ISSN 0163-5964

Computer Architecture News

A Publication of the Association for Computing Machinery Special Interest Group on Computer Architecture

Vol. 29, No. 5 - December 2001

REGULAR CONTRIBUTIONS

SPECIAL ISSUE: PACT 2001 WORKSHOPS

DEPARTMENTS

Improving Energy Saving in Hard Real Time Systems via a Modified Dual Priority Scheduling [*]

M.Angels Moncusí, Alex Arenas
{amoncusi,aarenas}@etse.urv.es
Dpt d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili
Campus Sescelades, Av dels Països Catalans, 26
E-43007 Tarragona, Spain

Jesus Labarta
jesus@ac.upc.es
Dpt d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
Jordi Girona, 1-3. D6 Campus Nord
08034 Barcelona, Spain

Abstract

In this paper, we present a modification of the Dual Priority scheduling algorithm, for hard real-time systems, that takes advantage of its performance to efficiently improve energy saving. The approach exploit the priority scheme to lengthen the runtime of tasks reducing the speed of the processor and the voltage supply, thereby saving energy by spreading run cycles up to the maximal time constraints allowed. We show by simulation that our approach improves the energy saving obtained with a pre-emptive Fixed Priority scheduling.

1. Introduction

The design of portable digital systems has a major drawback in the constraint of low power consumption [1] from the operability and lifelong of the systems point of view. A lot of efforts have been made during the last decade to minimize this problem, but the high performance of modern micro-processors and micro-controllers jointly with the increasing functionality of them obtained via software still requires improvements in the power-efficiency context.

In the use of scheduling strategies to save energy there exist two main approaches to reduce power consumption of processors, these approaches are speed reduction of the processor and power-down. The first approach consist in to turn low the clock frequency along with the supply of voltage whenever the system does not require its maximum performance. The second approach simply turns off the power when there are not tasks to execute in prevision, apart from the minimal amount of energy required by the idle processor state (clock generation and timer circuits). Both approaches are well suited for energy saving but their applicability should be accurately designed to obtain reliable operability, especially in hard real-time systems [2,3].

Recently, Shin et al. [4] have proposed a power-efficient version of the Fixed Priority scheduling for hard real-time system that deal with the two approaches presented before. The main idea in their study is the use of a pre-emptive Fixed Priority scheduling (Rate Monotonic scheduling RMS [5] or Deadline Monotonic scheduling DMS) to organize the tasks according with the pre-emptive priority scheduler into a run queue that is used to exploit both, execution time variation and idle time intervals, to save energy by reducing speed and voltage or power down. The process ensures that all tasks meet their deadlines. However, the strategy of Shin et al. [4] can only reduce the speed of the processor when there is only one task in the run queue, or bring the processor to power-down mode when there is an idle interval, otherwise the processor works at the maximum speed.

In this paper we present an improvement of the strategy followed by Shin et al. [4] by using a modification of the Dual Priority scheduling, first proposed by Davis and Wellings [6]. We harness the ability of the Dual Priority to execute periodic tasks as late as possible to save energy.

The Dual Priority scheme was designed to execute aperiodic tasks without deadlines as soon as possible while preserving the deadline constraints of the periodic tasks. The algorithm is implemented as a three queue structure. The upper run queue, the aperiodic run queue and the lower run queue. Whenever a periodic task is ready to be executed enters the lower run queue, eventually this task can be pre-empted by an aperiodic task, and finally, if the task can not be delayed more because otherwise its deadline could be compromised, the task promotes to the upper run queue where its execution is prioritized.

This scenario is interesting even when no aperiodic tasks are present, as in our case of study, in this particular case the algorithm needs only two queues. The energy-reduction is obtained mainly by means of speed and voltage reduction and sometimes using power-down. Our approach consist in to run the tasks at the lowest speed that

[*] This work has been partially supported by the Spanish Ministry of Education (CICYT) under the TIC-511/98 contract.

makes possible that the active task and the rest of tasks meet their timing constraints, without imposing the constraint of Shin et al. [4] of only one task in the run queue to save energy, and power-down the processor when there is an idle interval.

