Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems

D. Wu, B.M. Al-Hashimi and P. Eles

Abstract: A dynamic voltage scaling (DVS) technique for embedded systems expressed as conditional task graphs (CTGs) is described. The idea is to identify and exploit the available worst case slack time, taking into account the conditional behaviour of CTGs. Also the effect of combining a genetic algorithm based mapping with the DVS technique is examined and it is shown that further energy reduction can be achieved. The techniques are tested on a number of CTGs including a real-life example. The results show that the DVS technique can be applied to CTGs with an energy saving of up to 24%. Furthermore, it is shown that savings of up to 51% are achieved by considering DVS during the mapping optimisation. Finally, the impact of communications and communication link selection on the scheduling and mapping technique is investigated and results are reported.

1 Introduction

Energy efficiency is becoming a central issue in embedded system synthesis with increasing demand for portable devices, and with heat dissipation caused by excessive power consumption which can lead to reduced reliability. One potentially effective technique for decreasing power consumption in embedded systems is dynamic voltage scaling (DVS), which dynamically scales the supply voltage and operational frequency of system components during run-time in accordance with the temporal performance requirements of the application [1]. DVS exploits the slack time, i.e. the intervals when a PE is idle, such as to reduce power consumption.

Several approaches have demonstrated the efficiency of task scheduling with DVS techniques in reducing the power consumption of embedded applications [2–6]. The efficiency of such techniques can be further increased if the potential of voltage scaling is considered not only during the scheduling step, but also for optimisation of the task mapping [7]. In [8], a mobility based list scheduling was modified towards DVS utilisation: the authors optimise a static schedule towards the incorporation of aperiodic tasks. The static schedule provides guidelines to the online scheduler. In [9], a DVS optimised schedule was derived using a constructive list scheduling technique with a dynamic recalculation of task priorities based on average energy dissipation. In [7], a two-step iterative synthesis approach guided by a generalised DVS algorithm was presented, which optimises both the mapping and schedule towards energy efficiency by abetting the exploitation of DVS.

All of the approaches mentioned above have considered either systems consisting of independent tasks or purely data dominated applications specified as dataflow models. However, embedded system functionality often contains both data and control statements. This feature has been recognised by the research community and several system level representations have been proposed to capture both the data and control flow at task level [10, 11]. In [10, 12] such an abstract system representation, called the conditional task graph (CTG), has been defined and a scheduling algorithm has been proposed whereby the worst case delay is minimised. In [13], a technique performing mapping and scheduling simultaneously to take advantage of the resource sharing among mutual exclusive tasks was proposed.

Using system representations which capture both data and control flow allows for a more accurate modelling of a large class of embedded systems. This will lead to more exact performance estimations, schedule generations and, in general, more efficient system implementations. Based on such considerations, researchers [10, 12–14] have addressed scheduling and mapping of embedded systems expressed with CTGs or similar representations. However, such an accurate system representation also offers the potential of efficient implementations in terms of energy dissipation. Nevertheless, no work has yet addressed the problem of energy minimisation during synthesis of system specifications which capture both dataflow and the flow of control.

The main aim of this paper is to investigate the application of DVS techniques to data/control dominated embedded systems. The following are the two main contributions of this work:

1. A DVS technique for CTGs is proposed which is capable of exploiting the slack time, taking into account the conditional behaviour of the system.
2. A genetic algorithm (GA) based mapping technique is introduced to optimise the system implementation such as to efficiently exploit the proposed DVS technique, thus, leading to further energy savings.

IEE Proceedings online no. 20030837

doi: 10.1049/ip-cdt:20030837

Paper received 9th May 2003

D. Wu and B.M. Al-Hashimi are with the School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK

P. Eles is with Department of Computer and Information Science, Linköpings University, S-58183 Linköping, Sweden

2 Preliminaries

2.1 Conditional task graph and architectural model

Consider an application specified as a directed, acyclic graph $G(V, E_S, E_C)$ called a *conditional task graph* (CTG) [12]. Figure 1*a* shows an example CTG. Each node, $n_i \in V$ represents a task, an atomic unit to be executed without being pre-empted. There are two nodes, called *source* and *sink*, which represent the first and last node, respectively, such that all other nodes in the graph are successors of the source and predecessors of the sink. E_S and E_C are the sets of simple and conditional edges, respectively. $E_S \cap E_C = \Phi$ and $E_S \cup E_C = E$, where E is the set of all edges. An edge $e_{ij} \in E$ from n_i to n_j indicates that the output of n_i is the input of n_j. An edge $e_{ij} \in E_C$ is a *conditional edge* (represented with thick lines in Fig. 1) and it has an associated condition value. Transmission on such an edge takes place only if the associated condition value is met. A node with conditional edges at its output is called a *disjunction node*. Executing a disjunction node produces a *condition value*. For example in Fig. 1*a*, executing n_1 produces condition value A or $\bar{A}$. Alternative paths starting from a disjunction node meet in a *conjunction node*. A conjunction node can be activated after input from one of the alternative paths has arrived. Depending on the condition values, there exist different *tracks* through a CTG which may be followed at a certain execution. The CTG of Fig. 1*a* has three possible tracks, which are shown in Figs. 1*b*, *c* and *d* respectively.

If we consider the activation time of the source task as a reference, the finish time of the sink task is the delay of the system at a certain execution. This delay must be, in the worst case, smaller than a certain imposed deadline. Release times of some tasks as well as multiple deadlines can be easily modelled by inserting dummy nodes between certain tasks and the source or the sink node respectively. These dummy nodes represent tasks with certain execution time but which are not allocated to any processing element (PE). The above execution semantics is that of a so-called *single rate system*. It assumes that a node is executed at most once for each activation of the system. If tasks with different periods have to be handled, this can be achieved by generating several instances of the tasks and building a CTG which corresponds to a set of tasks as they occur within a time period that is equal to the least common multiple of the periods of the involved tasks. For further details concerning the CTG representation reference is made to [12].

The architecture considered in this work consists of multiple and heterogeneous PEs. DVS-enabled PEs can run at voltages between the threshold voltage and maximum voltage. Continuous voltages are calculated here, but these can be easily adapted for discrete voltages [7]. An assumption is made that the tasks are of sufficiently coarse granularity and that the PEs can continue operation during the voltage scaling, which allows for neglect of the scaling overhead in terms of power and time. Furthermore, the PEs may employ power management techniques, i.e. they may shut themselves down when idle. An infrastructure of communication links (CLs) connects these PEs through communication interfaces (CIs), which are able to adapt to the different operational frequencies caused by DVS. Figure 2 shows an example architecture, where CL_1 connects PE_1, PE_2, and PE_3, CL_2 connects PE_2 and PE_3. We assume that there is at least one CL directly connecting two communicating PEs. Multi-hop communication is not considered.

