ENERGY EFFICIENT SCHEDULING FOR REAL-TIME EMBEDDED SYSTEMS WITH PRECEDENCE AND RESOURCE CONSTRAINTS

Santhi Baskaran[1] and P. Thambidurai[2]

[1]Department of Information Technology, Pondicherry Engineering College, Puducherry, India

santhibaskaran@pec.edu

[2]Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry, India

ptdurai@pec.edu

ABSTRACT

Energy consumption is a critical design issue in real-time systems, especially in battery- operated systems. Maintaining high performance, while extending the battery life between charges is an interesting challenge for system designers. Dynamic Voltage Scaling and Dynamic Frequency Scaling allow us to adjust supply voltage and processor frequency to adapt to the workload demand for better energy management. Usually, higher processor voltage and frequency leads to higher system throughput while energy reduction can be obtained using lower voltage and frequency. Many real-time scheduling algorithms have been developed recently to reduce energy consumption in the portable devices that use voltage scalable processors. For a real-time application, comprising a set of real-time tasks with precedence and resource constraints executing on a distributed system, we propose a dynamic energy efficient scheduling algorithm with weighted First Come First Served (WFCFS) scheme. This also considers the run-time behaviour of tasks, to further explore the idle periods of processors for energy saving. Our algorithm is compared with the existing Modified Feedback Control Scheduling (MFCS), First Come First Served (FCFS), and Weighted scheduling (WS) algorithms that uses Service-Rate-Proportionate (SRP) Slack Distribution Technique. Our proposed algorithm achieves about 5 to 6 percent more energy savings and increased reliability over the existing ones.

KEYWORDS

Distributed Real-time System, DVS, Dynamic Slack, Energy efficient Scheduling

1. INTRODUCTION

Many embedded command and control systems used in manufacturing, chemical processing, avionics, telemedicine, and sensor networks are mission-critical. These systems usually comprise of applications that must accomplish certain functionalities in real-time [1]. Dynamic voltage scaling (DVS) is an effective technique to reduce CPU energy. DVS takes advantage of the quadratic relationship between supply voltage and energy consumption, which can result in significant energy savings. By reducing processor clock frequency and supply voltage, it is

DOI : 10.5121/ijcsea.2012.2216

possible to reduce the energy consumption at the cost of performance of processors [2]. Battery powered portable systems have been widely used in many applications. As the quantity and the functional complexity of battery powered portable devices continue to rise, energy-efficient design of such devices has become increasingly important. Also these systems have to concurrently perform a multitude of complex tasks under stringent time constraints. Thus minimizing power consumption and extending battery lifespan while guaranteeing the timing constraints has become a critical aspect in designing such systems. The focus is on task scheduling algorithms to meet timing constraint while minimizing system energy consumption.
In order to make energy efficient, in the scheduling, the execution time of the tasks can be extended up to the worst case delay for each task set. In real-time system designs, Slack Management is increasingly applied to reduce energy consumption and optimize the system with respect to its performance and time overheads. In energy efficient scheduling, the set of tasks will have certain deadline before which they should finish their execution and hence there is always a time gap between the actual execution time and the deadline. It is called slack time [3]. Therefore to minimize the energy consumed and to satisfy the deadline of the tasks, the processors run at variable speeds there by reducing the energy consumed by them. This is simulated with various task sets on different set of processors using proposed and existing task scheduling algorithms.

In this paper homogeneous distributed embedded systems executing a set of dependent tasks of a real-time application, which are normally represented by a task graph, is considered. The algorithm aims to reduce the energy consumption without missing any deadlines for a hard real-time task and with minimum deadline misses for soft real-time tasks [4]. Resources should be allocated efficiently among tasks and also care should be taken to see that no deadlock occurs [5]. Therefore it is necessary to introduce resource management mechanisms that can adapt to dynamic changes in resource availability and requirement.

This paper is organized in the following way. The related work is addressed in Section II. The various models used in this work are described in Section III. Proposed algorithm is described in Section IV. Section V discusses the simulation and analysis of results. Finally Section VI concludes this paper with future work.