This approach is especially interesting because the quadratic dependency of the power dissipation, in CMOS circuits, in the voltage supply [1]. The power dissipation satisfies approximately the formula

$$P \cong p_t C_L V_{dd}^2 f_{clk}$$

where p_t is the probability of switching in power transition, C_L is the loading capacitance, V_{dd} the voltage supply and f_{clk} the clock frequency. That means that it is always energetically favorable to perform slowly and at low voltage than quickly at high voltage.

The basic idea of the modified algorithm we present is to organize the run tasks in two levels of priorities. In the highest level there are the periodic tasks that their execution can no longer be delayed by tasks from the lower priority level otherwise they can miss their deadlines. The second level is occupied by those periodic tasks whose execution time can still be delayed without compromising the meeting of their deadlines. In its turn, each of the two levels is hierarchically organized according to any static priority assignment. To obtain an extra save in power another slight modification is introduced, the lower run queue is sorted by the promotional times instead of by fixed priorities. This approach is simple enough to be implemented in most of the kernels, in comparison with Shin et al. [4], we only use an extra run queue and a promotion time for each periodic task in the system. Then, the amount of extra complexity introduced by this new algorithm is minimal.

The paper is organized as follows, in the next section we describe the basics of the Dual Priority scheduling. Section 3 is devoted to the modification of the algorithm to reduce energy consumption. Finally, in section 4 we present the experimental results and the comparison with Power Low Fixed Priority scheduling, and in section 5 we draw the conclusions.

2. Dual Priority Scheduling

We assume that the framework of the hard real-time system we are going to deal with is made up of periodic tasks[1]. These tasks — numbered $1 \le i \le n$ — are specified by their periods, worst case execution times and deadlines (T_i, C_i and D_i respectively).

The system is organized as concurrent tasks ruled by a pre-emptive priority-based scheduler whose details are described below. The computation times for context switching and for the scheduler are assumed to be negligible, this enable us to perform the analysis straightforward without danger of loosing generality. The extent to which these assumptions are realistic is discussed in the analysis of the algorithm given in [6], and it turns out to be practical if the switch is subsumed to the worst case execution times of the different tasks.

The mechanics of the algorithm is the following: Let us assume that the tasks have some initial priorities assigned according to a fixed priority criterion in such a way that two different tasks have never the same priority. This initial priorities are altered by the scheduler according to the following scheme, first, two levels of priorities are organized, the highest level, or upper run queue (URQ) is for tasks that can no longer be delayed by less priority tasks otherwise they will miss their deadlines. The second level, or lower run queue (LRQ) is occupied by those periodic tasks whose execution time can still be delayed without compromising the meeting of their deadlines.

The scheduling algorithm is driven by the activation times of the tasks and the promotion instants to the URQ, whenever one of this time signals appears, in the following way, if:

1. The signal is the activation time (Ta_k) for some periodic tasks. In this case for all tasks with activation times less or equal to the current time t, the relative promotion time instant is pre-computed as $L_i=D_i - R_i$ (R_i corresponds to the worst case response time [8]), this value can be computed off line and provides the maximum time a task can be delayed so that it can still meet its deadline. Those tasks with $L_i=0$ are promoted to the URQ, and the rest remain in the LRQ. After that we compute the absolute promotion time instant for the k^{th} activation of task in the LRQ as $L_{ik}=Ta_k+L_i$, and a timer is activated to this value.
2. The signal is a promotion time instant. In this case, all tasks in the LRQ with $L_{ik} \le tc$ (current time) are moved to the URQ. Now, L_{ik} corresponds to $L_{ik}=D_{ik} - R_i$, where D_{ik} is the absolute deadline for the k^{th} activation of task i ($t0_i+ kT_i+ D_i$), where t0i is the first instant arrive time.

Finally, the next executing task is selected by picking the highest priority task from the highest non empty priority levels (i.e. URQ or LRQ, in this order).

