Utilising Application Flexibility in Energy Aware Computing

Iain Bate

Department of Computer Science,
University of York,
York, YO10 5DD, United Kingdom
e-mail: iain.bate@cs.york.ac.uk

Abstract

There is a vast amount of existing work that investigates energy aware computing. However to exploit the best possible solution requires a system wide approach to design. To design and optimise a whole system at once is not scalable. A viable alternative is Component-Based Engineering approaches where individual parts of the system are designed whilst allowing for the interactions with the rest of the system. The contributions of this paper are a method by which flexibility in the design of applications can be exploited to give the most energy efficient requirements in a computationally efficient way and an exploration of how different computational models relate to the most energy efficient requirements. The results show that the most obvious choice of requirements is not always the best and the overheads due to specific computational models are significant. While carrying out this work, power / energy anomalies, similar in nature to the timing anomalies already identified for Worst-Case Execution Time and multiprocessor systems, have been uncovered.

1 Introduction

In recent years there has been a significant amount of research into energy aware computing. One particularly active area is that of energy efficient scheduling. One of the first significant works in this area was by Shin and Choi [24]. They proposed a simple dynamic model that exploits both variable voltage scheduling and sleep modes. During periods where the processor is idle the processor is put into sleep mode and when the processor is busy, but there is slack time, the processor speed is reduced by manipulating its voltage and / or frequency so that in the worst-case it finishes just before the slack time runs out. Slack time is defined here as the difference between the deadline of a task and its response time.

Most of the subsequent works, e.g. [8], on energy efficient scheduling builds on this simple model by attempting to anticipate future workload so the processor speed can be selected more efficiently. All this work is based on the fact there is no choice over the timing requirements of the system. That is, they are fixed apriori. However for many applications, e.g. real-time control systems, this is not the case [15, 23, 5]. The hypothesis in this paper is that by carefully selecting the timing requirements significant energy can be saved. The work builds on our previous work [5] where the design goals did not include energy efficiency. While carrying out this work, power / energy anomalies, similar in nature to the timing anomalies already identified for Worst-Case Execution Time [14], have been found.

A component-based engineering approach is taken where instead of trying to design the whole system at once, the design and optimisation problem is split into parts. However by clearly allowing for the interactions within the system, then better (or at least equivalent) behaviour can be achieved, in contrast to not exploiting application flexibility at just choosing the maximum period and deadline, in a scalable fashion. In this paper the interactions of concern are between the processor's scheduler and the application's control software. The properties of interest are the timing requirements and the fitness (or quality) of the control system which includes both the software and the plant.

Efficiency, effectiveness and flexibility are goals that are common to many systems including embedded real-time systems. However experience has shown that in practice current techniques frequently fall short of delivering on these goals. For example, some of our other previous work has looked at moving Fixed Priority Scheduling (FPS) into safety critical systems and in particular engine control systems [4]. However whilst using the developed approaches on projects has achieved significant measurable benefits, our industrial partners found that there were still improvements to be made as the flexibility afforded by FPS was not being fully exploited [10]. The reasons included lack of knowledge over what the 'real' timing requirements are, a lack of tools that support their derivation under realistic assumptions reflecting computer-based control systems, and a lack of design tools that support multiple disciplines (e.g. control and scheduling) [10].

To provide the flexibility needed there are a range of possible real-time control techniques. One form of solution are techniques based on adaptive control methods, e.g. Sename's work [22] that uses H∞ control technique. Sename uses resource information so that variabilities in when events occur, e.g. due to communications, can be compensated for in order to provide the optimal control characteristics. A similar approach is adopted by Brandt but this time resource information is used as part of the state variable technique [16]. This allows the rates for an individual control loop to be controlled based on the current resource availability and how well the loop is meetings its objectives. The second form of solution are those that use non-adaptive control algorithms, e.g. Proportional-Integral-Derivative (PID), and switch the rate of the algorithm's depending on the currently available resources. Examples here include Marti [15] and Seto's [23] work. The problem with all the approaches is they rely on on-line adaption (either from a control or scheduling perspective or both) which for many systems, especially those that are considered hard real-time, energy conscious or critical, is not currently acceptable due to reasons of predictability and higher overheads [6, 10].

