready list, which contain tasks ready to be scheduled. A static schedule is constructed by scheduling the ready task with the highest priority as soon as the eligible PE becomes available. Thereby, the assignment of priorities defines the task execution order. Most traditional list scheduling approaches use various sophisticated algorithms to calculate these task priorities statically (before list scheduling) or dynamically (re-calculation after each scheduling step).

In contrast, genetic list scheduling algorithms (GLSA) construct and evaluate many different schedules during an iterative optimisation process. Task priorities are encoded into priority strings, hence, a manipulation through genetic operators (e.g. crossover, mutation, etc.) is possible. As common for genetic algorithms (GAs), the optimisation is guided by a objective, called fitness, which needs to be minimised or maximised. More details on our GLSA are given in Section 3.1. The three main advantages of GLSA over traditional LS are: a) The objective can be based on an arbitrary complex function which needs to be optimised. b) The enlarge search space provides the opportunity to find solutions with a potentially higher qual-

ity. c) There is a large freedom to trade-off between acceptable synthesis time and solution quality, as opposed to constructive techniques.

2.4. Problem Formulation

Using the common triplet notation for scheduling problems, our problem is 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 scheduling 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 [24].

We make the assumption that the specified tasks are of sufficiently coarse granularity and that the PEs can continue operation during the voltage scaling (as the case for the DVS processor in [5]), which allows to neglect of the scaling overhead in terms of power and time.

3. Energy-Efficient Synthesis Approach

In contrast to the GLSA based synthesis approach presented in [6, 10], our synthesis approach separates task mapping and scheduling into two nested optimisation steps; see Fig. 4(a).

- The GLSA for energy-efficiency, which produces an optimised sequencing of task executions (Section 3.1).
- A mapping optimisation based on GA, which distributes the tasks among the PEs of the architecture and, by this, decides on the execution time and power dissipation of each task (Section 3.2).

We have split these two steps due to the following reasons: a) The combination of list scheduling and mapping algorithms decide upon task priorities which task is to be scheduled next, but at this point it is *not* known where to execute the chosen task. Therefore, the execution time and power dissipation of the task are unknown as well. In this context, it is the duty of the scheduler to make a "greedy" mapping decision based on the power and time values with respect to the design objectives. However, DVS influences the execution times and power dissipations, hence, the mapping decision made upon the static values might proof to be wrong, especially from the energy reduction point of view. Separating the scheduling and mapping into two nested iterative optimisations overcomes this problem since the mapping is given before a schedule is constructed. b) Due to the constructive nature of list scheduling and mapping algorithms a solution is constructed one by one. This results in a greedy approach, which is likely to get trapped at low quality or infeasible solutions in the presence of tight area and timing constraints. A solution to overcome this problem was presented in [15]. However, this approach neglects issues related to power. By splitting the problem into two steps, we avoid this greediness problem and can leverage the advantage of an increased search space, which is explored iteratively. Clearly, increasing the search space results in high optimisation times, however, we show in Section 5 that these times are still reasonable.

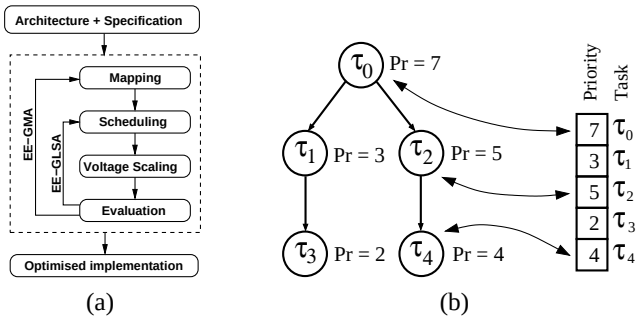


Figure 4. Presented energy-efficient synthesis approach and task priority encoding into priority string