This algorithm was conceived to schedule tasks with hard deadlines in a hard real-time environment containing periodic, sporadic and adaptive tasks coexisting. In this complex scenario there appears spare time due to tasks not consuming all its worst execution time. The on-line solution that Dual Priority scheduling presents is operative in the vast majority of kernels and computationally efficient [6,7]. Our goal is to take advantage of this performance from the energy saving point of view, the scheduling algorithm can be modified to extract the maximum time extension allowed by the real-time system, and this lengthen of time execution will be accompanied of a speed and voltage supply reduction, and finally energy reduction, as we explain in the next section.

[1] The results are not exclusive for periodic tasks. We have considered only periodic tasks as a matter of simplicity.

3. Power-Low Modified Dual Priority Scheduling

We have modified the Dual Priority Scheduling algorithm to help power saving in a hard real-time system. The original Dual Priority guarantees to meet the temporal constraints, then our modification only needs to care about when and how to reduce energy by slowing speed and voltage jointly (we are assuming a linear relation between speed and voltage supply decreasing). Figure 1 shows the pseudo code for the PLMDP (Power Low Modified Dual Priority Scheduling) that works as follows:

1. If both queues URQ and LRQ are empty, then the power-down mode is activated until the arrival of the next task ta_k.
2. If the queue URQ is empty but there are tasks in LRQ then the task i with the highest priority (ordered in terms of absolute promotion time L_{ik} for the k^{th} activation) is activated (line L6). Before execution we need to fix the ratio of processor speed according with the maximum spreading in time we are allowed. The speed ratio is calculated following the heuristics proposed by Shin et al. [4] that is built on the assumption that the delay is negligible. The safeness of the system under these conditions is proved on theorem 1 of the cited work. We calculate the time until the next signal for the current active task to promote Tp_i as the difference between ta_i plus the deadline, and R_i (worst case response time), see line L7. After that we determine the time we dispose before some other task will promote to the URQ. If there is a task that has not yet arrived, but with a promotion time shorter than the promotion time of the active task, then this task will preempt the active task. Before the arrival time of this task we have a time interval to execute the active task reducing speed (see L8-L9). In this case the speed of the processor should be the minimum possible speed. On the other hand, if the next promotion time will be the promotion time of the active task, it is discarded for the calculation because it do not impose a major restriction, then we should look for the following promotion time. If the following promotion time comes from a higher priority task T_j then we can execute the active task until this time tp_j reducing speed. To assign the corresponding speed in this case we calculate the amount of work that the task should execute (Γ_i). Γ_i will be the minimum time of the difference between the promotion time of T_j and the promotion time of T_i and the remaining of C_i. If the task T_j has less priority then we can execute the active task during the time interval defined by this promotion time plus the WCET of the active task, because in the calculation of the R_j we are taking into account the consumption of all the WCET (see L11-L16). Now Γ_i will be the minimum time of the difference between the promotion time of T_j plus the WCET of T_i and the promotion time of T_i and the remaining of C_i.
3. If the URQ has only one task to execute, then this is the active task and the processor speed is calculated (line L20) as the quotient between the minimum time of the next promotion time and the remaining C_i and the total time available to execute this tasks, that now is the minimum between the next promotion time and the current task deadline (see L21-L23)
4. If they are more than one task in the URQ, then we execute at maximum speed allowed by the processor (see L25-L26)

