Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems

Dong Wu and Bashir M. Al-Hashimi
Dept. of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
{dw00r, bmah}@ecs.soton.ac.uk

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

Abstract

This paper describes a new Dynamic Voltage Scaling (DVS) technique for embedded systems expressed as Conditional Task Graphs (CTGs). The idea is to identify and exploit the available worst case slack time, taking into account the conditional behaviour of CTGs. Also we examine the effect of combining a genetic algorithm based mapping with the DVS technique for CTGs and show that further energy reduction can be obtained. The techniques have been tested on a number of CTGs including a real-life example. The results show that the DVS technique can be applied to CTGs with energy saving up to 24%. Furthermore it is shown that savings of up to 51% are achieved by considering DVS during the mapping.

1 Introduction and related work

Energy efficiency is becoming an essential issue in embedded system synthesis, for reasons like the increasing demand for portable devices or the heat dissipation caused by excessive power consumption which may lead to reduced reliability. One possible and effective technique for decreasing power consumption of 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 [3]. DVS exploits the slack time, i.e. the intervals when a PE is idle, 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, 4-7]. 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[8]. In [9], a mobility based list scheduling was modified towards DVS utilization. They optimise a static schedule towards the incorporation of aperiodic tasks. The static schedule provides guidelines to the online scheduler. In [10], a DVS optimized schedule was derived using a constructive list scheduling technique with a dynamic re-calculation of task priorities based on average energy dissipation. In [8], a two-step iterative synthesis approach guided by a generalised DVS algorithm was presented. Their approach optimizes both the mapping and schedule towards energy efficiency by abetting the exploitation of DVS.

All 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 aspect 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 [11, 12]. In [1, 11] such an abstract system representation, called Conditional Task Graph (CTG), has been defined and a scheduling algorithm has been proposed so that the worst case delay is minimized. 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 papers like [1, 11, 13, 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 consumption. Nevertheless, no work has still 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 two main contributions of this work:

1. A novel 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 optimize the system implementation to efficiently exploit the proposed DVS technique, hence, leading to further energy savings.

2 Preliminaries

2.1 CTG and architectural model

We consider that an application is specified as a directed, acyclic graph $G(V, E_S, E_C)$ called *conditional task graph* (CTG) [1]. Figure 1(a) shows an example CTG. Each node, $n_i \in V$ represents a task, an atomic unit to be executed without being preempted. There are two nodes, called *source* and *sink*, which represent the first and last node respectively, so 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 Figure 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 Figure 1(a), executing n_1 produces *condition value A* or $\overline{A}$. 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 that may be followed at a certain execution. The CTG of Figure 1(a) has three possible tracks, which are shown in Figures 1(b)-(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 has to be, in the worst

case, smaller that 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. 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 solved 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 processes. For further details concerning the CTG representation the reader is referred to [1].

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. We consider continuous voltages here, but these can be easily adapted to the case with discrete voltages[8]. 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 to neglect the scaling overhead in terms of power and time. Furthermore, the PEs might employ power management techniques, i.e. they might shut down themselves when they are idle. An infrastructure of communication links (CLs) connects these PEs through communication interfaces, which are able to adapt to the different operational frequencies caused by DVS.

The schedule table produced by [1] captures all the details related not only to task activation but also to communication scheduling. In [8, 17] we have also shown how communication aspects have to be considered for scheduling and mapping with DVS. In order to concentrate on the specific aspects of importance for this paper and considering the space limitation, in this presentation we will make a simplifying assumption that communications take 0 time and consume 0 energy. However, all the algorithms and the conclusions of this paper are equally valid if communications are taken into consideration.

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

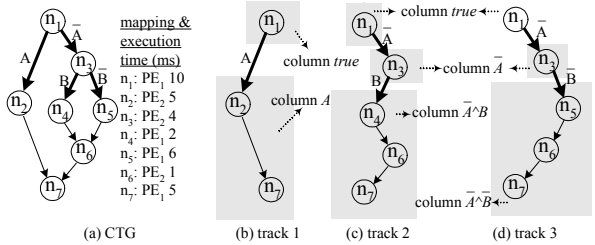


Figure 1. Conditional task graph and its tracks

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 [1] a scheduling algorithm is proposed for mapped conditional task graphs so that 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 Figure 1(a), assuming task mappings and task execution times as shown in the figure. The table has one row for each task, which contains start 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 Figure 1, there are 3 possible tracks. The schedule of track 1 is represented in columns *true* and A. The schedule of track 2 is captured in columns *true*, $\bar{A}$, and $\bar{A} \wedge B$. The schedule of track 3 is given in columns *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 that could be taken off line have been 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 transmission of messages, based on the current values of conditions.