An alternative approach is to derive a controller off-line that works for a range of non-ideal conditions which can then be scheduled with conventional scheduling approaches, e.g. static or fixed priority scheduling. Our previous work took this strategy. As part of this control scheduling framework was developed based on MATLAB [5]. The aim of this framework was to allow control systems for a particular plant to be designed and evaluated under realistic assumptions rather than artificial ones of linear time invariance. Amongst the outputs are the range of timing requirements (periods and deadlines) for which the control system requirements are met. In this paper we show how this flexibility over the timing requirements can be exploited to help improve the energy efficiency of the computing platform. Whilst the approach may seem simple and intuitive, it is effective but careful attention is needed to alleviate theoretical scalability concerns. To the best of our knowledge no existing published work has demonstrated a similar approach. In addition the work presents one of the first uses of the sustainability of timing analysis

One obvious approach is to choose the slowest periods for software in order to reduce utilisation and hence cpu energy usage. However the example in Figures 1 and 2 illustrates how the most obvious, and intuitive, choice is not always the correct one. In these figures the vertical axis illustrates the activity / task, the horizontal axis time, the rectangles the task being executed and the number within the rectangle the execution time. The example depicted is based on a simple two task system. Task A has a Best-Case Execution Time (BCET) and Worst-Case Execution Time (WCET) of 2. Its period can be chosen as either 8 or 12. Task B always has BCET and WCET of 2, and a period of 8. It is assumed task A has a higher priority than task B and that the task deadlines are equal to the period.

Figure 1 shows the case when task A has a period of 8. Figure 2 shows the case when task A has a period of 12. The energy consumption model is that when a task is executing 1 energy unit per time unit is used, it takes 3 time units to suspend and resume (from a timing perspective it is assumed there is no difference between treating suspension and resumption as one action or as separate actions), the energy usage during suspension is 0.1 energy unit per time unit, and the energy usage during the suspension and resumption period is the same as when a task is executing, i.e. 1 energy unit per time unit. The assumption that the energy usage is the same during execution, suspension and resumption is considered appropriate, at least for this example, based on a number of processor models including the ARM 8 [18]. The time frame the Figures show is 24 time units as this is the least common multiple of the task periods across both cases in the Figures. This corresponds to the time frame for which there is a repeating pattern of execution which is the minimum duration needed for worst-case analysis [13].

The Figures clearly show that in the case when task A has a period of 12 that the processor only spends 2 time units in suspended mode compared to 3 time units when its period is 8. This means that over the time frame shown (which is the duration for which the execution pattern repeats, i.e. the Least Common Multiple of the Periods) more energy is used when the task has a longer period, i.e. 22.2 energy units when the period is 12 versus 21.3 in the case of a period of 8.

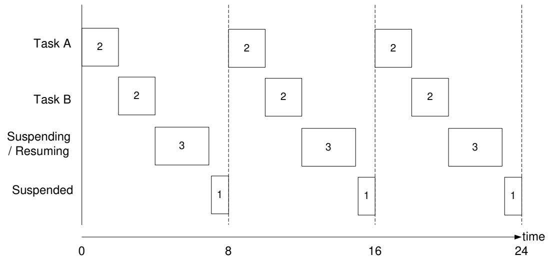

Figure 1. Period of Task A is 8

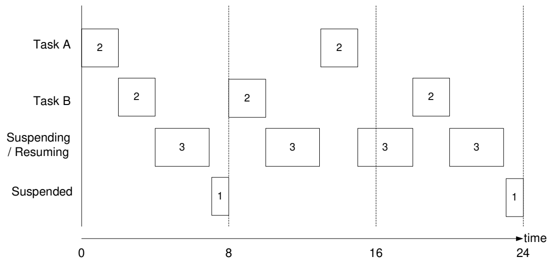

Figure 2. Period of Task A is 12

Now the research problem has been established and justified, the specific research questions to be answered are:

1. Given the rest of the system's timing requirements, how do we choose the most appropriate timing requirements for the control system?
2. Are there significant benefits over traditional design techniques for real-time control?
3. Can the timing requirements be chosen to minimise the number of context switches and processor speed changes?
4. Can the objectives be achieved with manageable computational complexity?