At practice it is obvious that only certain discrete values of the frequency of the clock, and then speed, are available, in this case the selection is always a frequency equal or larger than the calculated one to ensure time constraints. To see the difference between this algorithm and the LPFPS we have include a toy example in the annex.

```
L1  if empty(URQ) then
L2      if empty(LRQ) then
L3          Set timer to (next ta_j - wake up delay)
L4          Enter power-down mode
L5      else
L6          Active task_i = LRQ.head
L7          tp_i = ta_i + D_i - R_i
L8          if ta_k < tp_i and tp_k < tp_i then
L9              Speed = 1 / (ta_k - tc)
L10         else
L11             if tp_j < tp_m+C_i then      -- j∈hp(i), m∈lp(i)
L12                 Speed = min(tp_j - tp_i, remaining(C_i)) / (min(tp_j, td_i) - tc)
L13             else
L14                 Speed = min(tp_m + C_i - tp_i, remaining(C_i)) / (min(tp_m + C_i, td_i) - tc)
L15             endif
L16         endif
L17         Execute task_i
L18     endif
L19 else
L20     Active Task = URQ.head;
L21     if URQ.head.next = NIL then
L22         Speed ratio = min(tp_k, remainig(C_i)) / (min(tp_k, td_i) - tc)
L23         Execute task_i
L24     else
L25         Speed = 1.0
L26         Execute task_j
L27     endif
L28 endif
```

Figure 1. Pseudo code PLMDP

4. Experimental results

To check the capabilities of the PLMDP approach, we have simulated several examples and compared the total energy results (per hyper-period) obtained in front of the Low Power Fixed Priority Scheduling (LPFPS) proposed by Shin et al. [4]. We collected some of the experiments used by Shin et al: the Avionics task set [9], an Inertial Navigation System (INS) [10], and a Computerized Control Machine (CCM) [11].

The two first sets represent critical mission applications and the last one is an automatic control for specific machinery. The results are pictured in Figures 2 to 4, respectively. The average factor of improvement of our algorithm in front of LPFPS is 1,13 times for the avionics data set, 1,03 times for the INS task set and 2.07 times for the CCM data set.

If we pay attention to the specific behavior of the individual benchmarks we observe that the relationship between periods and WCET's are responsible of the main differences between both approaches, i.e., for example in the avionics and INS task sets, there is a sub-set of tasks that have a very large period compared with the respective WCET, this fact implies that for a long time there is a unique task in the system, and then our algorithm behaves very similar to LPFPS.

On the other hand, we observe also that both algorithms behave similar when the WCET is exhausted, in three of the four case studies, that is so because in general, in this case, there is not any extra time to consume, and then no more energy could be saved using only a scheduling strategy. However, in the CCM task set there appears a particular configuration of tasks that have very large ratio between the periods and WCET, but still it is possible to take advantage of many short times with significant reduction of speed even when the tasks are using the whole WCET, while LPFPS, in this same situation, has usually a few large time intervals where the speed can be reduced (see Figure 4). In the opposite situation, i.e. when the tasks consume less than a 10% of the WCET, it is difficult to perceive the differences. Finally, when the utilization of the WCET is around its half the differences between both performances are more relevant.

We have also evaluated the performance of both schemes in front of a simple task set represented by Table 1 (see Annex). In this case the average improvement is 1,71 times the energy efficiency obtained by LPFPS, Figure 5.

All the experiments represent the results of the normalized average energy obtained, varying the consumed worst execution time from 10% to 100%. We run the simulation over one hyper-period (that is, the minimum common multiple of the task's period).