cond values	*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 1. Schedule table

The problem formulation can 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 consumption is minimized. The execution of a CTG can proceed along different tracks depending on the actual condition values. Our objective is to minimize the total energy consumption assuming that every track is executed with equal probability.

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 by [8]:

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

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

where V_{max} is the maximum supply voltage, $E(V_{max})$ and $d(V_{max})$ is the energy dissipation and execution time at V_{max}, 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 off-line task scheduling is based on the assumption that a certain slack time is available and 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 side, 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

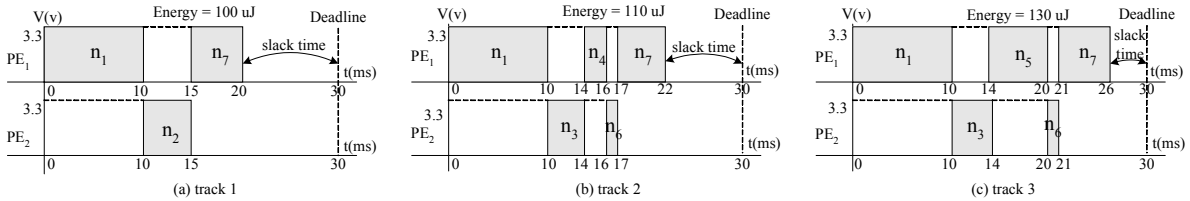


Figure 2. Schedule of the CTG of Figure 1(a)

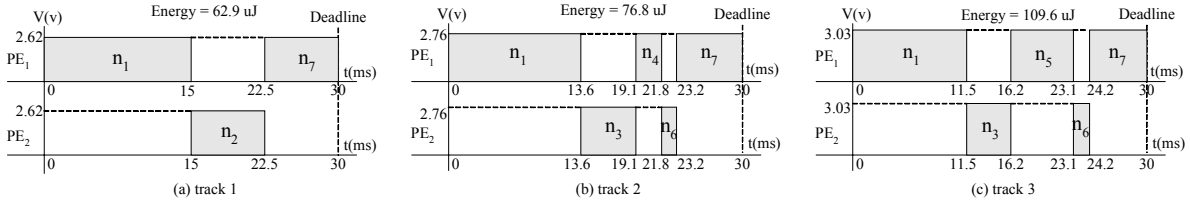


Figure 3. Schedule scaled for energy minimisation

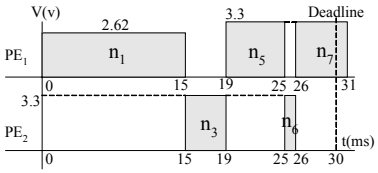


Figure 4. Improper scaling

quasistatic schedule with voltage scaling for CTGs, we consider the CTG of Figure 1(a). Let us assume that the deadline of the system is 30ms. Figures 2(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 [1], where the aim is to produce a schedule such that the worst case delay is as small as possible. As can be seen from Figure 2, the amount of slack time varies with the tracks, ranging from 4ms in the case of track 3 to 10ms in the case of track 1, since the deadline of 30ms is not to be exceeded. Figures 2(a)-(c) also show the energy dissipation of each track, assuming a supply voltage of 3.3V. In order to make use of DVS techniques for energy minimisation, a well known technique [4, 10] is to scale the schedules such that they, as much as possible, fit the imposed deadlines. 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 isolated from the others, is given in Figures 3(a)-(c). 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, in the case of 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 be conflicted, which cannot be tolerated in systems with hard-real time properties. For example, as shown in Figure 4, if n_1 is decided to run from 0 to 15, and the condition values come out to be $\overline{A} \wedge \overline{B}$ 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 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 modifiing 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 [1] whose aim is to make the worst case delay as small as possible. What we produce is a slack time exploited schedule table 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 to exploit the information concerning condition values, available at a certain time, in order to apply the largest possible scaling factor while still guarantee the deadline. The point in time when 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 found that the schedules of the tasks in each column correspond to such a scaling region. However, a column of the initial schedule table has not necessarily to 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. We illustrate such a situation with the CTG in Figure 5(a). Figure 5(b) presents the schedule of the track corresponding to condition value *A* according to the schedule table in Table 2. 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 Figure 5(c). The corresponding scaled schedule Table is shown in Table 3. It can be observed 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 basic 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 the available slack time. For example, as shown in Figure 5(c), after scaling the tasks with corresponding scaling factors, a slack time s3 is still available. This has to be exploited for further energy saving. The approach in [2] 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 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 5(d) is the result after exploit slack time s3.