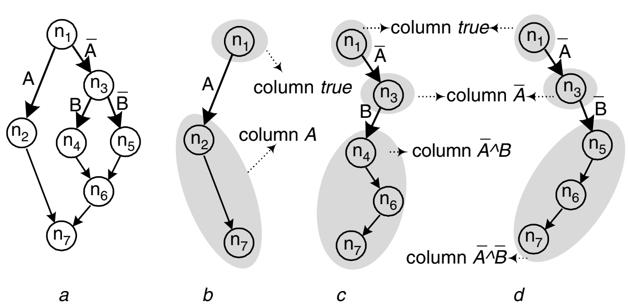

Fig. 1 *Conditional task graph and its tracks*

a CTG
b Track 1
c Track 2
d Track 3

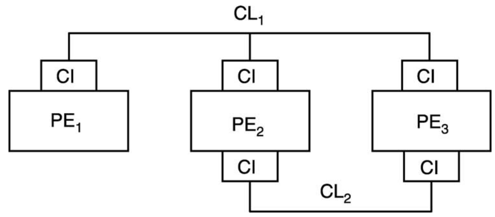

Fig. 2 *Example architecture*

Each task in a CTG may have multiple implementation alternatives, thus, it may potentially be mapped to several PEs able to execute this task. For each possible task mapping, certain implementation properties, e.g. execution time and power consumption, are given in a technology library. These values are either based on previous design experience or on estimation techniques.

The schedule table produced in [12] captures all the details related not only to task activation but also to communication scheduling. In [7, 15] we have also shown how communication aspects have to be considered for scheduling and mapping with DVS. In Section 3 it is assumed that communications between tasks takes zero time and consumes zero energy. However, the impact of communications is investigated in Section 4.

2.2 Schedule table

For a given execution of a CTG, a subset of the tasks is activated corresponding to the actual track, which depends on the values of certain conditions. In [12] a scheduling algorithm is proposed for mapped conditional task graphs whereby the worst case delay is as small as possible. The output of the algorithm is a *schedule table* which contains activation times for each task, corresponding to different values of the conditions. Table 1 is an example schedule table for the CTG of Fig. 1*a*, assuming that the task mappings and task execution times are as shown in Table 2. The table has one row for each task, which contains a start

Table 1: Schedule table

	true	*A*	$\bar{A}$	$\bar{A} \wedge B$	$\bar{A} \wedge \bar{B}$
n_1	0, 10				
n_2		10, 15			
n_3			10, 14		
n_4				14, 16	
n_5					14, 20
n_6				16, 17	20, 21
n_7		15, 20		17, 22	21, 26

Table 2: Task mapping and execution time

	mapping	execution time (ms)
n_1	PE_1	10
n_2	PE_2	5
n_3	PE_2	4
n_4	PE_1	2
n_5	PE_1	6
n_6	PE_2	1
n_7	PE_1	5

and end time for that task corresponding to different condition values. Each column in the table is headed by a logical expression constructed as a conjunction of condition values. The schedule table represents the schedules of all possible tracks corresponding to different condition values. As shown in Figs. 1*b–d*, there are three possible tracks. The schedule of track 1 is represented in column *true* and A. The schedule of track 2 is captured in column *true*, $\bar{A}$, and $\bar{A} \wedge B$. The schedule of track 3 is given in column *true*, $\bar{A}$, and $\bar{A} \wedge \bar{B}$.

The schedule table captures a quasistatic schedule of the system specified by the CTG considering the given task mapping. This means that all decisions which could be taken offline are made by the scheduling algorithm and are written into the schedule table. Based on this information, the real-time kernels running on each processing element will take the actual decisions on activation of tasks and communications, depending on the current values of conditions.

The problem formulation may be stated as follows: considering a system specified as a CTG, find a mapping, a schedule table and the voltage scaling, such that the deadline is satisfied and the energy dissipation is minimised. The execution of a CTG can proceed along different tracks depending on the actual condition values. Our objective is to minimise the total energy dissipation assuming that every track is executed with equal probability.

It should be noted that the scheduling is static and not online. However, this does not mean exhaust exploration. The scheduling algorithm traverses the CTG analysing each possible alternative track and considering for each track only the tasks executed for the respective condition values. Thus the algorithm proceeds along a binary decision tree corresponding to the alternative tracks, which is explored in depth first order.

3 Scheduling and mapping techniques with DVS for CTGs

The relation between energy dissipation $E(V_{dd})$, execution time $d(V_{dd})$ and supply voltage V_{dd} are expressed in [7]:

$$E(V_{dd}) = \frac{V_{dd}^2}{V_{\max}^2} \cdot E(V_{\max}) \tag{1}$$

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

where V_{max} is the maximum supply voltage, $E(V_{max})$ and $d(V_{max})$ are the energy dissipation and execution time at V_{max}, and V_t is the threshold voltage. Equations (1) and (2) will be used in the DVS technique in the following Sections. The application of DVS techniques for offline task scheduling is based on the assumption that a certain slack time is available and that this slack is also predictable, at least to a certain extent, at design time. In the case of system specifications which also capture the flow of control, as is the case with CTGs, constructing a quasistatic schedule with voltage scaling is even more difficult than for pure data flow systems, due to the additional problems related to the prediction of slacks. The values of the conditions are unpredictable, so the decision on how much slack time can be distributed to a task is taken without knowing which values the downstream conditions will later get, i.e. the execution path is determined incrementally during runtime. On the other hand, at a certain moment during execution, when the values of some conditions are already known (upstream conditions), they have to be used in order to take the best possible decisions.

3.1 DVS technique for CTGs

To illustrate the problems connected to the generation of a quasistatic schedule with voltage scaling for CTGs, we consider the CTG of Fig. 1*a* and its mapping information of Table 2. Let us assume that the deadline of the system is 30 ms. Figures 3*a–c* show the schedules of the three possible tracks through the CTG, as given in Table 1. The schedules are produced using the algorithm reported in [12], where the aim is to produce a schedule such that the worst

