AN ENERGY-EFFICIENT HEURISTIC FOR HARD REAL-TIME SYSTEM ON MULTI-CORE PROCESSORS

Da He  
University of Paderborn/C-Lab  
Fuerstenallee 11, 33102 Paderborn, Germany

Wolfgang Mueller  
University of Paderborn/C-Lab  
Fuerstenallee 11, 33102 Paderborn, Germany

ABSTRACT
In the electronic system development, energy consumption is clearly becoming one of the most important design concerns. From the system level point of view, Dynamic Power Management (DPM) and Dynamic Voltage and Frequency Scaling (DVFS) are two mostly applied techniques to adjust the tradeoff between the performance and power dissipation at runtime. In this paper, we study the problem of combined application of both techniques with regard to hard real-time systems running on cluster-based multi-core processors. To optimize the processor energy consumption, a heuristic based on simulated annealing with efficient termination criterion is proposed. The experiment results show that the proposed algorithm outperforms the existing approaches in terms of the energy reduction.

KEYWORDS
Dynamic Power Management, Dynamic Voltage and Frequency Scaling, Hard Real-Time, Multi-core Processor

1. INTRODUCTION
There are two mostly applied runtime energy reduction techniques for processors: Dynamic Power Management (DPM) and Dynamic Voltage and Frequency Scaling (DVFS). Generally, DPM tries to selectively shut down the processor cores when they are idle while DVFS slows down their operating speed or frequency. In fact, both techniques are working contradictory. The DPM preferred strategy will run the tasks as soon as possible, so that more idle time can be used for sleeping. On the contrary, the DVFS preferred algorithm will run the tasks as slow as possible, because lower operating speed is more energy efficient (in CMOS-based circuits). The main challenge is to find the optimal tradeoff between them. Another aspect of this paper is concerning real-time systems. Due to the hard timing constraints, a real-time task cannot be always executed at the lowest speed. The main problem is to find the most energy efficient speed for all the tasks that ensures the system schedulability.

Additionally, this paper focuses on multi-core processors. In terms of DPM and DVFS capabilities, the multi-core processors can be divided into three categories: per-core platform, full-chip platform and cluster-based platform. In per-core platform, the processor cores are independent with regard to DPM and DVFS usage, i.e. the shutdown or slowdown of one core will not affect the other cores. Clearly, this platform provides the most opportunity and flexibility of energy management, however, it suffers high implementation effort, because each core needs a dedicated on-chip voltage regulator. In contrast, full-chip platform admits a cost-efficient solution, where a common voltage regulator is shared by all the processor cores. However, the cores can only operate at the same speed at a time. With other words, DVFS technique is applied at the whole processor chip level. Note that DPM technique is still applied on the core basis. As a generalized form of per-core and full-chip platform, the cluster-based platform groups the processor cores into clusters. The cores in the same cluster share the same DVFS capability and the cores from different clusters can be controlled independently. In this paper, we focus on this emerging cluster-based multi-core platform. Furthermore, there are two types of real-time scheduling on multi-core platforms [Davis&Burns 2011]: partitioned scheduling and
global scheduling. This work applies the partitioned scheduling, because it provides the advantage that the tradition singe-core processor real-time scheduling, such as Earliest Deadline First (EDF) or Rate Monotonic (RM) can be adopted.

In order to optimal apply DPM and DVFS on multi-core processors, the main challenge is to find the most appropriate task partition and speed assignment, which minimize the total processor energy consumption and ensure system schedulability. Unfortunately, this optimization problem has been proven to be NP-hard [Aydin&Yang 2003]. Thus, one of our main contributions in this paper is to propose a simulated annealing based heuristic to deal with the problem. Since the proposed heuristic is an iterative algorithm, we additionally propose an efficient termination criterion using exponential regression technique.

The remainder of this paper is organized as follows. Section 2 gives an overview of related work. In section 3 we formally define the system model and the problem. Section 4 describes the detail of the heuristic search algorithm and the termination criterion is presented in section 5. Finally, before we conclude the paper in the section 7, the experiment results are presented in section 6.