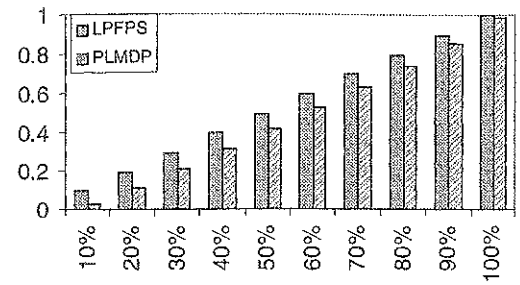


Figure 2: Avionics task set

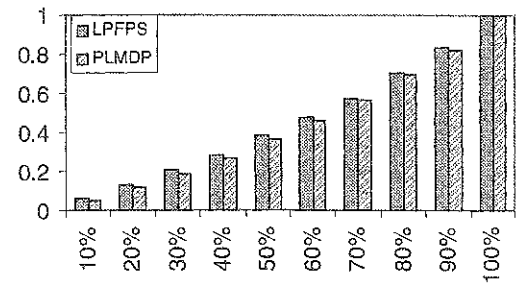


Figure 3: INS task set

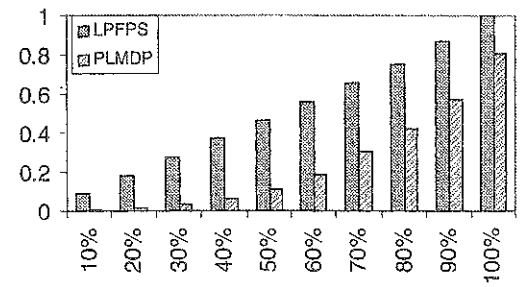


Figure 4: CCM task set

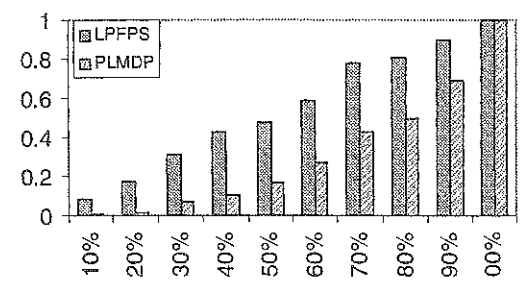


Figure 5: Shin et al.[4] task set

5. Conclusions

We have presented a modification of the Dual Priority Scheduling to improve the Fixed Priority Scheduling power aware while maintaining the low complexity of the algorithmic. This approach has been shown to over-perform the mentioned LPFPS power saving by an average factor than range from 1,03 up to 2.07 depending on the application. The algorithm does not increase the complexity of the LPFPS and can be implemented in most of the kernels.

6. Annex

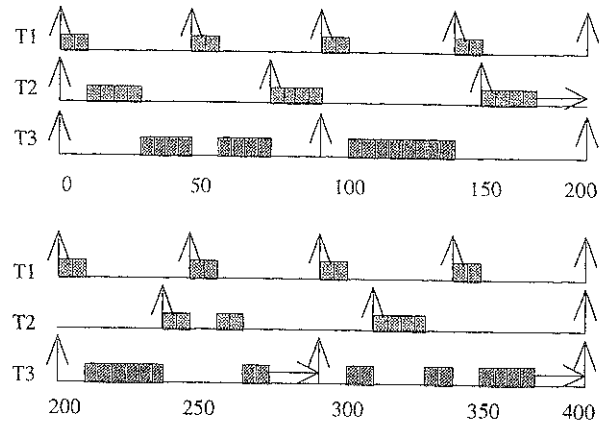
Here we present an example to better appreciate the functioning of our algorithm (PLMDP) in comparison with the LPFPS. The benchmark used is the same presented by Shin et al. [4], Table 1. In the Graphs (a) and (b) we represent the execution of both algorithms, when the tasks consume the 100% of its WCET, and in the Graphs (c) and (d) we represent the execution of the algorithms, when all tasks consume the 50 % of its WCET. In all these Graphs, the vertical up narrows represent the arrival of the task to the system, the vertical down arrows represent the promotion time of the task, and finally the horizontal arrow stands for the time we lengthen the time execution of the task. Each box represent five time units (although our minimal calculation unit corresponds to 1 time unit), and each line corresponds to task T1, T2 and T3 respectively. In Graphs (c) and (d), the shaded circles represent idle time in the system.

Table 1: Benchmark task set used by Shin et al.[4]

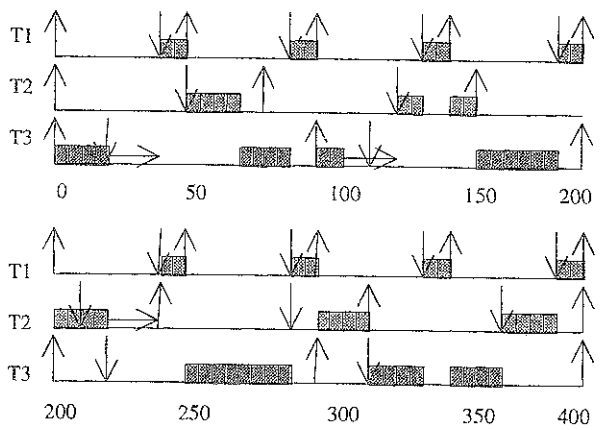
Task	T	D	WCET	R	D-R	P
T1	50	50	10	10	40	1
T2	80	80	20	30	50	2
T3	100	100	40	80	20	3

