Energy-Efficient Improved Priority Ceiling Protocol for Real Time System

Ajitesh Kumar
Department of Information Technology
Hindustan Institute Of Technology and Mannagement, Agra

S.K.Gupta
Department Of Computer Science
Bundelkhand Intitute Of Engineering and Technology, Jhansi

Mona Kumari
Department of Information Technology
Anand Engineering College
Agra

ABSTRACT

Minimization of energy consumption in the battery operated system is become a major issue. Most of the real-time systems consist of a battery operated microprocessor system with a limited battery life. So, energy consumption is becoming a critical issue in the design of embedded systems because of the popularity of portable devices such as mobile devices and personal digital assistants.

In this paper we propose a approach to handle task synchronization for Real Time Systems with energy efficiency consideration. Our proposed approach is a variation of the well known priority ceiling protocol (PCP) which is to enforce mutually exclusive access to shared resources. We are using the concept of speed locking in proposed approach so that we can save the enrgy consumption.

Keywords: Real time Systems, energy consumption, task scheduling, critical section, DVS, context switching

1. INTRODUCTION

There are a lot of real-time task scheduling algorithm with energy constraint have been proposed to supports dynamic voltage scaling. However, most of the work only considers independent real-time task on DVS platforms. Some work is also done in synchronizing of dependent real-time tasks for minimizing energy consumptions [2,7]. With consideration of energy consumption, we propose an approach for scheduling and synchronizing of soft real time tasks. In this paper, our aim to maximize the battery life time of the system and these systems usually have one processor, memory units and several non preemptive co-processors [3]. All non preemptive processing elements and memory units are shared resource among tasks.

Real-time tasks are scheduled based on priority-driven schemes and modeled with timing parameters such as periods, deadlines and minimum separation. Tasks whose deadlines can never be violated are called hard real-time tasks and those still contribute value to the system after having violated their deadlines are soft real-time tasks. A set of real-time tasks is called independent task set when the execution of each task is independent, whereas tasks will access a shared resource is called dependent task set. Various real-time task scheduling algorithm with energy constraint have been proposed to supports dynamic voltage scaling [14]. However, most of the work only considers independent real-time task on DVS platforms. For independent task sets, optimal algorithms have been proposed, such as rate monotonic (RM) [3] scheduling and earliest deadline first (EDF) [12] scheduling for fixed and dynamic priority tasks, respectively. For dependent task sets, shared resources are assumed to be accessed in a mutually exclusive manner by real-time tasks. When scheduling dependent real-time tasks, the main focus is on how to manage the priority inversion problem so that urgent tasks could be serviced with proper timing/quality- of-service guarantee. Excellent real-time concurrency control methodologies have been proposed to synchronize dependent real-time tasks, such as priority inheritance protocol (PIP) [10], priority ceiling protocol (PCP) [18], and stack resource policy (SRP) [20].

In these system, a higher priority task may blocked by lower priority task due to resource sharing. Without blocking time management, the task might miss its deadline. Here we consider the problem of reducing the number of context switches in real time system with priority driven preemptive scheduling. Context switches occur whenever a task relinquishes its control over the CPU to the next task in the system. By reducing the number of context switching we can save the energy.

To minimize the energy consumption and meet their deadline on such systems, we propose an Energy Efficient Improved Priority Ceiling Protocol. In this proposed protocol we reduce the number of context switches which are caused by task synchronization. In our method we are disallowing higher priority task on available slack to preempt the lower priority task those are in critical section.

2. PRELIMINARY

Here, we show the system model followed by motivational example for proposed approach.

2.1. Task and System model

This system model deals with energy minimization of random arrival of aperiodic tasks and it also operate on different frequency level.

We assume a task set of n aperiodic real time tasks $\tau = \tau_1, \tau_2, ...\tau_n$, and a set of resources $R = r_1, r_2 .....r_m$ those are non-preempt-able in the system. The tasks are independent and fully preemptive in nature. We are considered the task to be released at critical instance time with worst case execution time.