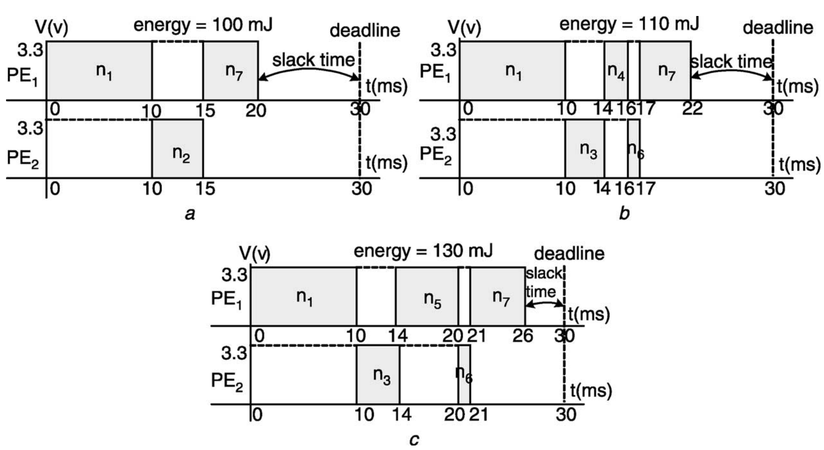

Fig. 3 *Schedule of the CTG of Figure 1a*

a Track 1
b Track 2
c Track 3

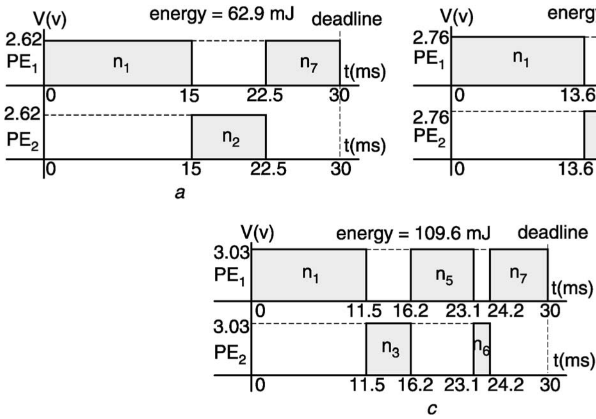

Fig. 4 *Schedule scaled for energy minimisation*

a Track 1
b Track 2
c Track 3

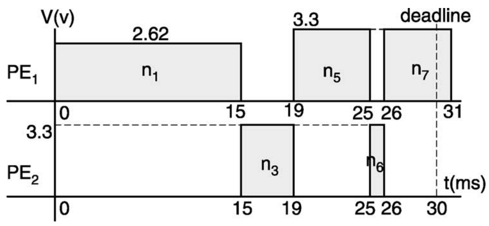

Fig. 5 *Improper scaling*

case delay is as small as possible. As can be seen from Fig. 3, the amount of slack time varies with the tracks, ranging from 4 ms in the case of track 3 to 10 ms in the case of track 1, since the deadline of 30 ms is not to be exceeded. Figures 3*a–c* also show the energy dissipation of each track, assuming that the PEs' power consumption at nominal supply voltage is 5 W. For example, in Fig. 3*a*, the tasks in track 1 consume an energy of $5\,\mathrm{W} \cdot (10\,\mathrm{ms} + 5\,\mathrm{ms} + 5\,\mathrm{ms}) = 100\,\mathrm{mJ}$. In order to make use of DVS techniques for energy minimisation, a well-known technique [2, 9] involves scaling the schedules such that they fit the imposed deadlines as much as possible. The *scaling factor* is the ratio between the deadline and the total length of the schedule. For example, the scaling factor for track 1 is calculated by $30/20 = 1.5$. The schedules obtained after scaling each track, considered in isolation from the others, is given in Fig. 4*a–c*. Figures 4*a–c* also show the energy dissipation of each track after scaling, assuming that the PEs' nominal voltage and threshold voltage are 3.3 V and 0.8 V respectively. For example in Fig. 4*a*, the execution time of track 1 is extended from 20 ms to 30 ms. Thus, using (1)–(2), it can be calculated that the supply voltage of PEs should be scaled to 2.62 V and the energy dissipation of track 1 becomes 62.9 mJ. It can be observed that, in order to produce minimal energy dissipation, the execution time of task n_1 varies from one track to the other. In the case of track 1, n_1 runs from 0 to 15. In the case of track 2 the same task runs from 0 to 13.6, and for track 3, n_1 runs from 0 to 11.5. During execution, however, the condition values of the CTG are not known in advance. If the supply voltage and, implicitly, the execution time of n_1 is decided upon improperly, the time constraints may conflict, which cannot be tolerated in systems with hard-real time properties. For example, as shown in Fig. 5, if n_1 runs from 0 to 15, and the condition values come out to be $\bar{A} \wedge \bar{B}$ later, the deadline will be missed even if the remaining tasks are run using maximum supply voltage. Thus, in order to exploit slack time as much as possible and, at the same time, meet time constraints, the worst case slack time (the maximum slack time that can be distributed to a task without later conflicting time constraints during upcoming scheduling decisions) should be identified dynamically and used to decide how much slack time a task can exploit. The main goal of our DVS

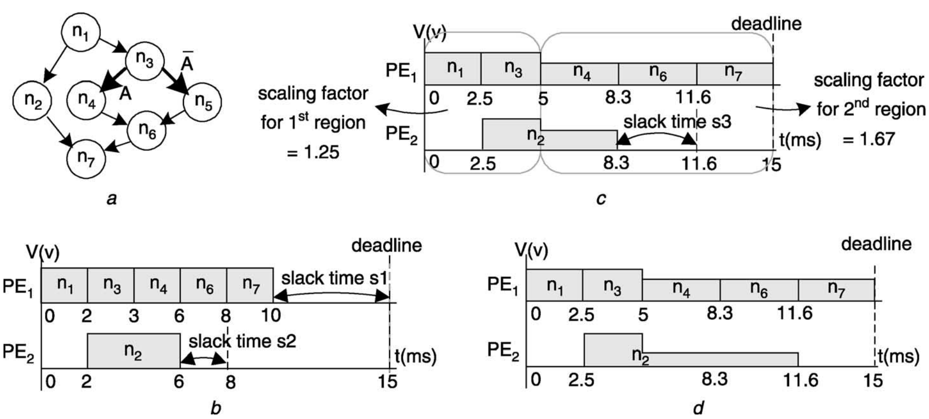

Fig. 6 *A CTG example and its scaling*

a CTG
b Schedule when the condition vaule is A
c Scaling the schedule of (b)
d Exploring slack on non-critical path

Table 3: Original schedule table

	true	*A*	*Ā*
n_1	0, 2		
n_2	2, 6		
n_3	2, 4		
n_4		4, 6	
n_5			4, 8
n_6		6, 8	8, 10
n_7		8, 10	10, 12

Table 4: Scaled schedule table

	true	*A*	*Ā*
n_1	0, 2.5		
n_2	2.5, 5	5, 8.3	5, 7.5
n_3	2.5, 5		
n_4		5, 8.3	
n_5			5, 10
n_6		8.3, 11.6	10, 12.5
n_7		11.6, 15	12.5, 15

scheduling technique for CTGs is the identification of a voltage schedule such that, under any possible set of condition values, deadlines are satisfied and, at the same time, high energy savings are achieved.

3.2 Energy-efficient scheduling

In this paper we propose a DVS technique for CTG. The basic idea is to identify the available worst case slack time taking into account the conditional behaviour of CTGs. This is achieved by dynamically identifying the worst case track, calculating the scaling factor (i.e. the ratio between the deadline and the total length of the schedule) and modifying the schedule table every time after a disjunction node (a node producing a condition value) has been scheduled. The input of our DVS technique is a schedule table generated by the scheduling methodology presented in [12] whose aim is to make the worst case delay as small as possible. A slack time exploited schedule table is produced indicating voltage levels and activation times such that deadlines are satisfied and at the same time energy dissipation is reduced.

Our strategy is based on the idea of exploiting the information concerning condition values, available at a certain time, in order to apply the largest possible scaling factor while still guaranteeing the deadline. The point in time where additional information concerning the future evolution of the system becomes available is the moment when a disjunction node ends. Therefore, at the beginning of the scheduling process, a more conservative scaling factor is applied. Once a disjunction node has been scheduled and, as a result, more available slack time can be identified, a higher scaling factor should be applied. Thus, the schedule of a CTG is divided into several *scaling regions* by the end times of the disjunction nodes. Each scaling region is then scaled with a certain, suitable scaling factor. Examining Table 1, it can be seen that the schedules of the tasks in each column correspond to such a scaling region. However, a column of the initial schedule table need not directly correspond to a scaling region. This will be the case whenever, according to the generated schedule, a task is running in parallel with a disjunction task and is finishing after that one. Such a situation is illustrated with the CTG in Fig. 6*a*. Figure 6*b* presents the schedule of the track corresponding to condition value A according to the schedule table in Table 3. Task n_2 is running over the finishing time of disjunction task n_3. However, when task n_3 has finished, the information concerning the selected tracks 'in our case, the one corresponding to condition value A) is available. Therefore, in order to make use of the available slack, a larger scaling factor will be applied and, consequently, the PE will be run at lower voltage, as shown in Fig. 6*c*. The corresponding scaled schedule table is shown in Table 4. It can be seen that task n_2 belongs to three different scaling regions, corresponding to the situations before and after the end of disjunction task n_3.