2. RELATED WORK

The two most commonly used techniques that can be used for energy minimization in distributed embedded systems are Dynamic Voltage Scaling (DVS) [6] and Dynamic Power Management (DPM) [7]. The application of these system-level energy management techniques can be exploited to the maximum if we can take advantage of almost all of the idle time and slack time in between processor busy times. Hence, the major challenge is to design an efficient scheduling algorithm which can exploit the slack time and idle time of processors in the distributed embedded systems to the maximum. Various energy-efficient slack management schemes have been proposed for these real-time distributed systems. The static scheduling algorithm uses critical path analysis and distributes the slack during the initial schedule [8]. The dynamic scheduling algorithm [9] provides best effort service to soft aperiodic tasks and reduces power consumption by varying voltage and frequencies. Resource adaptation techniques for energy management in distributed real-time systems need to be coordinated to meet global energy and real-time requirements. This issue is addressed based on feedback-based techniques [10], [11] to allocate the overall slack in the entire system.

One of the existing works for distributed real-time system is based on feedback control scheduling. This MFCS algorithm [11] can provide real-time performance guarantees efficiently, even in open environments. This algorithm framework has three components; *Monitors* that track the CPU utilization of each processor; *Resource reclaimer* that computes difference between task's actual execution time and worst case execution time for local and global slack adjustment; and *Feedback scheduler* that performs resource reclaimer recommended schedule adaptations dynamically. A processor in the distributed system receives subtasks of different tasks, arrived to the global Feedback scheduler. Within each processor the tasks are scheduled by the Basic scheduler using Earliest Deadline First algorithm (EDF), since EDF has been proven to be an optimal uniprocessor scheduling algorithm [12]. EDF algorithm allows dynamic rescheduling and pre-emption in the queues and processing nodes. The Monitor module periodically checks the processor utilization and sends message to the global feedback scheduler, which on arrival of a new task decides to assign the task to a processor where the utilization criterion for the local EDF scheduler is still satisfied.

Another existing work for distributed embedded real-time system uses Service Rate Proportionate (SRP) slack distribution technique [13] for energy efficiency. Both the Dynamic and the Rate-Based scheduling schemes have been examined with this technique. It introduces SRP a dynamic slack management technique to reduce power consumption. The SRP technique improves performance/overhead by 29 percent compared to contemporary techniques. This system does not consider resource constraints among the dependent tasks. In this work, a scheduling algorithm known as Weighted First Come First Served (WFCFS) with an efficient dynamic slack management technique is proposed, such that energy efficiency of the processors increases still maintaining the precedence, resource and timing constraints.

3. MODELS

In this section, we briefly discuss the system, application, precedence, resource and slack management models that we have used in our work.

3.1. System and Application Model

A distributed embedded system with p homogeneous processors each with its private memory is considered for scheduling the given real-time application. The system requires the complete details of the task processing times (i.e.) the execution time and deadline before program execution. Each processing element (PE) in the system is voltage scalable and, can support continuous voltage and speed changes. We assume that the energy consumption, when the processor is idle, is ignored.

The real-time applications can be modelled by a task graph G = (V,E), where V is the set of vertices each of which represents one computation (task), and E is the set of directed edges that represent the data dependencies between vertices. For each directed edge (v_i, v_j), there is a significant inter-processor communication (IPC) cost when the data from vertex v_i in one PE is transmitted to vertex v_j in another PE. The data communication cost in the same processor can be ignored. Each real-time application has an end-to-end deadline D, by which it has to complete its execution and produce the result.

The frequency selection is influenced by making a task more or less urgent by shifting its deadline back and forth. The range within which the local deadline at node i can be varied is bounded by [S_i−,S_i+]. The values for S can be derived from the local task parameters. If $wcet_i$ represents the worst case execution times of the local task at node i, then

$$S_i^- = wcet_i$$
$$S_i^+ = D - \sum_{j=i+1}^{n} wcet_j$$

3.2. Precedence Model

Tasks of a real-time application considered in this paper have precedence constraints. For example, a task T_i can become eligible for execution only after a task T_j has completed because T_i may require T_j's results. For implementing precedence constraints DAG is used. Precedence constraints between tasks can also be modelled as resource dependencies. The precedence constraint that T_j precedes T_i is equivalent to the situation where T_i requires a logical resource (before it can start its execution) that is available only after T_j has completed its execution.