Let P_i denote the priority, T_i denote the time period, E_i denote the worst case execution time and D_i denote the relative deadline. A task may access one or more non preemptable shared resources. Before a task τ_i accesses a

non-preempt-able shared resource r_j it must first lock the resource r_j and unlock the resource when access is completed. Here we assume in this system the processor (DVS) will support different operating speeds those are define as a set of $S = s_{slp}, s_1, s_2 .....s_n$ where $s_1 \leq s_2 \leq s_3 .... \leq s_n$. Let lowest speed is denoted by $s_{min} = s_1$ and highest speed is denoted by $s_{max} = s_n$, A processor can be in one of the three possible states namely active, idle, and sleep. In the active state the processor can run any of the speed levels between s_1 to s_n, while in idle state s_1 and sleep state s_{slp}

2.2. Motivations

Energy-efficient task scheduling for real-time system has become a active research topic in the recent few years. For the minimization of energy consumption many excellent algorithms for scheduling has been proposed. However, little work has been proposed for synchronizing real-time task on DVS based processor. Now we shall use the following example to illustrate the motivation of our proposed work.

Example 1: Suppose that an aperiodic task P arrives at time 1 with execution time 4 units and need a resource r_1 during its execution of 2-4 time units, the deadline of task P is 38; task Q arrives at time 3 and need a resource r_1 during its execution of 3^{rd} time units, the deadline of task Q is 36; task R arrives at time 6 and need a resource r_2 during its execution of 2-3 time units, the deadline of task R is 32; task S arrives at time 7 and need a resource r_1 during its execution of 3^{rd} time units, the deadline of task P is 24. Each task has same execution time of 4 units. We also suppose that T has highest assign priority, R has next highest priority, Q has 3^{rd} highest and P has lowest priority among all task.

The priority ceiling of the resources r_1 and r_2 are 1 and 2, respectively.

Schedule the given aperiodic task set along with the Priority Ceiling Protocol (PCP)[12],

1. Figure 1 shows at a time 1 task P arrives and start execution, the ceiling priority of the system at time 1 is Ω and a time 2 task P locks the resource r_1, after r_1 is allocated, the ceiling of the system is raised to 1, the priority ceiling of r_1 and at a time 3 a higher priority task Q arrives and preempts the task P and at a time 5 task Q attempts to access resource r_1, but already locked by P.Q becomes block and P resumes its execution and inherits the priority of Q.
2. At a time 6 higher priority task R arrives and preempts P again and at a time 7 a higher priority task S arrives and preempt R; Higher priority S need a resource r_1 at a time 9 but resource is already locked by task P so that S becomes block and task P inherits the priority of task S and resume its execution. During all these time, the ceiling of the system remains at 1.
3. At a time 11 task S release r_1, the ceiling of the system drops to Ω and at a time 12 when task R request r_2 , its priority 2 is higher than the ceiling of the system. Hence, its request is granted according to the allocation rule of PCP [12].