The central idea is to identify the scaling regions delimited by the end times of disjunction tasks and to scale the schedules of the tasks in each region after determining the slack time available and the corresponding scaling factor. The drawback of this scaling technique is that the tasks on the non-critical paths do not take advantage of

```
DVS technique for CTGs

Input: a schedule table generated by [12] - SchTable
       deadline - T_d
Output: a slack time exploited schedule table indicating voltage
levels and activation times - ScaledSchTable

pre-process SchTable
for (each column col in SchTable, from left to right)
{
identify the worst case track - track_worst
calculate the worst case total slack time - slack_worst
calculate the slack time distributable to col - slack_col
scale col with scaling_factor given by Equation(3)
apply DVS technique in[5]to col
update SchTable
}
```

Fig. 7 *DVS technique for CTGs*

Table 5: Pre-processed schedule table

	true	*A*	$\bar{A}$
n_1	0, 2		
n_2	2, 4	4, 6	4, 6

the available slack time. For example, as shown in Fig. 6*c* after scaling the tasks with corresponding scaling factors, a slack time s3 is still available. This must be exploited for further energy saving. The approach in [5] is used to exploit such slack times. This is achieved by: (1) identifying the extendable tasks (in our case only n_2); (2) identifying the task, among those extendables, leading to the highest energy saving if it is extended with a certain quantum of time; (3) extending the identified task with that quantum. The three steps above are repeated until there are no slack left. Figure 6*d* is the result after exploit slack time s3.

Our DVS technique is described in Fig. 7. Step 01 pre-processes the input schedule table, so that each column corresponds to a scaling region. As discussed before, this practically means that certain tasks have to be split and distributed over several columns. Table 5 shows two lines of the schedule table resulting after pre-processing Table 3. In this case task n_2 is the one which had to be split. Steps 02-10 apply DVS to all the columns in *SchTable*, in a left-to-right sequence. For each column *col*, step 04 firstly identifies all possible tracks which will be followed after the condition values heading *col* are known; then, the track with the latest end time (the end time of the sink node in the track) is identified, which is referred as the worst case track $track_{worst}$. Step 05 calculates the worst case total slack time $slack_{worst}$ which is obtained by subtracting the end time of $track_{worst}$ from the deadline T_d. Step 06 calculates the slack time distributable to *col*, $slack_{col}$, by distributing $slack_{worst}$ to the columns along the $track_{worst}$ in proportion to the columns' duration (i.e. the difference between the latest end time and the earliest start time of the tasks in the column). Step 07 scales *col* with the *scaling_factor* given by:

$$scaling\_factor = \frac{duration_{col} + slack_{col}}{duration_{col}} \tag{3}$$

where $duration_{col}$ is the duration of *col*. Step 08 exploits the slack times on a non-critical path using the DVS technique in [5]. Due to the scaling of *col*, Step 09 must update the contents in the columns which are successive to *col* along all the possible tracks.

To illustrate the proposed DVS technique for CTGs, we apply it to the schedule table for the CTG of Fig. 1*a* and Table 1. In this case, because the columns already correspond to the scaling regions, we can simply skip step 01.

Table 6: Result after processing column *true*

	true	*A*	$\bar{A}$	$\bar{A} \wedge B$	$\bar{A} \wedge \bar{B}$
n_1	0, 11.5				
n_2		11.5, 16.5			
n_3			11.5, 15.5		
n_4				15.5, 17.5	
n_5					15.5, 21.5
n_6				17.5, 18.5	21.5, 22.5
n_7		16.5, 21.5		18.5, 23.5	22.5, 27.5

Table 7: Result after processing column *A*

	true	*A*	$\bar{A}$	$\bar{A} \wedge B$	$\bar{A} \wedge \bar{B}$
n_1	0, 11.5				
n_2		11.5, 20.75			
n_3			11.5, 15.5		
n_4				15.5, 17.5	
n_5					15.5, 21.5
n_6				17.5, 18.5	21.5, 22.5
n_7		20.75, 30		18.5, 23.5	22.5, 27.5

Then, we begin to process column *true*. Step 04: taking into account that no condition value is yet known, there are three possible tracks: track1, track 2, and track 3 (Fig. 1). Track 3 is the worst case track, where the sink node n_7 ends at 26, compared to 20 in track 1 and 22 in track 2. Step 05: since the worst case track finishes at 26 and the deadline is 30, the worst case total slack time is 4 ms. Step 06: 1.5 ms slack time is distributed to column *true* which is given by (4*(10/26)), where 10 is the column's duration and 26 is the time needed to finish the worst case track. Step 07: the task in column *true*, n_1, is scaled with the scaling factor 1.15, which is given by $((10 + 1.5)/10)$ using (3). Step 08: since there is no non-critical path in column *true*, this step can be skipped. Step 09: column *true* is a part of track 1, track 2, and track3. In track 1, column A is successive to column *true*; in track 2, columns $\bar{A}$ and $\bar{A} \wedge B$ are successive to column *true*; in track 3 columns $\bar{A}$ and $\bar{A} \wedge \bar{B}$ are successive to column *true*. Therefore, the schedules of columns A, $\bar{A}$, $\bar{A} \wedge B$, and $\bar{A} \wedge \bar{B}$ are updated due to the scaling of column *true*. Table 6 is produced after the end of Step 09. Starting with Table 6, by repeating steps 04-09 for column *A*, Table 7 is generated, where the tasks placed in column *A*, i.e. n_2 and n_7, are both extended by 4.25 ms. Considering columns $\bar{A}$, $\bar{A} \wedge B$, and $\bar{A} \wedge \bar{B}$ separately, applying steps 04-09 to them, the final schedule table is obtained as shown in Table 8.

Using Table 8, Figs. 8*a*–*c* show the actual schedules of the three possible tracks of the CTG of Fig. 1a, which meet the deadline and at the same time produce minimal energy dissipation. By comparing Fig. 4 and Fig. 8, it can be observed that the actual schedule is the same as the schedule of Fig. 4 only in the case of track 3, which is the worst case track. It is important to note that the schedules of the other tracks in Fig. 4 are impracticable! This is because the schedules in Fig. 4 are produced upon the assumption that the condition values are known before executing the disjunction nodes, which is not true during the runtime of the application. In reality, the condition values are not known until all the disjunction nodes have finished their execution. Hence, it is not possible for an online voltage

Table 8: Final schedule table

	true	*A*	$\bar{A}$	$\bar{A} \wedge B$	$\bar{A} \wedge \bar{B}$
n_1	0, 11.5				
n_2		11.5, 20.75			
n_3			11.5, 16.2		
n_4				16.2, 19.6	
n_5					16.2, 23.1
n_6				19.6, 21.3	23.1, 24.2
n_7		20.75, 30		21.3, 30	24.2, 30