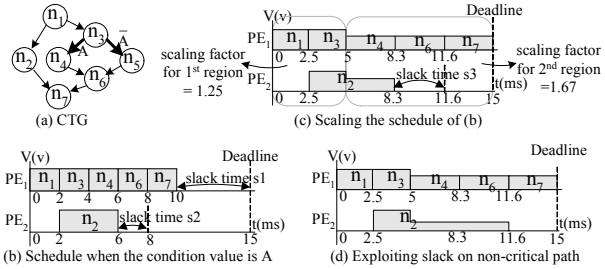


Fig 5. a CTG example and its scaling

cond values	*true*	A	$\overline{A}$
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 2. Original schedule table

cond values	*true*	A	$\overline{A}$
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

Table 3. Scaled schedule table

Our DVS technique is described in Figure 6. Step 01 pre-processes the input schedule table, so that each column corresponds to a scaling region. As discussed before, these practically means that certain tasks have to be split and distributed over several columns. Table 4 shows two lines of the schedule table resulted after pre-processing Table 2. In this case task n_2 is the one that 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 that 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:

$$scaling\_factor = \frac{duration_{col} + slack_{col}}{duration_{col}} \quad (3)$$

where $duration_{col}$ is the duration of *col*. Step 08 exploits the slack times on non-critical path using the DVS technique in [2]. Due to the scaling of *col*, Step 09 has to update the contents in the columns that are successive to *col* along all the possible tracks.

DVS technique for CTGs
Input: a schedule table generated by [1] – *SchTable*
deadline - T_d
Output: a slack time exploited schedule table indicating voltage levels and activation times - *ScaledSchTable*
01 pre-process *SchTable*
02 for (each column *col* in *SchTable*, from left to right)
03 {
04 identify the worst case track - $track_{worst}$
05 calculate the worst case total slack time - $slack_{worst}$
06 calculate the slack time distributable to *col* - $slack_{col}$
07 scale *col* with *scaling_factor* given by Equation (3)
08 apply DVS technique in [2] to *col*
09 update *SchTable*
10 }

Figure 6. DVS technique for CTGs

cond values	*true*	A	$\overline{A}$
n_1	0, 2		
n_2	2, 4	4, 6	4, 6

Table 4. Pre-processed schedule table

To illustrate the proposed DVS technique for CTGs, we apply it to the schedule table for the CTG of Figure 1(a), Table 1. In this case, because the columns already correspond to the scaling regions, we can simply skip step 01. Then we begin to process column *true*. Step 04: taking into account that no condition value is yet known, there are 3 possible tracks: track1, track 2, and track 3 (see Figure 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 Equation (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 $\overline{A}$ and $\overline{A} \wedge B$ are successive to column *true*; in track 3 columns $\overline{A}$ and $\overline{A} \wedge \overline{B}$ are successive to column *true*. Therefore the schedules of columns A, $\overline{A}$, $\overline{A} \wedge B$, and $\overline{A} \wedge \overline{B}$ are updated due to the scaling of column *true*. Table 5 is produced after the end of Step 09. Starting with Table 5, repeating steps 04-09 to columns A, $\overline{A}$, $\overline{A} \wedge B$, and $\overline{A} \wedge \overline{B}$ separately, the final schedule table is obtained as in Table 6.