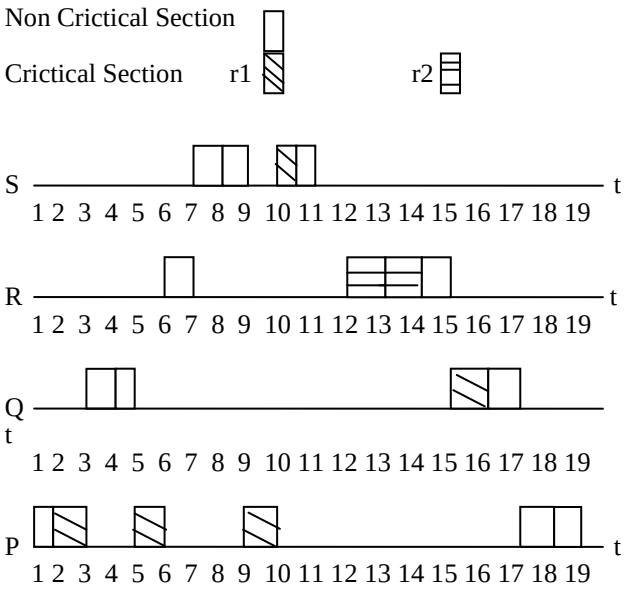

Figure 1. Schedule with Priority Ceiling Protocol

The above example shows that a lower priority task that lock the resource r_1 at starting of its execution for that reason all higher priority task are blocked for some time so that a number of context switches are increases. In this example, there are 9 context switches, and the holding time of resource by any task is long. Allowing the lower priority task to execute its critical section by a higher priority task on available slack, we can save the context switch and also reduce the holding time of resource by any task.

The proposed protocol is a variation of priority ceiling protocol (PCP) which is to enforce mutually exclusive access to shared resources. We are also proposed the optimal frequency level of tasks when they are using shared resources, so that we reduce the energy consumption and minimized the number of task those miss their deadline.

3. PROPOSED IMPROVED PRIORITY CEILING PROTOCOL

Assumptions:

- The priority assigned to all task are fixed.(Before)
- Before the execution of any task the requirement of resources are known.

3.1. Scheduling Rule

At a time t, when tasks are released, the priority $\pi(t)$ of every task is equal to its assigned priority accept under the priority inheritance rule.

Every ready job P is scheduled preemptively and in priority driven manner under the condition:
If some lower priority task is in the critical section when the higher priority task arrives.

- S1: if higher priority task will be blocked by lower priority task in future:
  Then we calculate the maximum amount of of slack available for higher priority and permits the lower priority task to execute its critical section, in that available slack.
- S2: if higher priority task will never blocked by lower priority task in future:
  Then we calculate the maximum amount of slack available for higher priority and also check the availability of resource used by higher priority task in the duration of their execution of higher priority task and then permit the lower priority task to execute its critical section, in that available slack.

3.2. Resource Allocation Rule

When any task τ request a resource R at a time t:

- If resource is not free then the request is denied and task τ is blocked
- If resource is free:
  - If τ's priority $\pi(t)$ is higher than current ceiling priority Π_t,resource is allocated to task τ
  - If τ's priority $\pi(t)$ is not higher than current ceiling priority Π_t,resource is is allocated to task τ only if task τ is the task holding the resource(s) whose priority ceiling equals Π_t;otherwise,τ's is denied and τ becomes blocked.

3.3. Priority Inheritance Rule

- A task uses its assigned priority, unless its blocks higher priority task or run on critical section, if a lower priority task blocks higher priority task , the lower priority task will inherits the priority of higher priority task.When the lower priority task executes its critical section then it resumes the priority it had at the point of obtaining the lock on the resource.
- When a higher priority task permits the lower priority task to executes its critical section in the duration of available slack , the lower priority task inherits the priority of higher priority task and executes its critical section and after that it resumes its own priority.

The performance of modifying rules for real time task set as compare to existing priority ceiling protocol can be seen in example 1. Here, same example 1 is scheduled by proposed improved priority ceiling protocol.

1. figure 2 shows at a time 1 task P arrives and start execution, the ceiling priority of the system at time 1 is