In Graph (a) we have represented the behavior of the LPFPS. The LPFPS algorithm is driven by the Fixed Priority Scheduling (see Shin et al. [4] for an extended explanation). Let focus our attention in the Graph (b), that represents the behavior of our algorithm, it is as follows: At t=0, all three tasks arrive to the system and then they are placed at the LRQ, the first task to promote according with our scheme will be T3, then it is activated. Its promotion time arrives at t=20, and we can execute this task until t=40 (promotion time instant of T1) without any problem. Executing T3 as late as possible implies that the execution time of T3 should start at its promotion time (t=20) and it would be preempted at t=40, that means that we have 40 time units to execute 20 time units, we can then reduce the speed and the power supply. The algorithm has nothing different from this behavior until time 200. At time t=200 we have again all task in the LRQ, but now the first promotion time corresponds to T2. After that T3 promotes, but because it has less priority than T2, it can not take the CPU, then the execution of T2 continues until the minimum value of T1 promotion time and the WCET of T2 plus the promotion time of T3. In this way the algorithm uses the exceeding time to work at slow processor speed and low voltage.

In the Graphs (c) and (d) we represent the execution of both algorithms, in a different situation, when all tasks consume the 50% of its WCET.



Graph a: Execution time in LPFPS when all tasks use 100% WCET.

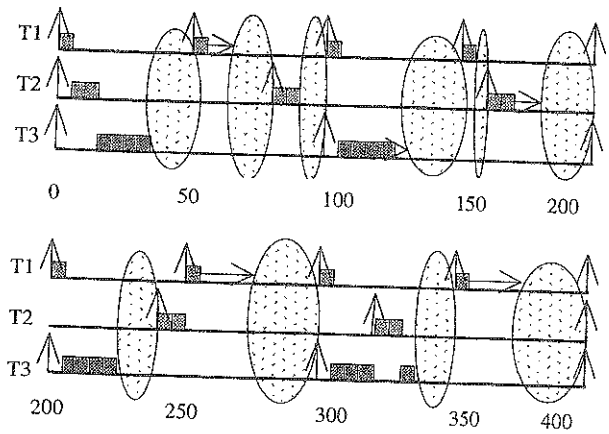


Graph b: Execution time in PLMDP when all tasks use 100% WCET

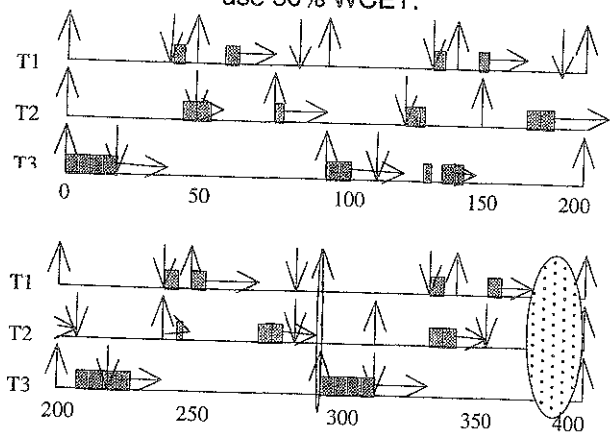
In Graph (c) we have represented the behavior of the LPFPS (Shin et al. [4]), and in Graph (d) we represent the behavior of our algorithm. Although the main behavior of the algorithm is identical to the behavior described before, this new situation provokes more idle time of the processor that should be used in energy saving. At time 0, all three tasks arrive to the system, but now, task T3 finishes at time 40 because it only executes the 50 % of its WCET. Task T1 is now the active task in the URQ, it executes 5 units time at maximum speed because its WCET is 10 and its deadline is 50, but this task finishes at time 45 because it now executes only 5 units. After that, there is only task T2 in the LRQ that promotes at time 50 to the URQ. At time 50 it will arrive task T1, it enters the LRQ and promotes to the URQ at time 90. Then we can reduce the clock speed expecting to finish at its deadline at time 80, the minimum between the deadline of the active task (t=80) and the promotion time of a higher priority task (t=90). As task T2 executes only a half of its WCET, it finishes at time 63 and the processor continues with task T1 that now can reduce speed again, expecting to finish by its deadline that is in this case the