3.1. Low-Energy Genetic List Scheduling Algorithm

In this section, we give an overview of our DVS optimised GLSA for energy-efficiency, called EE-GLSA. The algorithm generates an energy-efficient schedule of tasks and communications, for a *given* mapping. By imitating and applying the principles of natural selection and "survival of the fittest" on a population pool of individuals, GAs are able to evolve (optimise) solutions over several generations. In each generation a new population is evolved by mating (through crossover) the fittest individuals of the current population. Mutation provides an additional opportunity to enter unexplored regions of the search space by applying randomly changes to an individual. In our case, each individual is represented by a priority string (solution candidate) and each solution represents a schedule. Fig. 4(b) shows the encoding and the relations between priority string and tasks. A description of the EE-GLSA is given in Fig. 5. The distinguishing features of this GA can be found in steps 02, 03, and 04, which are explained next. The remaining steps vary only slightly from common GAs, and more details on the functionality of GAs can be found elsewhere [9]. In step 02, for each priority string of the population a schedule is generated by going through the following two steps: a) The priorities of each individual are assigned from the corresponding priority string. b) Based on this priority assignment, the execution order of tasks is determined by a list scheduling algorithm. In addition, our implementation of the list scheduler relies solely on the priorities to make scheduling decisions, i.e., no other optimisation technique (e.g. hole filling) is applied. Although such techniques can improve the timing behaviour by eliminating idle periods, we dissociate from them since DVS exploits exactly these idle times.

In step 03, the produced schedules are passed to our generalised DVS algorithm [23], which identifies the supply voltage for each task executed on a DVS-PE to minimise the energy dissipation. These voltage schedules are generated taking into account the PE power profiles, leading to a further energy reduction. Based on the steps 02 and 03, the fitness for each priority string is calculated in step 04. It is this fitness function which guides the optimisation process, and therefore, it should lead the search towards low energy and feasible implementa-

EE-GLSA

Input: - task graph TG
- mapping and execution properties corresponding to the mapping

Output: - timing and energy optimised schedule

01: **Initialisation:** Create initial population pool P of priority strings, half randomly generated and half based on mobility.
02: **Perform List Scheduling:** Generate, for each member of the solution pool, a schedule based on the corresponding priority string.
a) Assign task priorities from the priority string
b) Invoke list scheduler without hole filling
03: **Perform Voltage Scaling:** Invoke the generalised DVS technique, calculating supply voltages for each task executed on a DVS-PE. This is done under the consideration of the individual power dissipation of tasks.
04: **Assign Fitness:** Compute fitness of each individual in the population pool.
a) Calculate timing penalty
b) Calculate energy based on the supply voltages
c) Derive fitness based on energy and timing penalty
05: **Ranking:** Individuals are ranked according to their fitness.
06: **Selection:** According to the size of the generational overlap, select individuals for mating. High ranked individuals have a high probability to be selected.
07: **Mating:** Produce two-point crossover between a pair of selected individuals.
08: **Mutation:** Randomly change genes of individuals using a dynamic mutation probability scheme, with exponential decreasing probability during run-time.
09: **Offspring insertion:** Exchange low ranked individuals by newly produced individuals with respect to the size of the generational overlap.
10: **Termination:** If no improved individual (improvement $> 1\%$) has been produces for 10 generations, then terminate. Otherwise, continue with step 02.

Figure 5. The proposed EE-GLSA approach for energy-efficient schedules

tions. Our EE-GLSA relies on the following fitness function F_S to achieve these goals,

$$F_S = \underbrace{\left(\sum_{\varepsilon \in \mathcal{A}} E(\varepsilon)\right)}_{\text{Energy diss.}} \cdot \underbrace{\left(1 + \frac{\sum\limits_{\tau \in \mathcal{T}_d} DV_\tau^2}{T_{HP}^2}\right)}_{\text{Time penalty}} \tag{3}$$

where $\mathcal{A} = \mathcal{T} \cup \mathcal{C}$ defines the set of all activities and $\mathcal{T}_d$ represents all hard deadline tasks. The first part of the equation is used to calculate the total dynamic energy dissipation of all activities $\varepsilon \in \mathcal{A}$. Based upon the type of activity, the energy dissipation can be calculate in the following way,