2. RELATED WORK

In the area of energy aware real-time scheduling for multi-core processors, the existing work can be roughly divided into three categories in terms of the addressed platforms. [Aydin&Yang 2003] proved that the energy-aware task partitioning is NP-hard and proposed a load balancing approach for per-core platform. With regard to leakage power, [Chen et al. 2006] described several approximation algorithms on the per-core platforms, but their analysis is based on the ideal DVFS model, where the processor core may operate at any frequency within a given range. Moreover, [Lee 2009] classified the tasks into three categories according to their utilization and proposed a heuristic scheduling scheme based on the classification. In the domain of full-chip platform, most work are concentrating on balanced task partition. A dynamic repartitioning algorithm based on the existing partition is introduced in [Seo et al. 2008]. The goal is to balance the task load on different cores by considering dynamic slack. They further improved the algorithm by a dynamic core scaling scheme to shut down the unused cores. [Yang et al. 2005] introduced an approximation algorithm on the full-chip platform, however, they assumed the framed-based task model, where all the tasks share the same deadline and period.

Only recently, the cluster-based platform attracts more and more attentions. [Kolpe et al. 2011] has focused on the problem of clustering the cores into DVFS domains. The authors proposed to group similar cores into cluster according to their typical workload. [Chakraborty&Roy 2011] introduced a fundamentally alternate means for cluster-based multi-core processor design. They believe that a core, which is designed for a dedicated frequency/voltage domain, is more energy efficient than a core designed with runtime DVFS capabilities and configured to that frequency/voltage domain. Both work are not focusing on hard real-time systems. The closest related work are presented by [Kong et al. 2011] and [Qi&Zhu 2008]. Both of them considered the cluster-based multi-core platforms and hard real-time systems, however, they both assumed frame-based task model and ideal DVFS model. In this work, we will address a more general task model and non-ideal processor model, which are more common in the reality.

3. SYSTEM MODEL AND PROBLEM FORMULATION

The power model of a cluster-based multi-core processor is defined as follows. The cores in the same cluster share the same power model and the cores from different clusters could possess different power models. The cores in the same cluster must operate at the same frequency. However, each core can enter sleep state independently. We define $D$ as a set of processor cores. For each core $D_{x,y} \in D$, where $x$ is the cluster index and $y$ is the core index, we define two power states (DPM states) as $C_0$ and $C_1$, where $C_0$ is the active state and $C_1$ is the low power (sleeping) state. In addition, the active state $C_0$ contains a set of DVFS states denoted by $S = \{S_1, S_2, ..., S_s\}$, where $S_1$ is the full performance state and the remaining states are in non-increasing order of operating frequency. \(\forall i: 1 \leq i \leq s\), we denote $F(S_i)$ and $P(S_i)$ as the frequency and power consumption of the state $S_i$, respectively. Furthermore, the critical speed is denoted as $S_{acr}$, which is the most power efficient DVFS state [Niu&Li 2011], i.e. with the smallest $P(S_i)/F(S_i)$ (watt per hertz).
Since we are interested in hard real-time systems with independent periodic tasks, we adopt the classic real-time task model. The task set is denoted by $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ with $\tau_i = (W_i, T_i)$ where $W_i$ denotes the Worst Case Execution Time (WCET) at maximal speed and $T_i$ is the relative deadline (equal to period) of the task. The hyper period of a task set is the least common multiple of all task periods. Note that the execution of tasks repeat themselves in each hyper period. Furthermore, we assume that the task WCET increases linearly when the processor speed decreases.

The problem of energy optimization of hard real-time systems on multi-core platforms is composed of three parts: the task partitioning, the speed assignment to each task and the task scheduling. In order to find the optimal solution, all parts should be considered coherently, because they have influence on each other. In this article, we concentrate on the first two parts and assume that the last part is solved by a given real-time schedule like EDF or RM. The main advantage is that our approach can be built on top of those well-established schedules. Thus, our problem is to find the solution containing a task partition function $alloc: \Gamma \rightarrow D$ and a speed assignment function $assign: \Gamma \rightarrow S$, which result in the minimal processor energy consumption over a hyper period while ensuring the system schedulability. Furthermore, we refer to the processor energy consumption over one hyper period as the solution value.