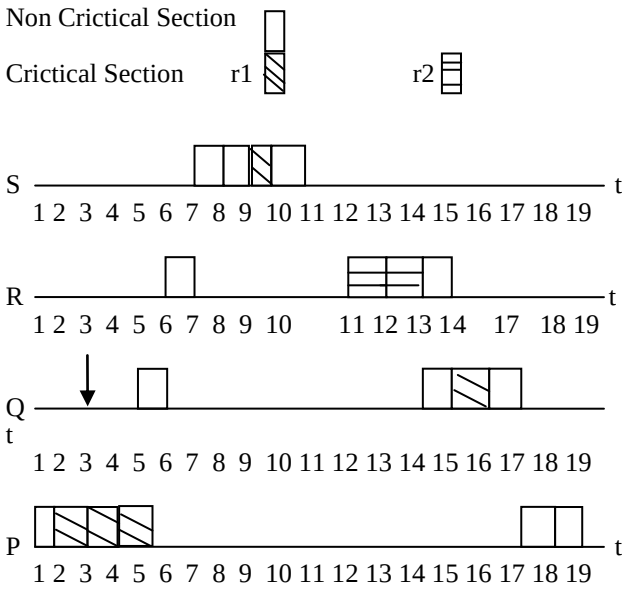

Figure 2. Schedule With Improved Priority Ceiling Protocol

Ω and at a time 2 task P locks the resource r_1, after r_1 is allocated, the ceiling of the system is raised to 1, the priority ceiling of r_1 and at a time 3 a higher priority task Q arrives, according to Scheduling Rule S1: now check task Q will be blocked by task P in future.The condition is true, so now we calculate the highest available slack for task Q. The highest available slack for Q is equal to $Q_{lst} - Q_{est} =$
$d_Q - e_Q - Q_{est} - e_R - e_S$ =(36-4-3-4-4)=21 time units
and lower priority task P requires 3 time units to execute
its critical section that is lower than available slack so we permits the lower priority task to executes its critical section, with the inherited priority. During all this time , the ceiling of the system remains at 1.
2. At a time 5 when task P finish its critical section, then task P preempts by task Q, and task P resumes its own priority. And the ceiling of the system drops to Ω At a time
6 the higher priority task will arrive and preempts the lower priority task then we check lower priority task is in critical section or not. the condition is false, so task R will executed.
3. At a time 7 the higher priority task S will arrive and preempt the task R then we check lower priority task is in critical section or not. the condition is false, so task S will executed with highest priority. At a time 9 task S locks the resource r_1, after r_1 is allocated, the ceiling of the system is raised to 1, and a time 10 again system ceiling goes down Ω.
4. At a time 11 task R request r_2 , its priority 2 is higher than the ceiling of the system Hence, its request is granted according to the allocation rule of IPCP.

So in our proposed protocol only 6 context switches that results to consume less energy as compare to existing approach. Besides reduction in energy consumption it also

Improve the responsiveness of the system by guaranteed, to not miss their deadline.

4. SPEED LOCKING ON IMPROVED PRIORITY CEILING PROTOCOL

The speed locking concept is same as used by YA-Shu Chen [1]. The processor speed is switched only when a task is scheduled for the first execution, or when all of the executing tasks have just finished their execution at this time point.
The new processor speed is set based on the following four conditions:

1. If no task is scheduled for the first execution, and all of the executing tasks have just finished their execution at this time point, then the speed is set as the one for the idle mode.
2. If a task is scheduled for the first execution, and all of the executing tasks have just finished their execution at this time point, then the processor speed is set as the base speed of the task which is calculated by the offline speed assignment.
3. If a task is scheduled for the first execution, and some of the executing tasks have not finished their execution at this time point, then the processor speed is set as the maximum of its base speed which is calculated by the offline speed assignment and the current processor speed.
4. Otherwise, the processor speed remains.

A speed locking concepts could be better illustrated by applying the concepts on same example 1.
We suppose that the base speed of task P,Q,R and S are set as s_{max}, $3/4s_{max}$, $1/2s_{max}$ and $1/4s_{max}$ where s_{max} = 100. In the example a lower priority task is assigned a higher base speed in execution due to a greater task preemption cost from higher priority tasks.