The structure of the paper is as follows. Section 2 presents related work. Next, a method for deriving the most energy efficient timing requirements is presented. The details of the evaluations, and results, can be obtained from [1]. Finally the conclusions, including a summary of the results, are presented in section 4.

2 Related Work

There are two main areas of real-time research that are relevant to this paper. These are real-time scheduling and energy aware scheduling. This section gives an overview of the relevant work in these areas.

Given the set of timing requirements there are a wide variety of scheduling paradigms that can be used. Two of the more popular approaches that offer flexibility and optimality (defined here as the ability to schedule tasks with up to 100% processor utilisation) under certain circumstances are Earliest Deadline First [25] and Fixed Priority Scheduling [11]. Neither of these approaches are ideal for energy aware systems.

As with real-time scheduling there are a wide variety of approaches to energy aware scheduling. These can be broken down into two broad categories - off-line and on-line strategies. The off-line strategies look at how the system can be synthesised to give the most energy efficient characteristics. Generally speaking these approaches look at how tasks should be allocated, e.g. [21], and then how the individual task attributes should be assigned. The attributes can be assigned to minimise execution time energy usage, e.g. [17, 20], to minimise processor specifics such as leakage, e.g. [12], or configure the processor to give overall benefit across the two previous areas [7, 27]. The on-line strategies are basically scheduling strategies that look to alter the voltage and frequency used by the processor or to maximise the time for which the processor can be suspended using current knowledge of execution state [24]. One of the original works in this area is the work of Weiser [26]. However later work, amongst others [19, 9], took the concepts further by trying to predict in advance the execution requirements such that scheduling is pro-active rather than reactive. This achieved considerable benefits from an energy usage perspective. Clearly there is a link between the off-line and on-line strategies as the off-line solutions will be generated in the context of a particular on-line schedule.

To the best of our knowledge no work has looked at using available flexibility in the timing requirements combined with off-line techniques to give an overall systems solution that optimises energy usage. The closest work to this subject is Seto's work [23] but this only used the flexibility to achieve schedulability.

3 Method for Deriving Energy Efficient Timing Requirements

In this section a method for choosing the most energy efficient timing requirements is presented. This work is done independent of any particular scheduling model or specific analysis. In this paper two objectives are considered when choosing the most appropriate timing requirements for energy aware real-time systems. Firstly meeting the mandatory constraints. That is, the overall task set must meet their hard real-time requirements in the worst case. The second is for the system to use the minimum energy in the average case. The rest of this section is split into two parts each dealing with one of these objectives.

3.1 Ensuring Constraints are Met

There is little point choosing a set of timing requirements to improve the energy properties of a system if the mandatory design constraints are not met. The constraints on the system from a timing and energy perspective are: *individual task requirements* - jitter and deadline, *transaction requirements* - precedence constraints and end-to-end deadline, and *energy requirements* - minimum and maximum energy usage.

From a timing perspective, timing checks are performed for each period and deadline within the range of requirements for which the control objectives are met. This is performed at appropriate steps, e.g. periods of 100, 110, 120, etc.. It is noted generally speaking it is invalid just to start with the shortest deadline and look for the minimum for which the timing requirements are met. The reason is the possible phasings of tasks meaning there may be no single critical instance, e.g. if there are offsets or blocking [3].

The individual task requirements are checked using conventional timing analysis, e.g. for fixed priority scheduling the analysis given in [2] could be used. This is performed assuming the processor is operating at its maximum frequency. This is a valid set of conditions as most energy-aware scheduling policies default to a conventional scheduling policy in the worst case [24, 26]. However the analysis performed needs to account for the scheduling overheads from the particular policy used. At this stage the work is independent of the computational model used.

The first stage of the timing analysis is to analyse that individual task requirements are met. Once the individual requirements are shown to be valid then the transaction requirements are checked using the analysis in [4] which is also contained in the longer version of this paper [1].

The overall approach, given in algorithm 1, effectively uses an exhaustive search. The algorithm can be simplified if a single task model is used to represent the control task as the step *"check transaction requirements"* can be omitted.