4. **HEURISTIC SEARCH ALGORITHM**

Unfortunately, the previously presented problem is NP-hard [Aydin&Yang 2003]. Therefore we propose a Simulated Annealing (SA) based heuristic search algorithm. The SA is motivated by the annealing process in material science and designed to solve optimization problem in general. Its main concept is starting with an arbitrary solution and iteratively improving the solution by investigating neighbor solutions. If a neighbor solution is better, then a movement is made and the neighbor solution becomes the current solution. One of the main features of SA is being able to accept worse solutions, i.e. the movement to a worse neighbor solution is also possible, however, only with a certain probability. This allows that SA may escape from a local optimum. In general, SA algorithm is problem-independent and applicable for a large variety of problems. Usually, the neighbors are randomly selected by uniform probability. Even though in many cases the generalized SA already produces excellent results, we believe that there is still great improvement potential by adapting the generalized SA to a particular problem. Our idea is to apply problem specific and heuristic information to guide the selection of particular neighbors, so that, hopefully, the SA can converge to an optimal solution more rapidly. In our context, two solutions are in a neighborhood, if they differ in the configuration of exactly one task, i.e. the core allocation and the frequency assignment. A neighbor solution is generated in three steps: (i) select a task (ii) reallocate it to a new core and (iii) reassign it with a new frequency.

For the task selection (the first step) we define the guidance based on the heuristic information learned from the current solution. More precisely, each task $\tau_i$ is associated with a penalty value denoted by $pen(\tau_i)$. Intuitively, the penalty value of a task indicates the wasted energy by the task. The higher the penalty value, the more possible the corresponding task is to be selected for configuration change. $pen(\tau_i)$ is formally computed by equation (1) which is composed of three parts (on the right side of the equation). The first part expresses the wasted active energy during the task execution. According to the definition, the critical speed (CS) is the most power efficient speed with regard to the active power, therefore, the corresponding task is penalized depending on the difference between its current speed and the CS. The second part describes the unbalanced task execution in a cluster. More specifically, $t_{unbalanced}(\tau_i)$ denotes the time (inside a hyper period), where a core executes $\tau_i$ and at least one of the other cores in the same cluster is enforced to operate at the same speed, though the originally required speed is lower. Obviously, the task with larger $t_{unbalanced}(\tau_i)$ should be selected more likely for configuration change. Figure 1 shows one example with two cores in one cluster. In this example we assume that $S_1$ is assigned to $\tau_1$ and $S_2$ to $\tau_2$, respectively. We observe the time interval between 0ms and 10ms, where the task $\tau_2$ is executed on the core $D_{1,2}$ with $S_1$, even though it is originally assigned with $S_2$. This is mainly due the fact that both cores are in the same cluster and can therefore only operate at the same speed. In case of a speed conflict, we always select the highest one, because we can then ensure that no task will finish later than its original WCET and thus the schedulability. On some platforms, this strategy is even enforced by the hardware constraints, e.g. Intel Core2Dual [Intel]. As this unbalanced execution is caused by $\tau_1$, $t_{unbalanced}(\tau_1)$ is set equal to 10ms. The constants $\lambda_1$ and $\lambda_2$ serve as the coefficients to adjust the impact of the first and the second part, respectively. The constant $\lambda_3$ is a technique...
parameter in order to prevent the penalty value from being zero. Based on the penalty values we derive the selection probability $prob(\tau_i)$ of in the equation (2).

$$pen(\tau_i) = \lambda_1 \times |F(\text{assign}(\tau_i)) - F(S_{sc})| + \lambda_2 \times t_{\text{unbalanced}}(\tau_i) + \lambda_3$$  

$$prob(\tau_i) = pen(\tau_i) / \sum_{\tau_j \in \text{FT}} pen(\tau_j)$$  

(1)  

(2)

After a task is selected, in the second step we need to reallocate it by selecting a new core. Hereby we would like to balance the processor load and avoid generating invalid solution, where the system schedulability is not ensured. For this purpose we associate each processor core $D_{x,y}$ with a reward value $rew(D_{x,y})$, which is simply the available utilization that is still free to be used. Formally we define the reward function in equation (3).

$$rew(D_{x,y}) = \begin{cases} 
U_{ub} - U_{D_{x,y}} & \text{if } \text{alloc}(\tau_k) \neq D_{x,y} \\
U_{ub} - U_{D_{x,y}} + U_k & \text{otherwise}
\end{cases}$$  

(3)

$U_{ub}$ is the upper bound of the total task utilization, so that all tasks are schedulable under a given schedule, e.g. $U_{ub}$ is 1 and 0.69 for EDF and RM, respectively. $U_{D_{x,y}}$ is the total task utilization on the core $D_{x,y}$ and $U_k$ is the utilization of the task $\tau_k$, assuming that the $\tau_k$ is selected in the first step. Analogous to the task selection, based on the reward values we derive the selection probability $prob(D_{x,y})$ of each core in equation (4).