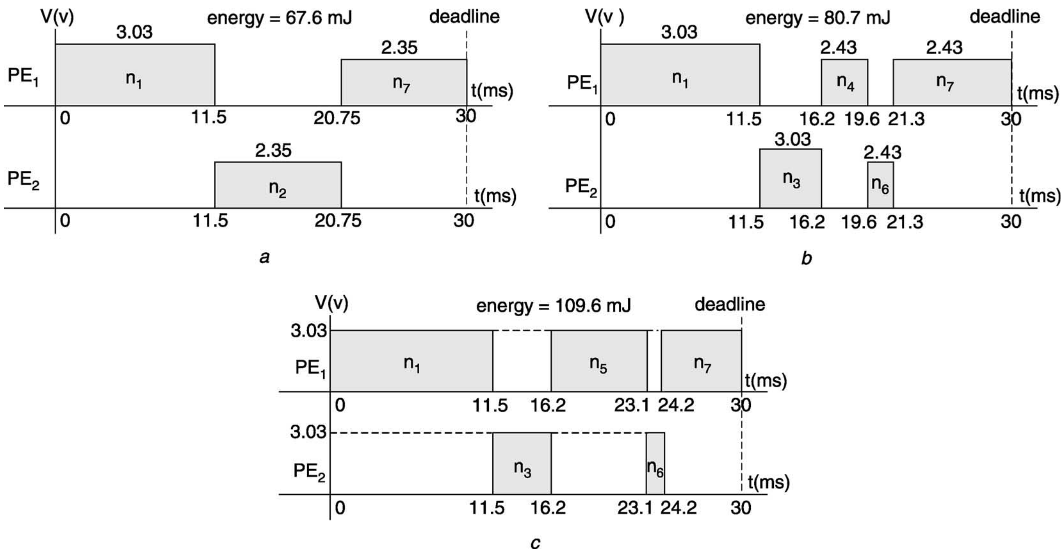

Fig. 8 *Actual schedule modified with DVS*

a Track 1
b Track 2
c Track 3

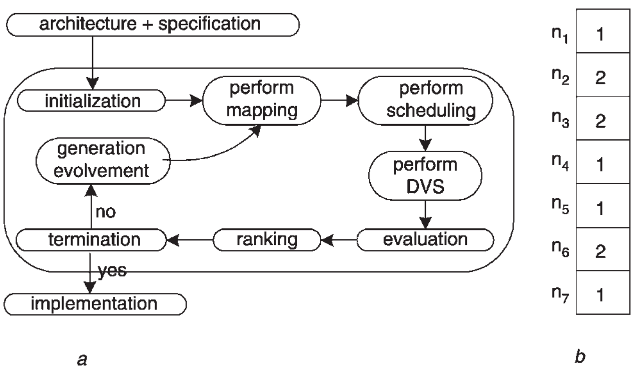

Fig. 9 *Energy-efficient mapping*

a Energy-efficient mapping
b Mapping string

scheduler to immediately use this information to achieve feasible and energy-efficient settings.

3.3 Energy-efficient mapping

In Sections 3.1 and 3.2 the DVS technique has been applied to an existing mapped and scheduled CTG. In this Section, we introduce a mapping approach specifically designed for better utilisation of DVS for CTG. Combining the mapping with the DVS technique for CTG can reduce system energy dissipation further. The flow of a mapping optimisation is shown in Fig. 9*a*. It is based on a genetic algorithm (GA) [16]. In each generation, a new population evolves from the current population by mating the fittest individuals and mutating. In our case, each individual is represented by a mapping string and represents a candidate mapping. Figure 9*b* shows a possible mapping string for the CTG of Fig. 1*a*, which means n_1 is mapped to PE1, n_2 is mapped to PE2, and so on. The algorithm constructs and evaluates many different mapping strings during an iterative optimisation process. The optimisation is guided by a fitness function. In our case, the fitness function is:

$$Fitness = \left(\sum_i E(n_i)\right) \cdot \left(\frac{\max(T_d, T_e)}{T_d}\right)^2 \qquad (4)$$

where $E(n_i)$ is the energy dissipation of task n_i, T_d is the deadline of the CTG, and T_e is the real execution time of the CTG. The first part of the fitness function is the total energy dissipation of all tasks, which has to be minimised. The second part of the function introduces a penalty factor due to deadline violations. If the length of the schedule is smaller than the deadline, the value of the second part is one: hence, no penalty is applied. In the opposite case, the squaring introduces a higher penalty to the fitness. Thus, the optimisation process is driven towards solutions with reduced energy dissipation, while, at the same time, the deadline is satisfied.

As can be seen in Fig. 9*a*, an initial population of mapping strings is first created randomly (initialisation). Then, for each individual in the population, a mapping is generated according to the mapping string (perform mapping). Next, a schedule table is produced for the mapped CTG using the scheduling algorithm in [12] (perform scheduling). After this, the schedule table is passed to the proposed DVS technique for CTG (Section 3.2) to generate a low energy schedule (perform DVS). According to the results of DVS, the fitness for the mapping string is calculated using (4) (evaluation). If no improved individual has been produced for a certain number of generations, the synthesis is stopped and the best implementation is reported. Otherwise, the synthesis continues with generation evolvement. This step implies the selection of high ranked individuals and the application of mating and mutation operators.

Mating selects a pair of high-ranked mapping strings as parent. Offspring are produced by replacing part of the first parent string with part of the second parent string. Hence, crossover results in two new offspring strings. By selecting high quality mapping strings for crossover, the chances to evolve mapping strings of higher quality are increased. Mating of the two strings is carried out with respect to an arbitrarily selected crossover point. In order to enter an unexplored region of the search space, the genetic algorithm also mutates the mapping strings occasionally with a low probability. The mutation is carried out by randomly changing an element of a randomly selected mapping string. For further details concerning the genetic algorithm based mapping see [7, 16]. The aim of this iterative process

Table 9: Results for the real-life example

Energy dissipation before DVS (mJ)	Energy dissipation after DVS (mJ) 100% deadline	105% deadline	110% deadline	120% deadline
440.00	355.15	335.61	318.28	288.87

Table 10: Results for the random examples

Example	node/edge/condition/ PE number	Energy dissipation (mJ) before DVS	after DVS
ctg1	13/16/2/2	525.00	391.29
ctg2	13/16/2/3	547.50	440.53
ctg3	13/16/3/2	625.00	548.12
ctg4	25/30/2/2	1475.00	1245.30
ctg5	25/30/2/4	1137.50	929.77
ctg6	25/30/3/2	1242.50	1131.11
ctg7	25/30/3/3	1413.75	1141.34
ctg8	25/29/4/2	1187.50	983.80
ctg9	35/41/2/2	1412.50	1122.18
ctg10	37/45/2/3	1803.75	1540.26
ctg11	35/41/2/5	1481.25	1191.05
ctg12	38/48/2/2	2072.50	1863.27
ctg13	42/52/2/4	2302.50	1921.13
ctg14	48/60/3/3	1845.00	1385.54
ctg15	59/71/3/3	3648.75	2998.32

is to finally produce an implementation which has low energy dissipation, and at the same time meets the deadline.