1. As shown in figure 3, when task P arrives at a time 1, the processor speed is set as s_{max}, at a time 2 task P successfully locks r_1 and processor speed remains the same. At time 5 task P successfully executed with processor speed s_{max}, and at the same time, Q will be executed with processor speed $3/4s_{max}$ (condition 2) and stretch the execution time of Q is 4 to 5.3. And a time 6, task R will arrives for first execution and set the processor speed $3/4s_{max}$ (condition 3).
2. At time 7, task S will arrives, and preempt the task R and execute with processor speed $3/4s_{max}$ (condition 3)
and stretch the execution time 4 to
5.3.
3. At time 12.3, current processor speed is $3/4s_{max}$, more than the base speed of task R. So that task R will executes with $3/4s_{max}$ (condition 3).
As a result, no task will violate its timing constraint under speed locking on IPCP.

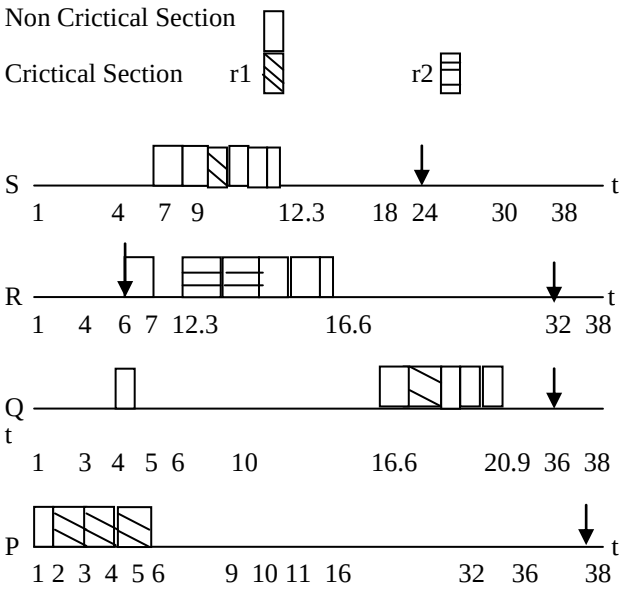

Figure 3. Improved Priority Ceiling Protocol With Speed Locking

Table I
COMPARISON OF TASK ENERGY CONSUMP TION

Task	Task Energy Consumption in existing approach[24]	Task Energy Consumption in proposed approach	% Energy Saving
P	$e_p*(s_{max})^3=4*100^3$	$e_p*(s_{max})^3=4*100^3$	0
Q	$e_q*(s_{max})^3=4*100^3$	$e_q*(3/4s_{max})^3=5.3*75^3$	44.10
R	$e_r*(s_{max})^3=4*100^3$	$e_r*(3/4s_{max})^3=5.3*75^3$	44.10
S	$e_s*(s_{max})^3=4*100^3$	$e_s*(3/4s_{max})^3=5.3*75^3$	44.10

Now we schedule the same set of task under PCP with speed locking.
When task P arrives at a time 1, the processor speed is set as s_{max}, at a time 2 task P successfully locks r_1 and processor speed remains the same. At a time 3 when task Q preempts P, the processor speed will remain same(condition
3). Note that the processor speed stays at s_{max} until all task finish its execution.
Table 1 evaluate the performance of Improved Priority Ceiling Protocol, it shows the percentage energy saving.
Now we assume that, 1 joule energy will be consume in one process switching and 0.5 joule energy will be consume in speed switching overhead.
So Total energy consumption = (Task energy consumption + Process Switching Overhead + Speed Switching Overhead)

Now the total energy consumption in the case of speed locking in PCP is $(16 * 100^3 + 9 + 0) = 16000009$ joule.
And in case of proposed approach the total energy consumption is $(4 * 100^3 + 5.3 * 75^3 + 5.3 * 75^3 + 5.3 * 75^3 + 6 + 0) = 10707818.5$ joule.
So, from above calculation we have consumed less energy in comparison with existing approach.

5. OBSERVATION AND DISCUSSION

In this section, we present the result of Improved Priority Ceiling Protocol.