$$prob(D_{x,y}) = rew(D_{x,y}) / \sum_{D_{x,y}} rew(D_{x,y})$$  

(4)

Intuitively, the core with more available utilization will be selected more likely. Finally, in the third step a new frequency is randomly selected and assigned to the selected task according to the uniform distribution of all possible frequencies. In addition, we take the algorithm LA+LTF+FF [Chen et al. 2006] to compute the initial solution of the heuristic. LA+LTF+FF is an 1.667-approximation algorithm, which attempts to partition the tasks in a similar way as worst fit strategy and assigns all tasks with $S_{sc}$ if possible. Originally it is designed for per-core platforms, however, our preliminary experiments have shown that it is still a good candidate as the starting point in our context.

**Algorithm 1: HSAMC**

**Input:** A system model  
**Output:** A task partition and a speed assignment

1. Generate initial solution according to LA+LTF+FF, mark it as current solution  
2. Get the value of the solution  
3. While termination criterion not met do  
4. Use the guidance to generate a neighbor solution of the current solution  
5. If generated solution is schedulable then  
6. Get the value of generated solution  
7. Accept generated solution and mark it as current solution with probability $p$  
8. End If  
9. End While
Algorithm 1 (HSAMC) illustrates our main algorithm in detail. In each iteration a neighbor solution is randomly generated based on the penalty and reward values. In case of being schedulable, i.e. the solution can guarantee the system schedulability, which can be checked by means of utilization test, we compare the value of the new solution with the current one. The new solution is accepted with the probability $p$, which is dependent on the difference between the values of two solutions. If we denote the value of the current solution and the new solution as $e$ and $e'$, respectively, the acceptance probability is defined in equation (5), where $K$ is a constant.

$$p = \begin{cases} 
1 & \text{if } e > e' \\
\exp \left( \frac{e - e'}{e + K} \right) & \text{otherwise}
\end{cases}$$

(5)

### 5. TERMINATION CRITERION

There is one open question in the previous section, i.e. after how many iterations the algorithm should stop. In this section we propose a novel performance analysis mechanism to provide an efficient termination criterion and be able to accurately predict the quality of the solution. In order to make this analysis possible, we slightly modified Algorithm 1. Originally, it only stores the current solution and returns it as the output when the algorithm stops. Instead, we additionally remember the best solution ever found by the algorithm and return it as the output. Since the algorithm is able to accept worse solutions, the current solution is not necessarily the best solution ever found.

In order to better understand the finite time behavior of Algorithm 1, we first made several experimental tests through simulation. The virtual processor platform is composed of two Intel XScale processor core grouped into one cluster. The power model is taken from [Xu et al. 2004]. Figure 2 (left part) shows the simulation result by running Algorithm 1 on a randomly generated task set, where the x-axis shows the number of iterations and the y-axis is the value of the best solution found so far. There are two major observations: (i) the more iterations spent in the algorithm, the better solution can be obtained. (ii) The improvement of the solution value exponentially decreases as the algorithm proceeds. By the other experiments the same trend is observed as well. Therefore we reasonably conclude that the solution value is an exponential decreasing function of the number of iterations.

![Figure 2: Runtime behavior of Algorithm 1 based on one example](image)

Our core idea is then to use the exponential regression technique at runtime to simulate the behavior of the algorithm and predict the quality of the configuration. Obviously, the accuracy of the regression gets better while the algorithm advances and the number of observed data points increases. The regression model takes the form:

$$y_i = ae^{bx_i} + c, i = 1, \ldots, l$$

(6)

where the $y_i$ and $x_i$ are the response variable and input variable, respectively. The regression parameters $a$, $b$ and $c$ are to be determined by curve fitting to a series of observed data points $\{y_i, x_i\}_{i=1}^l$. Note that $y_i$ indicates the value of the best solution found by the algorithm until the $i$-th iteration and $\forall i: 1 \leq i \leq l, x_i = i$. 