3.4 Experimental results

The proposed DVS and mapping technique has been tested on a number of CTG examples to demonstrate its capability in reaching high quality solutions in terms of low energy dissipation. The experiments were carried out on a Pentium III 866/256MB PC running CYGWIN. The examples consist of two sets: (1) A real-life example taken from [17]. It is a vehicle cruise controller modelled as a CTG, which consists of 32 tasks, 35 edges, and 2 conditions. The system specification has been mapped into an architecture consisting of five PEs connected through a communication bus. The initial PEs, considered in [17], are not DVS-enabled. We extended the same PEs with DVS capabilities, such that $V_t = 0.8\,v$ and $V_{max} = 3.3\,v$. (2) We have generated 15 random mapped CTG examples (ctg1 – ctg15) using the tool provided by [12], with various complexities in terms of the number of nodes, edges, conditions, and considering DVS-enabled PEs with $V_t = 0.8\,v$ and $V_{max} = 3.3\,v$.

Firstly, to test the effectiveness of the proposed DVS technique for CTG, we used the algorithm presented in [12] to generate a schedule for each example and then applied the proposed DVS technique (Section 3.2) to it. Table 9 gives the experimental results for the real-life example with different deadlines. It can be seen that the proposed DVS technique reduces the energy dissipation, and the reduction becomes greater as the deadline increases, e.g. the energy dissipation is 355.15 with a deadline of 100% of the length of the schedule produced by [12]. The energy dissipation is reduced further to 288.87 with a 120% deadline. Table 10 shows the results for the randomly generated examples with a deadline equivalent to 110% of the minimal one produced by [12]. For this experiment, the task mapping is not optimised, but we consider an implicit mapping generated randomly together with the task graph. It can be seen that, for all the examples, the proposed DVS technique reduces the energy dissipation effectively. For example, the energy dissipation of ctg1 before DVS is 525.00, and it is reduced to 391.29 after DVS; similarly, ctg10 consumes 1803.75 energy before DVS, and it is reduced to1540.26 after DVS.

We have performed another set of experiments in order to demonstrate the quality of our mapping approach. The results are shown in Table 11. Column 2 of the table shows the energy reduction when our DVS technique is applied to the mapping and scheduling solution proposed in [13].

Table 11: Results of the mapping techniques

Examples	Energy reduction (%) [13] + proposed DVS	[12] + proposed mapping & DVS	CPU time (s) [13] + proposed DVS	[12] + proposed mapping & DVS
ctg1	23.86	38.65	0.80	10.74
ctg2	22.55	42.73	0.88	35.00
ctg3	18.06	33.56	0.72	9.82
ctg4	14.07	27.21	0.86	65.08
ctg5	18.18	31.23	0.77	143.23
ctg6	15.48	31.35	0.91	85.62
ctg7	17.27	27.69	0.93	256.40
ctg8	12.92	22.62	0.79	39.22
ctg9	21.10	30.49	0.75	14.82
ctg10	19.72	28.41	0.75	26.91
ctg11	22.32	30.68	0.76	39.15
ctg12	20.23	44.84	1.22	342.06
ctg13	19.07	50.99	0.81	1777.65
ctg14	22.21	33.22	1.30	116.34
ctg15	18.04	28.85	0.99	3639.51

In column 3, we show the results obtained when the same DVS technique is applied together with the mapping and scheduling technique proposed in this paper. It can be seen that, using the GA based mapping specifically developed for DVS, the energy dissipation is reduced further, e.g. in the case of ctg12, the reduction achieved is 44.84%, that is 24.51% higher than that achieved in [13]. Table 11 also provides some information about the CPU time of the proposed DVS and mapping technique. Due to the iterative optimisation feature, the greater energy reduction achieved by our approach is at a cost of increased CPU time. The presented results show how our approach deals with CTGs with condition numbers ranging from 2 to 4 which we believe is realistic in some real-life applications. However, we have also examined the proposed approach with CTGs that have higher number of condition. It has been found that for a CTG with a condition number of 8 and task number of 125, the CPU time is approximate 3 hours mainly due to the large search space.

4 Integration of communications with scheduling and mapping

In Section 3, low power scheduling and mapping techniques for embedded systems expressed as conditional task graphs (CTGs) are presented assuming that the time and energy costs of communications between tasks are zero. This Section investigates the impacts of communications and communication link (CL) selection on system energy efficiency through a motivational example (Section 4.1).

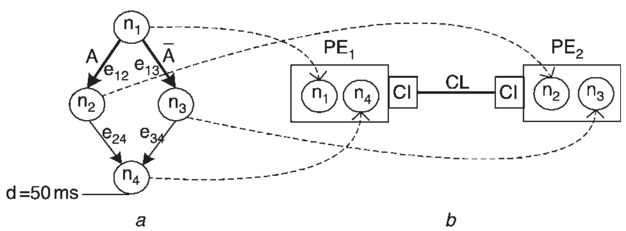

Fig. 10 *Motivational example*
a CTG
b Architecture

Table 12: Task implementation information

Task	Mapping	Execution time (ms)
n_1	PE_1	10
n_2	PE_2	20
n_3	PE_2	20
n_4	PE_1	10

Table 13: Costs of communications

	time (ms)		power (W)	
	CL	CL′	CL	CL′
e_{12}	2	0.5	1	1.5
e_{24}	2	0.5	1	1.5

The integration of communications with the presented low power scheduling and mapping is considered in Section 4.2.

4.1 *Motivational example*

Consider the CTG of Fig. 10 and its mapping to the architecture of Fig. 10*b* consisting of PE1 and PE2 connected by CL. Task implementation information is given in Table 12. Using the scheduling technique outlined in Section 3 (i.e. without consideration of communications), Figs. 11*a* and *b* show the schedules and energy dissipations when the condition value is *A*, before and after DVS has been applied. Now, assuming that communications have time and energy costs, the costs of e_{12} (communication between n_1 and n_2) and e_{24} are given in Table 13. Considering communications, Fig. 12 shows the schedules and energy dissipations before and after voltage scaling. Finally, assume that PE1 and PE2 are connected by a faster but more power-consuming communication link CL' (Table 13). It can be shown that the energy dissipaton in this case before and after voltage scaling is 201.5 mJ and 158.9 mJ respectively. Figure 13 summarises the results of this example. It can be seen that taking communications into consideration increases the energy dissipation, as expected. Furthermore, the selection of CL considerably influences the performance of the design. A system employing faster and more power-consuming CL dissipates less energy than a system employing slower and less power-consuming CL. This is because using faster CL provides more slack time for voltage scaling the tasks.