Algorithm 1 - *General algorithm for checking the requirements are met*

```
for each period step period_step
 for each deadline step deadline_step
  determine whether control constraints are met
  if control constraints met then
   assign priorities using an appropriate algorithm
   perform individual task timing analysis
   if individual tasks are all schedulable then
    check transaction requirements with analysis in [4]
  end if
 end loop
end loop
```

3.2 Finding the Best Requirements for Energy Efficiency

Once the mandatory timing requirements have been demonstrated as being met for certain periods and deadlines of the control software. The next stage is to determine which of these periods and deadlines result in the lowest average-case energy usage. This was achieved using a simulation environment that executes a randomly generated task set for a given duration (Dur) and monitors the energy used. There are four key parts to the simulation environment which is written in 'C'.:

1. *Problem generation* - Before the simulation starts, a random number of tasks are generated each with a BCET and WCET. For all the tasks, except the control scheduling tasks, a period is randomly chosen within a defined range. The deadline of the tasks is then set to be equal to the period.
2. *Task execution* - Each time a task is released it is given an execution time chosen randomly, according to a normal distribution, in the range [BCET, WCET]. It is worth highlighting a range [WCET, WCET] gives the lowest worst-case energy usage.
3. *Scheduler* - A scheduler, which adheres to the chosen computational model, then controls which tasks are in the delay and run queues, and which task is currently executing.
4. *Processor* - Based on the selected processor and energy model, current frequency and a duration of execution, the energy used is calculated.

Searching for the optimum set of timing requirements from an energy perspective can therefore be represented as follows:

Algorithm 2 - *General algorithm searching for requirements leading to minimum energy*

```
generate a random task set
if task set meets criteria, e.g. utilisation range then
 use Algorithm 1 to find valid timing requirements
 for each of the valid timing requirements
  simulate the task set for duration D
 end loop
 choose requirements resulting in the lowest energy usage
end if
```

The algorithm is currently expensive. In the next two sections it is shown how the computational complexity can be reduced when more specific details of the computational and scheduling model are available.

3.3 Computational Model and Timing Analysis

To demonstrate the impact of flexible timing requirements, the system is to be scheduled using the Low Power Fixed Priority Scheduling (LPFPS) approach of Shin in [24]. The reasons are:

1. the method (and benefits) of choosing the appropriate timing requirements are independent of the specific implementation used,
2. LPFPS forms the basis of most other methods, and
3. the relative simplicity of the computational model allows a clear analysis of where savings are being achieved. Otherwise it is will be less than clear as to whether they are provided by the method proposed in this paper or by the scheduling approach itself.

The essence of the LPFPS algorithm is that if sufficient slack time is known to exist before a task's deadline then the processor rate is reduced. If the run queue is empty then the processor is suspended until further work exists. At this point the processor comes out of suspended mode and execution is resumed. Clearly future work could show the benefits of the approach with other computational models. In addition the model of the ARM 8 processor given in [18] and the system example taken from [5] are used.

The processor model can be represented by the following: energy at a particular frequency (f) can be calculated using equation (1), the frequencies can be any integer value in the range 1 MHz to 100 MHz, energy usage when idle is 0.5 mW, and energy usage when suspended is 0.09 mW.

$$power = 0.000218 \cdot f^3 + 1.007241 \quad (1)$$

Finally, the overheads associated with the system are as follows: time to perform a context switch is 0.1 ms, time to change frequencies is 0.025 ms, and suspending the processor and then bringing it out of suspended mode takes 0.5 ms.

The timing analysis used is the standard response time analysis for fixed priority scheduling [2] assuming the WCET is that calculated for the highest processing rate, i.e. when the processor frequency is maximum. The analysis for calculating the Worst-Case Response Time (WCRT) of task i, R_i, when the system is preemptive is given in [4] and the longer version of this paper [1].

3.4 Reducing Complexity

To reduce the computational effort needed to apply the approach we can exploit the concept of sustainable timing analysis that has recently been proposed [3]. Sustainable scheduling is where the results of timing analysis are shown to be valid when certain parameters are relaxed. For the purposes of this paper, the key result in [3] is when using the timing analysis in [4] any reduction in C_i, B_i, or increase in T_i or D_i means the task set remains schedulable. The improved algorithm is given in Algorithm 3.