5.1. No. Of Context Switches

Since one context switch can be associated with each task arrival, the number of context switches will equal the number of tasks when there is no preemption. Each preemption leads to one extra context switch when the preempted task resumes whereas each blocking causes two more context switches. The reduction in context switches due to synchronization using Improved PCP is shown in Fig.4.

5.2. Effect of Discrete Speed

Experiments were also conducted to evaluate the energy consumption of Speed locking-PCP with different number of processor speeds. Note that when the number of available processor speeds is finite, the available speeds are derived by picking up real numbers between 0.1 to 1 with an equal consecutive distance. For example, 0.1, 0.55, and 1 were the available speeds of a processor. Fig. 5 and 6 shows the average energy consumption of speed locking-IPCP and PCP with different tasks in a task set and the total CPU utilizations at the maximum processor speed. The overhead in speed switching was 0.5 ms at the maximum processor speed in all cases. The horizontal axis represents the number of speed levels of a processor, and the vertical axis represents the average energy consumption.
The required number of available processor speeds, in general, increased, when the total CPU utilization increased, or when the total number of tasks increased. Compare the figure 5 and 6 the energy consumption of task execution, when there were 5 available processor speeds, was 10 % more than that when there is more number of speed levels.It means when we have more no of speed levels then we consume less energy.

5.3. Effect of Speed Switching Overhead

The results show that the overhead in speed switching had little impacts on the number of required processor speeds.

Fig. 7 and 8 shows the average energy consumption of Speed Locking-IPCP and PCP, when there are ten tasks with the total CPU utilization at the maximum speed, respectively, and the available speeds could be any number between 0.1 and 1.0. The horizontal axis represents the overhead in speed switching at the maximum processor speeds (ranged from

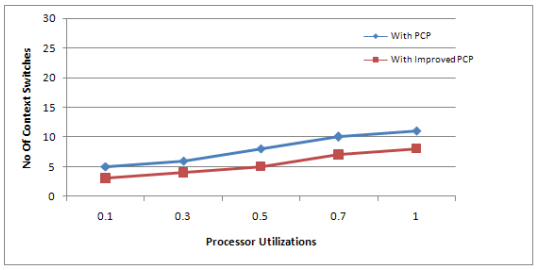

Figure 4. Reduction in Context Switches

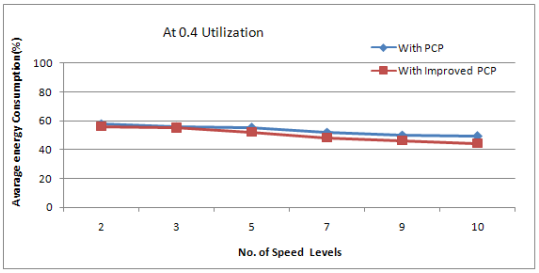

Figure 5. Average Energy Consumption Vs No. of Speed Levels with U= 0.4 and 10 task

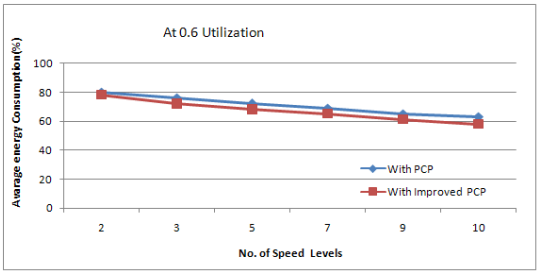

Figure 6. Average Energy Consumption Vs No. of Speed Levels with U= 0.6 and 20 task

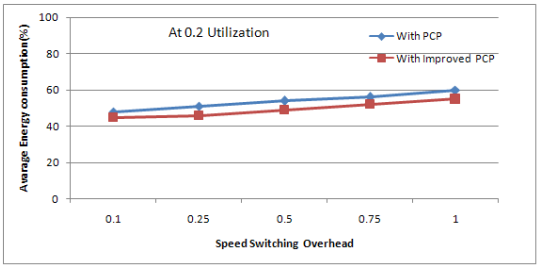

Figure 7. Average Energy Consumption Vs Speed Switching Overheads with U= 0.2 and 10 Task