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

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 and execution time of communication activities. The second part of the fitness function (3) introduces a penalty factor due to deadline violations of deadline tasks which are given by $DV_\tau = \max\left(0, t_F(\tau) - t_d(\tau)\right)$, where $t_F(\tau)$ and $t_d(\tau)$ denote the finishing time and deadline of task τ, respectively. T_{HP} is the hyper task graph period, used to relate the deadline violations. Squaring has been applied in order to assign a higher penalty to larger violations of imposed deadlines. The parameters of the GA where set as follows: The population pool consists of 25 individuals, the dynamic mutation probability is calculated as $M_P = max(0.15, 1/exp(N_S \cdot 0.05))$ (N_S is the current generation), and the generational overlap is 50%.

3.2. Low-Energy Task Mapping Algorithm

The mapping step determines which PE carries out which task. Thereby, it determines the execution time and power dissipation at nominal supply voltage. The mapping also specifies the area requirements of tasks in terms of bytes or gates, whether implemented as software or hardware. Obviously, due to the interrelation between scheduling and mapping, the distribution of tasks among the PEs has an influence on how well the allocated DVS-PEs can exploit their energy reduction possibilities.

We have extended a GA based task mapping algorithm similar to the one given in [8] such that it solves our specific problem. The extension is based on the presented EE-GLSA algorithm (see Section 3.1), which is called from inside the mapping optimisation loop and is used to calculate parts of the mapping fitness function F_M. In our GA based mapping approach, called EE-GMA, solution candidates are encoded into mapping strings. Each gene of these strings captures a mapping of a task to a PE. The GA we use to evolve the solutions is similar to the previously presented one (Fig. 5). In order to guide the optimisation not only towards low energy and timing feasible solutions, using the scheduling fitness, but also towards feasibility in term of area, the fitness function F_M uses an additional objective, namely area. The fitness we assign to individuals is expressed by,

$$F_M = F_S \cdot \prod_{\pi \in \mathcal{P}} AP_\pi \tag{4}$$

where F_S denotes the schedule fitness (including the DVS reduced energy and the timing penalty, as given by Equation (3)) and AP_π represents an area penalty for each PE $\pi \in \mathcal{P}$ with exceeded area constraints. The exact equation for the calculation of AP_π is given by,

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

where the used area is denoted as SA_π and the maximal available area is represented by AA_π. If the available area AA_π is not exceeded we do not need to assign an area penalty, hence, F_S is multiplied by one. Otherwise, the used area SA_π and the available area AA_π are related and multiplied by a constant k, which allows to adjust the aggressiveness of the penalty. During our experiments we have set $k = 0.02$. This was found to

be low enough to keep a high population diversity while avoiding, at the same time, infeasible results. The parameters of the GA where set as follows: The population pool consists of 50 individuals, the dynamic mutation probability is calculated as $M_P = max(0.05, 1/exp(N_M \cdot 0.05))$ (N_M represents the current generation), and the generational overlap is 20%.

4. Variable-Voltage vs. Multi-Voltage

This section clarifies the differences between variable-voltage and multi-voltage DVS-PEs and introduces necessary equations, later used in the experimental results. The generalised DVS technique [23], as used in our synthesis approach, produces supply voltages under the assumption that a continuous voltage range is available. However, real DVS processors [1, 2, 5] show a limited number of supply voltages at which tasks can be executed. For example, the DVS processor given in [5] uses a 7 bit frequency register, allowing to operate at 15 different discrete voltage-frequency (5 bit VCO) settings. Therefore, the continuous selected supply voltages are not directly applicable, however, they can be used as a base for mutli-voltage selection. It has been shown in [14] that the two neighbouring discrete voltages V_{d1} and V_{d2}, $V_{d1} < V_{dd} < V_{d2}$, around the continuous selected voltage V_{dd} are the ones which minimise energy, under the assumption that the time overhead for switching between voltages can be neglected. The corresponding execution times t_{d1} and t_{d2} for task execution at V_{d1} and V_{d2}, can be calculated as,