Using the sustainable argument, in Algorithm 3 where $TDAC$ stands for Task Deadlines Are Changing, means that the priorities used are sub-optimal, however when the task set is unschedulable then the next task set to be evaluated has new priorities assigned. This gives each task set the best possible chance of being schedulable. The priorities are assigned so that each task in a transaction has a deadline such that tasks featuring earlier in the transaction have a deadlines one clock cycle (Δ) less than the task that proceeds it. The algorithm for deadline assignment is given in Algorithm 4. Then, all tasks are assigned a priority according to the Deadline Monotonic Priority Ordering (DMPO) [2]. This means that tasks belonging to a transaction execute in the intended precedence order. This means it is more likely for a solution to be schedulable when the next step, timing analysis, is performed. Full details of the algorithm with examples and proofs are in [4].

Algorithm 3 - *Revised Version of Algorithm 1*

```
for each period step period_step
  for each deadline step deadline_step
    if control constraints met then
      assign priorities according to [4]
      perform individual task timing analysis
      use analysis in [4] to check transactions
    end if
    if timing requirements are met then
      exit inner loop
    end if
  end loop
  if timing requirements are met then
    exit outer loop
  end if
end loop
```

So what do these optimisations mean in practice? Essentially if we start searching through the periods and deadline from the minimum to the maximum, then as soon as a schedulable solution is found it can be assumed all tasks with a higher deadline and period are also schedulable. Clearly the computational saving depends on the specifics of the task set. In the best case only one set of timing requirements would need to be checked, if the minimum period and deadline resulted in the timing requirements being met. However in the worst-case, all timing requirements would be checked if only the maximum period and deadline lead to the timing requirements being met.

Algorithm 4 - *Algorithm for Deadline Assignment*

```
for each task in the transaction
    if the following task has an equivalent deadline then
        reduce the task's deadline by Δ
        assign the value of false to the TDAC flag
    end if
    while the transaction deadline is not met
        assign the value of true to the TDAC flag
        take the longest deadline and reduce it by Δ
        for each task in the transaction
            if the following task has an equivalent deadline then
                reduce the task's deadline by Δ
                assign the value of false to the TDAC flag
            end if
    end loop
end if
```

Further improvements could be attained if Algorithm 2 was applied in stages. Initially a larger step (for *period_step* and *deadline_step*) to identify a suitable range and then a smaller step to find a better result. This is illustrated in Algorithm 5. When calling function *algorithm2* the following parameters are passed: minimum deadline D_{min}, maximum deadline D_{max}, minimum period T_{min}, maximum period T_{max}, *deadline_step* and *period_step*. Two variables are returned: the best deadline D_b and the best period T_b. The algorithm could be applied in more stages with the range progressively being more focussed.

Algorithm 5 - *Efficient version of Algorithm 2*

(D_{best}, T_{best}) = algorithm2 (D_{min}, D_{max}, T_{min}, T_{max},
deadline_step, period_step)
(D_{best1}, T_{best1}) = algorithm2 (D_{best}-deadline_step,
D_{best}+deadline_step,
T_{best}-period_step,
T_{best}+period_step,
factor * deadline_step,
factor * period_step)

4 Conclusions

The work summarised in this paper, and fully contained in [1], has provided the following answers to the questions posed at the start of this paper.

1. Given the rest of the system's timing requirements, how do we choose the most appropriate timing requirements for the control system?
   The method presented uses ranges of timing requirements established in prior work. A key result is that if the most obvious choice of timing requirement is made, rather than applying the presented algorithm, then the most energy efficient requirements may not be found. Even in cases where the best requirements use the maximum period, then flexibility in the deadlines are usually beneficial. For example in one test this was the case in 31% of the tests performed. The energy savings ranged from tens of percentage

up to hundreds of percent in the most extreme cases. The importance of allowing for overheads is also shown in the evaluation results [1].

2. Are there significant benefits over traditional design techniques for real-time control?
Yes. In all the cases the results were better than the traditional 'rule of thumb' technique for both timing and energy.
3. Can the timing requirements be chosen to minimise the number of context switches and processor speed changes?
Again the simple answer is yes. Also the choice of computational model, e.g. non-preemptive versus preemptive scheduling, is significant.
4. Can the objectives be achieved with manageable computational complexity?
Again the answer is yes. Arguably the approach is an enumeration-based search algorithm with theoretically high complexity but by exploiting the sustainability of the analysis the problem has been shown to be manageable.

References

[1] http://www.cs.york.ac.uk/ ijb/rtcsa_2008_long.pdf.