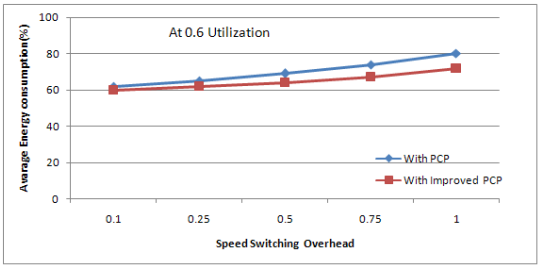

Figure 8. Average Energy Consumption Vs Speed Switching Overheads with U= 0.6 and 20 Task

0 to 1 ms), and the vertical axis represents the normalized energy consumption. As shown in Fig. 10, when the overhead in speed switching was 1.0 ms at the maximum processor speed, the normalized energy consumption of Speed Locking-PCP could reach 0.2, which was 20% of the average energy consumption of PCP with Speed Locking. It means when speed switching overhead increases, the energy consumption is also increases.

6. CONCLUSION AND FUTURE WORK

In this paper we present a task synchronization techniques which minimizes the system energy consumption for real time system. We must emphasize on the major goal of this research work is to minimize the energy consumption by reducing the number of context switching, and at the same time to minimize the number of deadline misses. We present a solution for reducing the number of context switches in task synchronization.

The objective is to minimize the energy consumption of a given task set, provided that the schedulability of tasks is guaranteed. We use the concept of speed locking[24] in task execution and extend the Proposed Priority Ceiling Protocol by locking the processor speed in a restricted manner so that the cost in speed switching is better managed. One particular characteristic of the concept is that no speed switching occurs when a lock request to a semaphore is blocked, or when a task resumes its execution from a blocked request. To reduce the cost in priority inversion, tasks with lower priorities are assigned higher processor speed. The proposed protocol is called Energy-Efficient Improved Priority Ceiling Protocol (EEIPCP) which is an extension of well known priority ceiling protocol (PCP).
The performance evaluations shows that our proposed protocol can significantly reduce the number of context switching normally range from 10 to 20%, and minimize the energy consumption and outperform previous work. We have performed simulations to compare the energy saving under the original protocol with proposed one. The percentage of energy savings normally range in 15 and 20 in either system environments.

REFERENCES

[1] A. Arya Paul, B. Anju.S.Pillai., "Reducing the Number of Context Switches in Real Time Systems." 2011 IEEE.

[2] Abhilash Thekkilakattil,Abhilash Thekkilakattil,Radu Dobrin and Sasikumar Punnekkat., "Preemption Control using Fre- quency Scaling in Fixed Priority Scheduling ", IEEE/IFIP INTERNATIONAL CONFERENCE ON EMBEDDED AND UBIQUITOUS COMPUTING 2010.

[3] C. L. Liu and J. W. Layland. "Scheduling algorithms for multiprogramming in a hard real-time environment." pages 46-61. JACM, Vol. 20, No. 1, January 1973.

[4] C. Rusu, R. Melhem, and D. Moss, "Maximizing the system value while satisfying time and energy constraints," in Proc. Real-Time Syst. Symp., 2002, p. 246.

[5] D. Zhu, R. Melhem, and B. Childers. "Scheduling with dy- namic voltage/speed adjust- ment using slack reclamation in multi-processor real-time systems." pages 84-94. In Proceed- ings of IEEE 22th Real-Time System Symposium, 2001.

[6] F. Yao, A. Demers, and S. Shankar. "A scheduling model for reduced CPU energy." pages 374-382. IEEE. Proceedings of the 36th Annual Symposium on Foundations of Computer Science, 1995.