4.2 *Communication energy model and CL selection*

Most of the previous work on integrating communication within co-synthesis of embedded systems has focused on optimising area and performance. In [18, 19], the automatic generation of the necessary software and hardware for communications in embedded systems was considered. In [20], a communication model was presented to estimate the performance of CLs with different parameters, including bus width and operating frequency. The model is used to integrate communication protocol selection with hardware-software partitioning. In [12, 21], the impact of communication infrastructures and protocols on the overall performance of task scheduling was investigated. There has

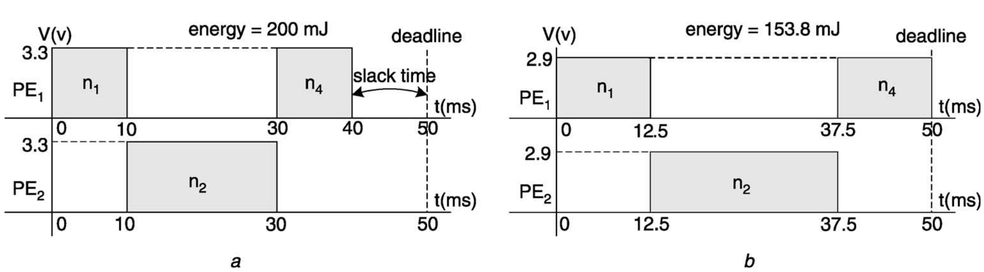

Fig. 11 *Schedule without communication costs*
a Schedule before voltage scaling
b Schedule after voltage scaling

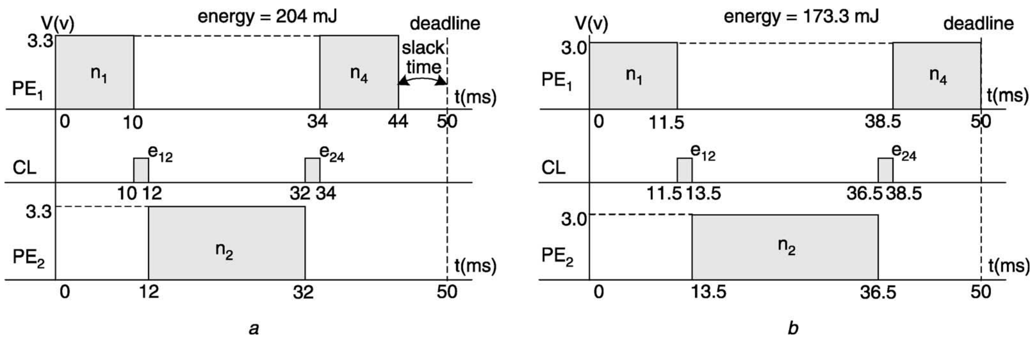

Fig. 12 *Schedule with communications costs (CL of Table 13)*
a Schedule before voltage scaling
b Schedule after voltage scaling

been some research examining the influence of communications on energy dissipations of embedded systems. In [22, 23], bus encoding techniques were proposed to reduce the energy dissipation. Some modern communication interfaces support multiple data rates. Recently in [24] a speed selection method aiming to globally optimise the energy dissipation of embedded systems was presented. This is achieved by exploiting the power/performance trade-offs between communications and computations on DVS-capable processors. In this Section, we integrate CL selection with the scheduling and mapping technique of Section 3 with the aim of reducing system energy dissipation.

A communication involves a sender and a receiver. Thus, the energy dissipation of a communication is the sum of energy dissipated by the sender and the receiver:

$$E_{com} = E_s + E_r \tag{5}$$

The energy dissipation of sender/receiver is dependent on several factors, including communication interface, communication protocol, and the size and pattern of communication data. Based on the observation that the power consumption of an interface in sleep mode is trivial compared with the power consumption in activity mode (i.e. the duration when the interface is sending/receiving data), the energy dissipation of the sender/receiver can be estimated as:

$$E = PT \tag{6}$$

where P is the power of the sender/receiver obtained from the communication interface manufacturers, and T is the time needed to send/receive the communication data. T can be estimated using the complex model outlined in [12, 20]. However in this Section, T is simply calculated by dividing the communication data size by the baud rate of CL.

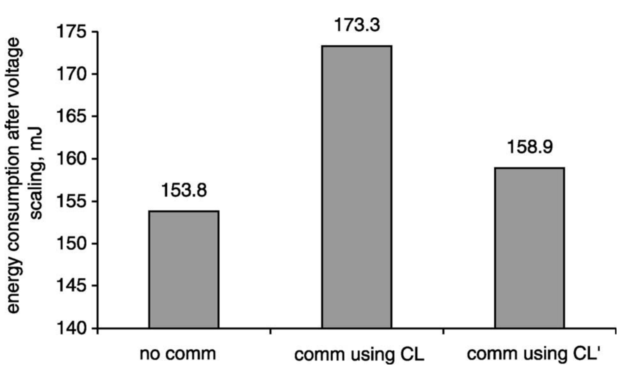

Fig. 13 *Summary of motivational example results*

Assuming that the sender and receiver use the same time for a communication, the energy dissipation of a communication is:

$$E_{com} = (P_s + P_r)T \tag{7}$$

The synthesis technique (including mapping, scheduling and voltage scaling) of Section 3 has been extended to utilise the communication energy model of (7). The aim is to select a suitable CL, which leads to minimal system energy dissipation. To allow for this selection, it is assumed that there is a library of CLs with different characteristics (power consumption and baud rate).

4.3 Experimental results

Two experiments have been performed under the assumption that PEs are communicating with each other using a single communication link. The first experiment examines system energy dissipation employing different CLs. Two CLs [25, 26] with different baud rate and power consumption are considered, where CL1 has baud rate and power consumption of 115 k bits/s and 4 mW, CL2 has baud rate and power consumption of 1 M bits/s and 35 mW. Column 4, 5 and 6 of Table 14 show the results of ten CTGs with variable complexities where CL1 and CL2 are employed. To indicate the extra energy dissipation caused by communications, column 3 gives the energy dissipations where no communications are considered. It can be seen that energy dissipation of ctg1 ~ ctg10 employing CL2 is less than where employing CL1, with up to 4.9% (ctg4) reduction, this is because for a given size of communication data, the energy dissipations (which are the products of power and time) of the two CLs are similar. However, using the faster bus (CL2) produces less communication time cost, and as a result provides more slack time for voltage scaling the tasks.

The second experiment examines the effect of increasing communication data size transmitted through CLs. Column 7, 8 and 9 of Table 14 show the results, assuming the data size is three times that of the first experiment (the data size ranges from 0 ~ 320 bits in the first experiment). As can be seen, energy reduction increases. For example, in the case of ctg2, the energy reduction increases from 4.0% to 16.8%, because where the data size is three times as large, a greater slack time difference (between using CL2 and CL1) is found than is the case with a small data size, which can be explored during DVS. Based on the above analysis, a conclusion can be drawn to the effect that CL selection involves a number of factors including communication data size and CL characteristics (baud rate and power consumption). Fast CLs tend to produce less system energy dissipation than

Table 14: Experimental results taking into account communications

	node/edge//condition/ PE number	energy w/o com (mJ)	1× com data size ene with com(mJ) CL1	CL2	reduction (%)	3× com data size ene with com(mJ) CL1	CL2	reduction (%)
ctg1	13/16/2/5	282.76	297.80	286.29	3.9	345.6	290.85	15.8
ctg2	13/16/3/5	405.07	427.85	410.91	4.0	499.21	415.12	16.8
ctg3	25/30/2/3	932.54	1015.32	973.86	4.1	1078.98	994.57	7.8
ctg4	25/30/3/3	916.32	986.56	937.98	4.9	1042.98	965.13	7.5
ctg5	25/30/3/5	773.01	812.92	777.37	4.4	910.79	826.04	9.3
ctg6	25/29/4/2	687.07	744.33	718.34	3.5	805.57	730.25	9.3
ctg7	25/29/4/4	669.76	723.45	705.44	2.5	779.70	695.96	10.7
ctg8	25/29/4/5	599.76	634.88	603.55	4.9	727.07	616.04	15.3
ctg9	35/41/2/2	873.23	894.86	877.20	2.0	952.87	882.67	7.4
ctg10	37/45/3/3	1160.75	1228.69	1174.53	4.4	1315.51	1182.14	10.1

slow CLs. This trend becomes more significant when heavier communications are needed in the system. While the presented results are based on a single CL between PEs, it is possible to employ multiple CLs of different characteristics. In this case, a communication is mapped and scheduled on a CL after its predecessor task is scheduled, such that the communication can finish as early as possible.