[2] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–92, 1993.

[3] S. Baruah and A. Burns. Sustainable scheduling analysis. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 96–11, 2006.

[4] I. Bate and A. Burns. An integrated approach to scheduling in safety-critical embedded control systems. *Real-Time Systems Journal*, 25(1):5–37, Jul 2003.

[5] I. Bate, J. McDermid, and P. Nightingale. Establishing timing requirements for control loops in real-time systems. *Microprocessors and Microsystems*, 27(4):159–169, 2003.

[6] J. Buffington, V. Crum, B. Krogh, C. Plaisted, R. Prasanth, P. Bose, and T. Johnson. Validation and verification of intelligent and adaptive control systems. Technical Report AFRL-VA-WP-TP-2003-334, Air Force Research Laboratory, 2003.

[7] J. Chen, Y. Dong, H. Yi, and X. Yang. Compiler-directed energy-aware prefetching optimization for embedded applications. In *Proceedings of the International Conference in Embedded Systems and Software*, pages 230–243, 2005.

[8] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance setting for dynamic voltage scaling. In *Mobile Computing and Networking*, pages 260–271, 2001.

[9] D. M. H. Aydin, R. Melhem. *Handbook of Scheduling: Algorithms*, chapter Periodic Reward-Based Scheduling and Its Application to Power-Aware Real-Time Systems, pages 36.1 – 36.17. CRC Press, 2004.

[10] S. Hutchesson and N. Hayes. Technology transfer and certification issues in safety critical real-time systems. In *Digest of the IEE Colloquium on Real-Time Systems*, number 98/306, April 1998.

[11] D. Katcher, H. Arakawa, and J. Strosnider. Engineering and analysis of fixed priority schedulers. *IEEE Trans. Softw. Eng.*, 19(9):920–34, Sept. 1993.

[12] K. Khouri and N. Jha. Leakage power analysis and reduction during behavioral synthesis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(6):876–885, December 2002.

[13] J. Leung and M. Merril. A note on preemptive scheduling of periodic real-time tasks. *Information processing Letters*, 11(3):115–118, November 1980.

[14] T. Lundqvist and P. Stenstrom. Timing anomalies in dynamically scheduled microprocessors. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99)*, pages 12–21, December 1999.

[15] P. Marti, G. Fohler, K. Ramamritham, and J. Fuertes. Jitter compensation in real-time control systems. In *Proceedings of the 22nd Real-Time Systems Symposium*, pages 39–48, 2001.

[16] P. Marti, C. Lin, S. Brandt, M. Velasco, and J. Fuertes. Optimal state feedback based resource allocation for resource-constrained control tasks. In *Proceedings of the 25th IEEE Real-Time Systems Symposium*, pages 161–172, 2004.

[17] M. Moncusi, A. Arenas, and J. Labarta. *A modified dual-priority scheduling algorithm for hard real-time systems to improve energy savings*, pages 17–36. Kluwer Academic Publishers, 2003.

[18] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the IpARM microprocessor system. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '00)*, pages 96–101, 2000.

[19] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, pages 89–102, 2001.

[20] C. Rusu, R. Melhem, and D. Mossé. Multi-version scheduling in rechargeable energy-aware real-time systems. *Journal of Embedded Computing*, 1(2):271–283, 2005.

[21] M. Schmitz, B. Al-Hashimi, and P. Eles. Iterative schedule optimization for voltage scalable distributed embedded systems. *ACM Transactions on Embedded Computing Systems*, 3(1):182–217, 2004.

[22] O. Sename, D. Simon, and D. Robert. Feedback scheduling for real-time control of systems with communication delays. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 454–461, 2003.

[23] D. Seto, J. Lehoczky, and L. Sha. Task period selection and schedulability in real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 188–198, 1998.

[24] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Design Automation Conference*, pages 134–139, 1999.

[25] J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo. *Deadline Scheduling For Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publishers, 1998.

[26] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of 1st USENIX Symposium on Operating Systems Design and Implementation*, pages 13–23, 1994.

[27] C. Zhang, F. Vahid, and W. Najjar. A highly configurable cache for low energy embedded systems. *ACM Transactions on Embedded Computing Systems*, 4(2):363–387, 2005.