$$t_{d1} = t_{exe} \cdot \frac{V_{d1} \cdot (V_{dd} - V_t)^2}{(V_{d1} - V_t)^2 \cdot V_{dd}} \cdot \frac{\frac{V_{dd}}{(V_{dd}-V_t)^2} - \frac{V_{d2}}{(V_{d2}-V_t)^2}}{\frac{V_{d1}}{(V_{d1}-V_t)^2} - \frac{V_{d2}}{(V_{d2}-V_t)^2}} \quad (5)$$

$$t_{d2} = t_{exe} - t_{d1} \quad (6)$$

where t_{exe} denotes the execution time of the task at the continuous selected voltage V_{dd}.

5. Experimental Results

The proposed synthesis approach was tested on several benchmark examples to demonstrate its capability to produce high quality solutions in terms of energy, timing, and area requirements. It was implemented using C++ on a Pentium-III/750Mhz/128MB Linux PC. The benchmarks consist of four sets: 1) We have used TGFF [7] 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 [13]. The PEs of these benchmarks are characterised by non uniform power profiles. Since the initial PEs, considered in [13], are not DVS enabled, we extended the same PEs with DVS capabilities, such that $V_t = 0.8V$ and $V_{max} = 3.3V$. 3) The benchmark sets `TG1` and `TG2` where taken from [11] and consist of 30 graphs, each. These specifications include DVS-PEs with constant power dissipation (uniform power profile) and the given time constraints represent tight deadlines. 4) The final benchmark `meas` was taken from [3] and represents a measurement application with 12 tasks and 12 communications. The architecture consists of two identical, uniform power profile DVS-PEs.

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

Example	No. of tasks/ edges	**EVEN-DVS** [18] Reduction mobility (%)	**EVEN-DVS** [18] Reduction GLSA (%)	**Proposed** Reduction EE-GLSA (%)
Tgff1	8/9	45.50	46.27	71.05
Tgff2	26/43	2.80	22.91	26.79
Tgff3	40/77	25.98	51.89	69.18
Tgff4	20/33	6.66	12.55	12.99
Tgff5	40/77	5.34	11.13	17.14
Tgff6	20/26	1.23	1.35	1.61
Tgff7	20/27	10.16	24.47	29.90
Tgff8	18/26	7.28	10.01	13.83
Tgff9	16/15	2.25	16.76	24.85
Tgff10	16/21	26.08	34.65	35.77
Tgff11	30/29	1.28	13.67	16.96
Tgff12	36/50	3.14	4.49	5.11
Tgff13	37/36	16.73	19.56	20.71
Tgff14	24/33	12.78	23.44	28.12
Tgff15	40/63	0.84	2.13	4.15
Tgff16	31/56	16.63	28.68	29.88
Tgff17	29/56	13.06	19.34	22.20
Tgff18	12/15	0.00	6.87	23.44
Tgff19	14/19	20.63	23.98	27.84
Tgff20	19/25	37.77	45.02	52.30
Tgff21	70/99	0.07	6.13	19.45
Tgff22	100/135	13.48	19.87	29.10
Tgff23	84/151	6.70	15.05	23.20
Tgff24	80/112	0.06	2.08	8.53
Tgff25	49/92	1.50	14.18	20.16
Hou	20/29	7.29	22.46	39.40
Hou_c	8/7	20.64	20.64	28.56

Table 2. Scheduling comparison between EVEN-DVS [18] and the proposed EE-GLSA approach