Using Table 6, Figures 7(a)-(c) show the actual schedules of the three possible tracks of the CTG of Figure 1(a), which meet the deadline and at the same time produce minimal energy dissipation. By comparing Figure 3 and Figure 7, it can be observed that the actual schedule is the same as the schedule of Figure 3 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 Figure 3 are impracticable! This is because the schedules in Figure 3 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

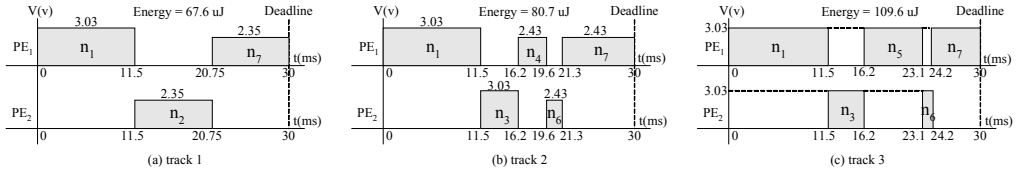


Figure 7. Actual schedule modified with DVS

	true	*A*	$\overline{A}$	$\overline{A}$ ^ *B*	$\overline{A}$ ^ $\overline{B}$
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 5. Result after processing column *true*

	true	*A*	$\overline{A}$	$\overline{A}$ ^ *B*	$\overline{A}$ ^ $\overline{B}$
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

Table 6. Final schedule table

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

3.3 Energy-Efficient Mapping

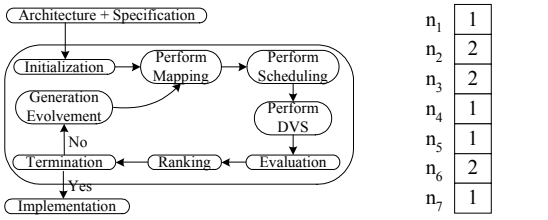
In Sections 3.1 and 3.2 the DVS technique has been applied to an existing mapped and scheduled CTG. In this section, we introduce a mapping approach specifically designed for better utilization of DVS for CTG. Combining the mapping with the DVS technique for CTG can reduce system energy dissipation further. The flow of a mapping optimisation is shown in Figure 8(a). It is based on a genetic algorithm (GA) [15]. 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 8(b) shows a possible mapping string for the CTG of Figure 1(a), which means n_1 is mapped to PE_1, n_2 is mapped to PE_2, 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, 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 1, 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 consumption, while, at the same time, the deadline is satisfied.

As can be observed in Figure 8(a), firstly, an initial population of mapping strings is created randomly (Initialization). 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 [1] (Perform Scheduling). After this, the schedule table is passed to the proposed DVS technique for CTG (see 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 Equation (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.

The aim of this iterative process is to finally produce an implementation that has low energy dissipation, and at the same time meets the deadline.



(a) energy-efficient mapping (b) mapping string

Figure 8. Energy-efficient mapping

4 Experimental Results

The proposed DVS and mapping technique has been tested on a number of CTG examples to demonstrate their capability to produce 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 [16]. 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 5 PEs connected through a communication bus. The initial PEs, considered in [16], are not DVS-enabled. We extended the same PEs with DVS capabilities, such that V_t=0.8v and V_{max}=3.3v. (2) We have generated 15 random mapped CTG examples (ctg1 – ctg15) using the tool provided by [1], with various complexities in terms of the number of nodes, edges, conditions, and considering DVS-enabled PEs with V_t=0.8v and V_{max}=3.3v.

Firstly, to test the effectiveness of the proposed DVS technique for CTG, we use the algorithm presented in [1] to generate a schedule for each example and then apply the proposed DVS technique (see Section 3.2) to it. Table 7 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 higher 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 [1]. The energy dissipation is reduced further to 288.87 with a 120% deadline. Table 8 shows the results for the randomly generated examples with a deadline equivalent to 110% of the minimal one produced by [1]. For this experiment, the task mapping has not been optimized, but we considered 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 9. Column 2 of the table shows the energy reduction when our DVS technique is applied to the mapping and scheduling solution proposed in [13]. 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 observed that, using the GA based mapping specifically developed for DVS, the energy dissipation is reduced further, e.g. in the case of ctg12, the achieved reductions is 44.84%, that is 24.51% higher than the approach of [13]. Table 9 also provides some information about the CPU time of the proposed DVS and mapping technique. Due to the iterative optimization feature, the higher energy reduction achieved by our approach is at the cost of increased CPU time.