minimum between the promotion time of task T2 and the deadline task T1. After that, task T1 executes at low speed and finishes by time 80. At time 80 it arrives task T2 to the LRQ and it stands alone until time 100 when task T1 and task T3 arrive. The promotion time of task T3 occurs at time 120 while the promotion time of task T2 occurs at time 130, so that in the LRQ task T3 will have the highest priority because the promotion time is earlier. For that reason task T2 should execute at the lowest possible speed only until task T3 arrives and pre-empt task T2.

Graph c: Execution time in LPFPS when all tasks use 50% WCET.



Graph d: Execution time in PLMDP when all tasks use 50% WCET.



To summarize the comparison, in Graphs (a) and (b), when tasks consume all its WCET, nor LPFPS nor PLMDP have idle interval times. In this particular case there is not big differences between the performance of both algorithms. The only difference is that the energy saving occurs at different times but globally the total amount is the same. On the other hand when tasks consumes its 50 % of WCET, Graphs (c) and (d), PLMDP has only 20 units of idle time, while LPFPS has 167 free units time, this effect translates in our algorithm in an energy saving of around 300 % with respect to LPFPS.

In general, real time systems behave in a mixed situation with a few tasks consuming 100% of its WCET and the rest consuming fractions of its WCET, then our algorithm shows to improve the energy saving obtained with a fixed priority scheduling algorithm.

7. References

[1] A.P. Chandrakasan, S. Sheng and R. W. Brodersen, "Low-power CMOS digital design", *IEEE Journal of Solid-State circuits*, vol. 27, pp. 473-484, April 1992.

[2] D. Mosse, H. Aydin, B. Childers and R. Melhem, "Compiler-assisted power-aware scheduling for real-time applications" Workshop on Compilers and Operating systems for Low Power COLP 2000, Philadelphia, Pennsylvania, October 2000.

[3] H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics" 13th Euromicro Conference on Real-Time Systems, Delft, Netherlands, June 2001.

[4] Y. Shin and K. Choi, "Power conscious Fixed Priority scheduling in hard real-time systems" DAC 99, New Orleans, Louisiana, ACM 1-58113-7/99/06, 1999.

[5] C. L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", JAMC 20, pp. 46-61, 1973.

[6] R. Davis and A. Wellings, "Dual Priority scheduling", Proceeding IEEE Real Time Sistems Symposium, pp. 100-109, 1995.

[7] A. Burns and A.J. Wellings, "Dual Priority Assignment: A practical method for increasing processor utilization", Proceedings of 5th Euromicro Workshop on Real-Time Systems, Oulu, IEEE Computer soc. Press, pp. 48-55, 1993.

[8] M. Joseph and P. Pandya, "Finding response times in a real-time system", British Computer Society Computer Journal, 29(5): 390-395, Cambridge University Press, 1986.

[9] C. Locke, D. Vogel and T. Mesler, "Building a predictable avionics platform in Ada: a case study", Proceedings IEEE Real-Time Systems symposium, December 1991.

[10] A. Burns, K. Tindell and A. Wellings, "Effective analysis for engineering real-time fixed priority schedulers", IEEE Transactions on Software Engineering, 21, pp. 475-480, May 1995.

[11] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi and H. Shin, "Visual assessment of a real-time system design: a case study on a CNC controller", Proceedings IEEE Real-Time Systems symposium, December 1996.