To give insight into the energy efficiency of the proposed synthesis approach, we compare it first with the approach presented in [18], which neglects the PE power profiles (in the following we call this approach EVEN-DVS). Table 2 shows this comparison for the benchmark sets `tgff` and `Hou`. Each benchmark is characterised by its complexity in terms of nodes and edges. Column 3 shows the achieved energy reductions (with respect to a task execution at nominal supply voltage) of the EVEN-DVS approach using a mobility based schedule. This represents the approach presented in [18]. Column 4 shows the energy reduction for the same DVS technique, however, the mobility based scheduling was replaced by a GLSA. More exactly, the scheduling is performed using our GA based approach but without the generalised DVS technique which is part of the EE-GLSA. In Column 5 the energy reductions of the proposed EE-GLSA approach, based on a generalised DVS technique and a GLSA, are presented. Comparing Column 3 and 5, it can be observed that our approach is able to reduce the energy dissipation of all examples further. The achieved reductions are up to 43.2% percent higher. However, to avoid the misleading argument that these higher energy reductions are solely introduced by the GLSA, we have combined the EVEN-DVS approach with exactly the same scheduling technique (GLSA) as used in our approach, Column 4 shows the results. It can be observed that the proposed EE-GLSA technique results in higher energy savings for all examples, with reductions of up to 24.78% com-

Example	LEneS [11]		Proposed		
	Average Reduc. dis. (%)	CPU time (s)	Average Reduc. cont. (%)	Average Reduc. dis. (%)	CPU time (s)
TG1	28	10–120	41.16	37.61	3–16
TG2	13	10–120	18.82	15.83	0.3–1.7

Table 3. Comparison between the results of the LEneS algorithm [11] and the proposed EE-GLSA

pared to the EVEN-DVS based GLSA. This indicates that the generalised DVS technique [23] combined with the presented scheduling approach generates solutions, that lead to higher energy savings. Regarding the computational complexity, the results in the Columns 3, 4, and 5 where achieved in most 0.23*s*, 1.19*s*, and 17.99*s*, respectively, for benchmarks with up to 100 tasks.

Next, we compare the approach proposed by Gruian et al. [11], called LEneS, with the presented scheduling technique. Similar to EVEN-DVS, LEneS neglects the PE power profile during the voltage selection. Table 3 presents the results obtained by both algorithms for the benchmarks TG1 and TG2. LEneS was able to reduce the power consumption of both benchmark sets on average by 28% and 13%. The optimisation took between 10*s* and 120*s* for each of the 60 task graphs in the benchmark sets. The presented EE-GLSA was able to reduce these values further. The average energy reductions resulted in 41.16% and 18.82%. However, since our approach produces continuous selected scaling voltages, we have adopted the same discrete voltages (0.9V, 1.7V, 2.5V, and 3.3V) as given in [11] to ensure a fair and accurate comparison. Using Equations (5) and (6) the energy reductions of the multi-voltage system are calculated as 37.61% and 15.81%. Note that, since the benchmark sets TG1 and TG2 show constant power dissipation among the executed tasks, our approach is not able to leverage its additional energy reduction feature to consider these power variations. However, the achieved savings are 9.61% and 2.81% higher, showing that our approach performs well even when applied to systems with uniform power profiles. Comparing the computational times indicates a performance advantage of the proposed method, which produced results in 0.3*s* to 16*s*. Another interesting observation is that the multi-voltage setting (using just 4 discrete voltages) consumes only less than 4% more power than the variable-voltage approach.