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

Table 7. Results for the real-life example

Example	node/edge/condition/PE number	Energy dissipation	
		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

Table 8. Results for the random examples

Examples	Energy reduction (%)		CPU time (s)	
	[13]+ proposed DVS	[1]+proposed mapping & DVS	[13]+ proposed DVS	[1]+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

Table 9. Results of the DVS the mapping techniques

5 Conclusions

We have presented, for the first time, a novel DVS technique and an energy-efficient mapping technique for data/control dominated embedded systems expressed as conditional task graphs. The DVS technique exploits the slack time taking into account the conditional behaviour of a CTG. The GA based mapping produces a solution optimized for the utilization of DVS. Combining the proposed mapping and the DVS technique for CTG with the scheduling proposed in [1], it is possible to improve the power efficiency of the data/control dominated embedded systems and, at the same time, to meet the imposed deadline. Experimental results show that the proposed DVS technique significantly reduces the system energy dissipation, compared to the approaches which neglect the availability of DVS, and that this optimisation can be achieved in a reasonable amount of time. Current work undertaken by the authors examines the influences of communications on the synthesis of low power embedded systems expressed as CTGs.

References:

[1] P. Eles, A. Doboli, P. Pop and Z. Peng, "Scheduling with bus access optimization for distributed embedded systems," *IEEE Trans. on VLSI,* vol.8, no.5, pp.472-91, Oct. 2000.

[2] M. T. Schmitz and B. M. Al-Hashimi, "Considering power variations of DVS processing elements for energy minimisation in distributed systems," in Proc. ISSS'01, pp.250-255, Montreal, Canada, 2001.

[3] T. D. Burd, T. A. Pering, A. J. Stratakos and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE Journal of Solid State Circuits*, vol.35, no.11, pp.1571-80, Nov. 2000.

[4] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in Proc. ISLPED'98, pp.197-202, Monterey, CA, USA, 1998.

[5] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in Proc. DAC'01, pp.828-833, Las Vegas, NV, USA, 2001.

[6] I. Hong, D. Kirovski, G. Qu, M. Potkonjak and M. B. Srivastava, "Power optimization of variable-voltage core-based systems," *IEEE Trans. on CAD*, vol.18, no.12, pp.1702-14, Dec. 1999.

[7] Y. Zhang, X. Hu and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in Proc. DAC'02, pp.183-8, New Orleans, Louisiana, USA, 2002.

[8] M. T. Schmitz, B. M. Al-Hashimi and Petru Eles, "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems," in Proc. DATE'02, pp.514-21, Paris, France, 2002.

[9] 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'00, pp.357-364, San Jose, CA, USA, 2000.

[10] F. Gruian and K. Kuchcinski, "LEneS: task scheduling for low-energy systems using variable supply voltage processors," in Proc. ASP-DAC'01, pp.449-55, Yokohama, Japan, 2001.

[11] P. Eles, K. Kuchcinski, Z. Peng, A. Doboli and P. Pop, "Scheduling of conditional process graphs for the synthesis of embedded systems," in Proc. DATE'98, pp.132-38, Paris, France, 1998.

[12] K. Strehl, L. Thiele, D. Ziegenbein, R. Ernst and J. Teich, "Scheduling hardware/software systems using symbolic techniques," in Proc. CODES'99, pp.173-177, Rome, Italy, 1999.

[13] Y. Xie and W. Wolf, "Allocation and scheduling of conditional task graph in hardware/software co-synthesis," in Proc. DATE'01, pp.620-625, Munich, Germany, 2001.

[14] S. Chakraborty, T. Erlebach, S. Kunzli and L. Thiele, "Schedulability of event-driven code blocks in real-time embedded systems," in Proc. DAC'02, pp.616-21, New Orleans, USA, 2002.

[15] R. P. Dick and N. K. Jha, "MOGAC: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," *IEEE Trans. on CAD*, vol.17, no.10, pp.920-35, Oct. 1998.

[16] P. Pop, "Scheduling and communication synthesis for distributed real-time systems," Licentiate thesis, Linköpings University, Sweden, 2000.

[17] M. T. Schmitz, B. M. Al-Hashimi and Petru Eles, "Synthesizing energy-efficient embedded systems with LOPOCOS," *Design Automation for Embedded Systems*, vol. 6, pp.401-24, 2002.