Unfortunately, solving the exponential regression is not a trivial work and the traditional approaches, such as Gauss-Newton algorithm, are often very time consuming. However, there exists linearization technique transforming the exponential regression problem to the linear regression problem, provided that the parameter $c$ can be eliminated. Thus we reformulate the exponential regression model into the following function:

$$y'_i = y_{i-1} - y_i = ae^{bx_i} + c - (ae^{bx_i} + c) = (ae^{-b} - a)e^{bx_i}, i = 2, ..., l$$

(7)

Intuitively, $y'_i$ expresses the value improvement at the $i$-th iteration. Figure 2 (right part, original value) illustrates its runtime behavior based on the previous example. The behavior of the improvements actually reflects the gradient of the regression function and is an exponential decreasing function of the iteration number as well. A larger improvement represents a larger gradient value. On the contrary, smaller improvements indicate smaller gradient values and therefore the possible convergence to the optimum. By looking at equation (7), since the offset parameter $c$ disappears, we can linearize it to a linear regression model shown in equation (8). Hereby the nature logarithm is applied on both side of the equation.

$$\ln(y'_i) = \ln((ae^{-b} - a)e^{bx_i}) = \ln((ae^{-b} - a) + bx_i), i = 2, ..., l$$

(8)

If we let $Y_i = \ln(y'_i)$ and $X_i = x_i$, the final linear regression model is obtained as follows, where the slope and offset are $b$ and $\ln(ae^{-b} - a)$, respectively.

$$Y_i = bX_i + \ln(ae^{-b} - a), i = 2, ..., l$$

(9)

Now we can apply the well-known Least-Squares Estimation (LSE) technique by means of minimizing the sum of squared residuals to determine the regression parameters. The regression process will take place in each iteration of Algorithm 1. With other words, in each iteration we will estimate the parameters $a$ and $b$ of the regression model. The careful readers may notice that there are still two open problems: (i) what happens, if $y'_i$ is equal to zero, because the expression $\ln(0)$ is not allowed? (ii) how to find the termination criterion? In what follows, these questions will be answered.

In order to prevent the variable $y'_i$ from being zero, we propose to use smoothing technique to forecast such data. Furthermore, there is also a positive side effect that the regression technique works better on the smoothed data set than the original one, because the "noise" data are often smoothed out. For this, we apply the exponential smoothing approach which tries to weight the past data with regard to their time stamp, i.e. the older the data, the less the weight. The smoothing equation is shown in (10) where $y'_i$ and $\hat{y'_i}$ are the original observed data and the smoothed data, respectively.

$$\hat{y'_i} = \begin{cases} y_i & \text{if } y'_i > 0 \\ ay'_i + (1-a)y'_{i-1} & \text{otherwise} \end{cases}$$

(10)

$a$ is the smoothing constant and the less the value, the more the data set is smoothed. Figure 2 (right part, smoothed value) shows the smoothed data with $\alpha = 0.1$ and clearly all the zero data are replaced by non-zero data. Moreover, the smoothing constant $\alpha$ is a key factor to steer the convergence speed of $\hat{y'_i}$ towards zero. The larger the $\alpha$, the faster the curve approaches zero. Since the value improvement is an indicator of the gradient, the fast convergence consequently implies a fast termination of the algorithm. Obviously, fast termination has its price, because some local optima may be mistakenly recognized as the global optimum and reported as the final output of the algorithm. As a conclusion, the right choice of the $\alpha$ value has impact on the performance analysis result. Finally, we are ready to define the termination criterion.

By performing the function regression during the algorithm execution we can simulate the speed of the solution improvement. As a result, we have an estimation of the parameters $a$ and $b$ of the regression model $y = ae^{bx} + c$ at each iteration. In fact, $c$ is the value of the optimal solution. It is not hard to see that $ae^{bx} = y - c$ expresses the estimated absolute difference between the value of the solution found until $x$-th iteration and the optimal solution, which is in turn an estimator of the solution quality.

Finally, the termination criterion can be derived. Algorithm 1 terminates, if the estimated absolute difference is less than $\beta$ or the iteration number reaches a predefined threshold $l_{th}$. $\beta$ and $l_{th}$ are constants to
be decided by the user. Note that the threshold, on the one hand, ensures that the algorithm will definitely terminate and on the other hand, gives a possibility to define an upper bound of the affordable runtime.