We have further conducted a set of mapping optimisation experiments and achieved similar results and observations as for the GLSA and EE-GLSA based schedule optimisation. The results are given in Table 4, which compares EVEN-DVS and the proposed EE-GLSA when included into the same mapping algorithm (EE-GMA, Section 3.2). The proposed technique was able to further reduce the energy dissipation when compared to the results of the EVEN-DVS approach, with improvements of up to 42.26%. The optimisation times for EVEN-DVS varied between 1.91*s* and 172.38*s* for task graphs with up to 100 nodes. Our approach optimised the same examples in 2.27*s* to 87657*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

Example	EVEN-DVS [18]		Proposed	
	Reduction (%)	CPU time (s)	Reduction (%)	CPU time (s)
Tgff1	65.23	1.91	70.60	6.53
Tgff2	11.80	13.34	47.08	46.78
Tgff3	24.60	39.68	66.86	2394.47
Tgff4	75.37	12.15	82.88	585.47
Tgff5	24.92	41.23	54.00	1824.16
Tgff6	70.44	11.66	82.14	374.11
Tgff7	21.59	5.55	28.75	51.08
Tgff8	65.18	8.49	72.44	49.91
Tgff9	40.36	4.63	46.28	63.97
Tgff10	9.41	3.98	23.58	14.81
Tgff11	16.02	14.48	25.79	133.10
Tgff12	48.36	37.61	80.45	3295.83
Tgff13	44.92	32.56	61.22	1958.38
Tgff14	1.88	14.39	17.09	96.06
Tgff15	10.34	54.39	22.85	1066.99
Tgff16	27.05	24.64	28.97	275.79
Tgff17	29.69	26.80	45.32	396.99
Tgff18	17.80	2.80	30.02	12.27
Tgff19	36.59	4.12	47.14	23.96
Tgff20	60.87	7.18	76.42	144.99
Tgff21	29.30	59.71	33.41	3441.92
Tgff22	22.74	172.38	47.48	3438.95
Tgff23	40.90	87.85	61.97	87657.76
Tgff24	58.07	98.07	72.08	16355.14
Tgff25	20.95	44.36	26.44	2740.64
Hou	9.41	11.43	41.48	31.51
Hou_c	20.53	1.97	37.76	2.27

Table 4. Comparison between the mapping optimisation for EVEN-DVS [18] and EE-GLSA

DVS approach [23] shows a higher computational complexity than the voltage scaling used in EVEN-DVS.

The next experiment is concerned with the benchmark example meas. We had to re-calculate the throughput constraints at nominal supply voltage $V_{dd} = 5V$ for the same scheduling and mapping as given in [3], since we employ a different communication model (contention, requests for the bus, etc.). Unfortunately this makes a direct comparison to the results given in [3] impossible. Nevertheless, due to the highly serialised structure of this example, we could calculate the theoretically optimal supply voltages 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*.

The final experiment demonstrates that our scheduling optimisation (EE-GLSA) does not only reduce significantly the dissipated energy, but simultaneously improves the timing behaviour compared to constructive techniques. This is of great importance since high quality solutions could be found in design space regions where infeasible and feasible solutions are spatially placed closely together. Making a wrong decision might involve a more costly implementation of the system. To clarify this, consider the mapping and scheduling results shown in Table 5. The ten examples are taken from Gruian's benchmarks set TG1 and use an architecture of 10 identical DVS-PEs. Column 2 shows the reduction results obtained by EVEN-DVS, which is based on a constructive list scheduling heuristic

Example	**EVEN-DVS** [18]		**Proposed**	
	Reduc. (%)	CPU time (s)	Reduc. (%)	CPU time (s)
r000	unsolved	18.60	26.53	194.86
r001	21.97	13.87	47.35	804.73
r002	unsolved	16.51	25.89	189.97
r003	28.43	15.98	44.29	769.58
r004	unsolved	19.97	36.15	360.58
r005	37.45	16.75	49.83	1596.67
r006	unsolved	17.55	34.62	827.22
r007	unsolved	20.20	32.48	269.07
r008	unsolved	19.25	26.32	207.46
r009	37.64	14.99	54.23	1535.28

Table 5. Comparison between the mapping optimisation for EVEN-DVS **[18] and** EE-GLSA **for** TG1

(mobility based). Observe that for 6 out of 10 task graphs the scheduling and mapping attempt failed (unsolved), making it necessary to increase the performance of the allocated system. On the other hand, our EE-GLSA is able to improve infeasible schedules by providing feedback to the optimisation process. In this way, it was possible to find feasible mappings and schedules for all examples by using our EE-GLSA approach. This effect is likely to appear in the presence of tight deadline specifications, as e.g., the benchmark set TG1 of Gruian et al. [11]. Of course, these higher quality results require longer optimisation times.