[7] Giuseppe Lipari, Gerardo Lamastra, and Luca Abeni., "Task Synchronization in Reservation-Based Real-Time Systems", IEEE TRANSACTIONS ON COMPUTERS,VOL. 53, NO. 12,DECEMBER 2004

[8] H. Aydin, R. Melhem, D. Moss, and P. Meja-Alvarez, "Deter- mining optimal processor speeds for periodic real-time tasks with different power characteristics," in Proc. Euromicro Conf. Real-Time Syst,, 2001, p. 225.

[9] Intel Corporation. Enhanced Intel Speed Step Technology for the Intel Pentium M Processor, March 2004.

[10] L. Sha, R. Rajkumar, and J. Lehoczky. "Priority Inheritance Protocols: An approach to real-time synchronization." Page 1175V1185. IEEE Transactions on Computers, 1990.

[11] Linwei Niu., "System-Level Energy-Efficient Scheduling for Hard Real-Time Embedded Systems." 2011EDAA

[12] Marko Bertogna and Sanjoy Baruah., "Limited Preemption EDF Scheduling of Sporadic Task Systems", IEEE TRANS- ACTIONS ON INDUSTRIAL INFORMATICS, VOL. 6, NO. 4, NOVEMBER 2010.

[13] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. "Dynamic speed scaling to manage energy and temperature." pages 520-529. In Proceedings of the 2004 Symposium on Foundations of Computer science, 2004.

[14] Padmanabhan Pillai and Kang G. Shin "Real-Time Dynamic Voltage Scaling for Low-Power and Embedded Operating Systems" U.S. Airforce Office of Scientific Research under Grant AFOSR F49620-01-1-0120.

[15] Ramesh Mishra, Namrata Rastogi, Dakai Zhu, Daniel Mosse, and Rami Melhem. "Energy aware scheduling for distributed real-time systems." page 21. In International Parallel and Distributed Processing Symposium, 2003.

[16] R. Jejuikar and R. Gutpa, "Energy aware task scheduling with task synchronization for embedded real-time system," IEEE Trans. Comput. Aided Design of Integr. Circuits Syst., vol. 25, no. 6, pp. 10241037, 2006.

[17] Ron Cytron, Morgan Deters and Christopher Gill . "Rate- Monotonic Analysis in the C++ Type System' 'Sponsored by DARPA under contract F33615-00-C-1697. Oct 2002.

[18] S. Davari and S.K. Dhall. " On a real-time task allocation problem." 19 Annual Hawaii May 2012.

[19] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. "Algorithms for power savings." pages 37-46. In Proceedings of the Four- teenth Annual ACM-SIAM Symposium on Discrete Algo- rithms, 2003. International Conference on System Sciences, January 1985.

[20] T. P. Baker. "A stack-based resource allocation policy for real time processes." IEEE 11th Real-Time Systems Symposium, December 4-7, 1990.

[21] T.-W. Kuo and A. K. Mok. "Load adjustment in adaptive real- time systems." IEEE 12th Real-Time Systems Symposium, December 1991.

[22] W. Kim, J. Kim, and S. Mi, "Preemption-aware dynamic voltage scaling in hard real-time systems," in Proc. Int. Symp. Low Power Electronics and Design,

2004, pp. 393398.

[23] Yann-Hang Lee, Krishna P. Reddy, and C. M. Krishna. "Scheduling techniques for reducing leakage power in hard real-time systems." pages 105-112. In 15th Euromicro Confer- ence on Real-Time Systems (ECRTS), 2003.

[24] Ya-Shu Chen, Chuan-Yue Yang, and Tei-Wei Kuo, "Energy- Efficient Task Synchronization for Real-Time Systems," IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL.6, NO. 3, AUGUST 2010.

[25] Y. Zhang, X. Hu, and D.Z. Chen. Task scheduling and voltage selection for energy minimization. pages 183-188. Annual ACM IEEE Design Automation Conference, 2002.