5 Conclusions

We have presented, for the first time, a DVS technique and an energy-efficient mapping technique for data/control dominated embedded systems expressed as CTGs. The DVS technique exploits the slack time, taking into account the conditional behaviour of a CTG. The GA based mapping produces a solution optimised for the utilisation of DVS. Combining the proposed mapping and the DVS technique for CTG with the scheduling proposed in [12], it is possible to improve the power efficiency of data/control dominated embedded systems and, at the same time, to meet the imposed deadline. Experimental results show that the proposed scheduling and mapping technique significantly reduces system energy dissipation, compared with approaches which neglect the availability of DVS, and that this optimisation can be achieved in a reasonable amount of time. It has also been shown that energy saving can be achieved using the proposed technique, taking into account the impact of communications.

6 Acknowledgments

The authors wish to thank the reviewer for providing useful comments which enhance the paper's presentation.

7 References

1 Burd, T.D., Pering, T.A., Stratakos, A.J., and Brodersen, R.W.: 'A dynamic voltage scaled microprocessor system', *IEEE J. Solid-State Circuits*, 2000, **35**, (11), pp. 1571–1580
2 Ishihara, T., and Yasuura, H.: 'Voltage scheduling problem for dynamically variable voltage processors'. Proc. 1998 Int. Symp. on Low power electronics and design, Monterey, CA, USA, 10–12 August 1998, pp. 197–202
3 Hong, I., Kirovski, D., Qu, G., Potkonjak, M., and Srivastava, M.B.: 'Power optimization of variable-voltage core-based systems', *IEEE Trans. Comput.-Aided Des. Int. Circuits Syst.*, 1999, **18**, (12), pp. 1702–1714
4 Quan, G., and Hu, X.: 'Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors'. Proc. 38th Design automation Conf., Las Vegas, NV, USA, 18–22 June 2001, pp. 828–833
5 Schmitz, M.T., and Al-Hashimi, B.M.: 'Considering power variations of DVS processing elements for energy minimisation in distributed systems'. Proc. Int. Symp. on System synthesis, Montreal, Que., Canada, 30 September–3 October 2001, pp. 250–255
6 Zhang, Y., Hu, X., and Chen, D.Z.: 'Task scheduling and voltage selection for energy minimisation'. Proc. Design automation Conf. New Orleans, LA, USA, 10–14 June 2002, pp. 183–188
7 Schmitz, M.T., Al-Hashimi, B.M., and Eles, P.: 'Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems'. Proc. Design, automation and test in Europe Conf., Paris, France, 4–8 March 2002, pp. 514–521
8 Luo, J., and Jha, N.K.: 'Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems'. Proc. IEEE/ACM Int. Conf. Computer aided design (ICCAD), San Jose, CA, USA, 5–9 November 2000, pp. 357–364
9 Gurian, F., and Kuchcinski, K.: 'LEneS: task scheduling for low-energy systems using variable supply voltage processors'. Proc. Asia and South Pacific design automation Conf. ASP-DAC 2001, Yokohama, Japan, 30 January–2 February 2001, pp. 449–455
10 Eles, P., Kuchcinski, K., Peng, Z., Doboli, A., and Pop, P.: 'Scheduling of conditional process graphs for the synthesis of embedded systems'. Proc. Design, automation and test in Europe Conf., Paris, France, 23–26 February 1998, pp. 132–138
11 Strehl, K., Thiele, L., Ziegenbein, D., Ernst, R., and Teich, J.: 'Scheduling hardware/sofrware systems using symbolic techniques'. Proc. 7th Int. Workshop on Hardware/software codesign (CODES), Rome, Italy, 3–5 May 1999, pp. 173–177
12 Eles, P., Doboli, A., Pop, P., and Peng, Z.: 'Scheduling with bus access optimization for distributed embedded systems', *IEEE Trans. Very Large Scale Integr. (VLSI) Sys.*, 2000, **8**, (5), pp. 472–491
13 Xie, Y., and Wolf, W.: 'Allocation and scheduling of conditional task graph in hardware/software co-synthesis'. Proc. Design, automation and test in Europe Conf., Munich, Germany, 13–16 March 2001, pp. 620–625
14 Chakraborty, S., Erlebach, T., Kunzli, S., and Thiele, L.: 'Schedulability of event-driven code blocks in real-time embedded systems'. Proc. 2002 Design automation Conf., New Orleans, LA, USA, 10–14 June 2002, pp. 616–621
15 Schmitz, M.T., Al-Hashimi, B.M., and Eles, P.: 'Synthesizing energy-efficient embedded systems with LOPOCOS', *Des. Autom. Embedded Syst.*, 2002, **6**, (4), pp. 401–424
16 Dick, R.P., and Jha, N.K.: 'MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 1998, **17**, (10), pp. 920–935
17 Pop, P.: 'Scheduling and communication synthesis for distributed real-time systems'. Licentiate thesis, (Linkopings University, 2000)
18 Ortega, R.B., and Borriello, G.: 'Communication synthesis for embedded systems with global considerations'. Proc. 5th Int. Workshop on Hardware/software codesign, Braunschweig, Germany, 24–26 March 1997, pp. 69–73
19 Ortega, R.B., and Borriello, G.: 'Communication synthesis for distributed embedded systems'. Proc. IEEE/ACM Int. Conf. Computer aided design, San Jose, CA, USA, 8–12 November 1998, pp. 437–444
20 Knudsen, P.V., and Madsen, J.: 'Integrating communication protocol selection with hardware/software codesign', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 1999, **18**, (8), pp. 1077–1095
21 Pop, P., Eles, P., and Peng, Z.: 'Scheduling with optimized communication for time-triggered embedded systems'. Proc. 7th Int. Workshop on Hardware/software codesign, Rome, Italy, 3–5 May 1999, pp. 178–182
22 Stan, M.R., and Burleson, W.P.: 'Bus-invert coding for low-power I/O', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 1995, **3**, (1), pp. 49–58

23 Benini, L., De-Micheli, G., Macii, E., Sciuto, D., and Silvano, C.: 'Address bus encoding techniques for system-level power optimization'. Proc. Design, automation and test in Europe, Paris, France, 23–26 February 1998, pp. 861–866
24 Liu, J., Chou, P.H., and Bagherzadeh, N.: 'Communication speed selection for embedded systems with network voltage-scalable processors'. Proc. 10th Int. Symp. on Hardware/software codesign, Estes Park, CO, USA, 6–8 May 2002, pp. 169–174
25 Philips, 'SCC2691 universal asynchronous receiver/transmitter (UART)', 1995
26 Philips, 'SC28L91 3.3V-5.0V universal asynchronous receiver/transmitter (UART)', 2000