6. Conclusions

We have presented a new approach for the energy-efficient scheduling and mapping of distributed embedded systems. 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 genetic list scheduling and mapping algorithm can be extended to solve the specific problems introduced through voltage scaling. We have also validated the quality of the proposed approach through extensive benchmark examples and a comparison with two recently proposed synthesis techniques for DVS enable distributed systems. This has shown that with the usage of a GA based synthesis approach for DVS enabled architectures it is possible to find better solutions in reasonable amounts of time.

Acknowledgements

The authors wish to acknowledge Flavius Gruian (Lund University, Sweden) and Neal K. Bambha (University of Maryland, USA) for kindly providing their benchmark sets.

References

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

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

[3] N. Bambha, S. Bhattacharyya, J. Teich, and E. Zitzler. Hybrid Global/Local Search Strategies for Dynamic Voltage Scaling in Embedded Multiprocessors. In *Proc. CODES*, pages 243–248, April 2001.

[4] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto. Energy Estimation for 32 bit Microprocessors. In *Proc. CODES*, pages 24–28, May 2000.

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

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

[7] R. Dick, D. Rhodes, and W. Wolf. TGFF: Task Graphs for free. In *Proc. CODES*, pages 97–101, March 1998.

[8] R. P. Dick and N. 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.

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

[10] M. Grajcar. Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System. In *Proc. DAC*, pages 280–285, 1999.

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

[12] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava. Power Optimization of Variable-Voltage Core-Based Systems. *IEEE Trans. Computer-Aided Design*, 18(12):1702–1714, 1999.

[13] J. Hou and W. Wolf. Process Partitioning for Distributed Embedded Systems. In *Proc. CODES*, pages 70 – 76, March 1996.

[14] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proc. ISLPED*, pages 197–202, 1998.

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

[16] D. Kirovski and M. Potkonjak. System-level Synthesis of Low-Power Hard Real-Time Systems. In *Proc. DAC*, pages 697–702, 1997.

[17] Y.-T. S. Li, S. Malik, and A. Wolfe. Performance Estimation of Embedded Software with Instruction Cache Modeling. In *Proc. ICCAD*, pages 380–387, Nov. 1995.

[18] J. Luo and N. K. Jha. Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems. In *Proc. ICCAD*, pages 357–364, Nov 2000.

[19] A. Manzak and C. Chakrabarti. Variable Voltage Task Scheduling for Minimizing Energy or Minimizing Power. In *Proc. ICASSP*, pages 3239–3242, 2000.

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

[21] G. Quan and X. S. Hu. Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors. In *Proc. DAC*, pages 828–833, 2001.

[22] A. Rae and S. Parameswaran. Voltage Reduction of Application-Specific Heterogeneous Multiprocessor Systems for Power Minimisation. In *Proc. ASP-DAC*, pages 147–152, 2000.

[23] M. T. Schmitz and B. M. Al-Hashimi. Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems. In *Proc. ISSS*, pages 250–255, Oct 2001.

[24] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. Co-Synthesis with Energy Minimisation for Heterogeneous Distributed Systems containing Power Managed Processing Elements. Tech. Report UOS-TR-MTS01, University of Southampton, Sept. 2001.

[25] V. Tiwari, S. Malik, and A. Wolfe. Power Analysis of Embedded Software: A First Step Towards Software Power Minimization. *IEEE Trans. VLSI Systems*, Dec 1994.

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