6. EVALUATION

Our evaluation is performed by using a SystemC RTOS simulation framework [Zabel 2009]. We extended their library by adding the additional support for multi-core simulation with DPM/DVFS capabilities. Since we are interested in cluster-based multi-core processor platforms, different configurations are investigated, e.g. the platform with 4 cores grouped into 2 clusters or into 1 cluster. In this experiment setup we concentrate on symmetry cluster-based platforms, where all clusters contain the same number of cores. However, our algorithm is not constrained to symmetry configuration. Furthermore, for the sake of simplicity we assume that all cores possess the same power model. In the following test scenarios we refer to the platform with X cores grouped into Y clusters as coreXclusterY. For each core we adopt the power model of Intel XScale processor from [Xu et al. 2004], which supports 5 DVFS states. In each test scenario we applied EDF as real-time task schedule on each core. According to our preliminary experiments, the parameters of our algorithm are set as follows: $\lambda_1 = 10$, $\lambda_2 = 1$, $\lambda_3 = 1$, $K = 0.001$ and $\alpha = 0.01$. Hereby, we used millisecond as the unit for $t_{\text{balanced}}$. In the experiment we randomly generated 1000 task sets and the size of each task set is between 7 and 14. The period of each task is within [0.05 ms, 20 ms] and the utilization of each task set is within [0.4, 3.5]. We executed the task sets on the processor with 4 cores.

![Figure 3: Energy reduction comparison](image-url)

Figure 3 shows the simulation results on different platforms. Here the main focus is on the energy reduction efficiency by comparing our algorithm with the LA+LTF+FF algorithm (LALTFFF) in terms of task number. In the figure, x-axis shows the number of tasks, according to which the results are classified. The y-axis shows the processor energy consumption that is normalized with regard to the LALTFFF algorithm. Furthermore, HSAMC0.01, HSAMC0.05 and HSAMC0.01 indicate our algorithm with different $\beta$ values. For instance, HSAMC0.01 will terminate, if the estimated distance between the solution found by the algorithm and the optimal solution is less than 0.01. The figures clearly confirmed our expectation. The less the $\beta$ value, the more energy can be saved. Our algorithm outperforms the LALTFFF algorithm on all platforms and the energy reduction is up to 15%.

<table>
<thead>
<tr>
<th>Task number</th>
<th>core4cluster1(HSAMC0.01)</th>
<th>core4cluster2(HSAMC0.01)</th>
<th>core4cluster4(HSAMC0.01)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3988</td>
<td>5058</td>
<td>4043</td>
</tr>
<tr>
<td>8</td>
<td>5502</td>
<td>5579</td>
<td>5153</td>
</tr>
<tr>
<td>9</td>
<td>7641</td>
<td>7565</td>
<td>6996</td>
</tr>
<tr>
<td>10</td>
<td>8950</td>
<td>10452</td>
<td>8815</td>
</tr>
<tr>
<td>11</td>
<td>12007</td>
<td>12071</td>
<td>11399</td>
</tr>
<tr>
<td>12</td>
<td>12497</td>
<td>13203</td>
<td>13541</td>
</tr>
<tr>
<td>13</td>
<td>14898</td>
<td>13704</td>
<td>13900</td>
</tr>
<tr>
<td>14</td>
<td>16330</td>
<td>16003</td>
<td>14129</td>
</tr>
</tbody>
</table>
7. CONCLUSION & FUTURE WORK

With the continuous increasing of system complexity and the technology advance towards multi-core processor platforms, the problem of energy efficiency becomes more and more difficult. In this article we focused on scheduling hard real-time tasks on cluster-based multi-core processor platforms. The well-established techniques DPM and DVFS are applied together to minimize the processor power consumption. We proposed a simulated annealing based heuristic algorithm with an efficient termination criterion. Through the experiment results, our approach shows its great energy efficiency in comparison with the existing algorithms.

As future work, more evaluation needs to be performed to investigate the impact of different $\alpha$ values. In this paper it is assumed to be 0.01. Since $\alpha$ plays an important role to steer the convergence speed of the algorithm, a best compromise between the speed and the solution quality should be found. Furthermore, we plan to study the impact, if the DPM and DVFS state switching overhead are considered.

REFERENCES


F. Kong et al., 2011. Energy-Efficient scheduling of Real-Time tasks on Cluster-Based multicores, in DATE.


H. Zabel et al., 2009. Accurate RTOS modeling and analysis with SystemC. In Hardware-dependent Software. Springer.