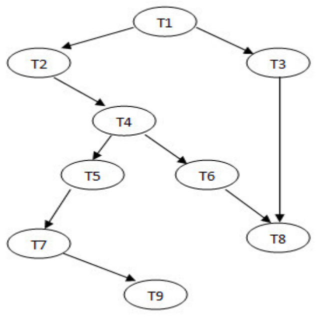


Figure 1. Precedence Constraints of a Task Set

Thus, if T_j has completed its execution before T_i arrives, then this logical resource is immediately available for T_i and T_i becomes eligible to execute upon arrival. Furthermore, if T_j has not completed its execution when T_i arrives, then the logical resource is not available and, therefore, T_i is conceptually blocked upon arrival. Later, when T_j completes its execution, the logical resource becomes available and T_i is unblocked.

3.3. Resource Model

To model non-CPU resources and resource requests, we make the following assumptions:

1) Resources are reusable and can be shared, but have mutual exclusion constraints. Thus, only one task can be using a resource at any given time. This applies to physical resources, such as disks and network segments, as well as logical resources, such as critical code sections that are guarded by semaphores.
2) Only a single instance of a resource is present in the system. This requires that a task explicitly specify which resource it wants to access. This is exactly the same resource model as assumed in protocols such as the Priority Inheritance Protocol and Priority Ceiling Protocol [14].
3) A task can only request a single instance of a resource. If multiple resources are needed for a task to make progress, it must acquire all the resources through a set of consecutive resource requests. In general, the requested time intervals of holding resources may be overlapped.

We assume that a task can explicitly release resources before the end of its execution. Thus, it is necessary for a task that is requesting a resource to specify the time to hold the requested resource. We refer to this time as HoldTime [15]. The scheduler uses the HoldTime information at run time to make scheduling decisions.

3.4. Slack Management Model

In real-time system designs, slack management is increasingly applied to reduce power consumption and optimize the system with respect to its performance and time overheads. This slack management technique exploits the idle time and slack time of the system through DVS in order to achieve the highest possible energy consumption. In energy efficient scheduling, the set of tasks will have certain deadline before which they should finish their execution and hence there is always a time gap between the actual execution time and the deadline. It is called slack time.

Conventional real-time systems are usually over estimated to schedule and provide resources using the wcet. In average case, real-time tasks rarely execute up to their worst case execution time (wcet). In many applications, actual case execution time (acet) is often a small fraction of their wcet. However, such slack times are only known at runtime through resource reclaimers. This slack is passed to schedulers to determine whether the next job should utilize the slack time or not.

The main challenge is to obtain and distribute the available slack in order to achieve the highest possible energy savings with minimum overhead. But most of these do not address dynamic task inputs. Only a few that attempt to handle dynamic task inputs assume no resource constraints among tasks. But in reality, few tasks need exclusive accesses to a resource. In exclusive mode no two real-time tasks are allowed to share a common resource. If a resource is accessed by a real-time task, it is not left free until the task's execution is completed. Other tasks in need of the same resource must wait until the resource gets freed. Our proposed algorithm handles this issue through the Restriction Vector (RV) [16] algorithm.

Our slack management algorithm decides when and at which voltage should each task be executed in order to reduce the system's energy consumption while meeting the timing and other constraints. Our solution includes two phases: First we use static power management schemes based on wcet to statically assign a time slot to each task. Then we apply dynamic scheduling algorithm to further reduce energy consumption by exploiting the slack arising from the run-time execution time variation. Here a small amount of slack time called unit slack is added to all the tasks and finally we find the subset of tasks that can be allocated this slack time so that total energy consumption is minimized while the deadline constraint is also met.

4. PROPOSED ENERGY EFFICIENT SCHEDULING ALGORITHM

We proposed a new algorithm called Weighted FCFS which increases the efficiency of the system by reducing the total energy consumption. In addition to this deadline hit ratio is a major factor to be considered in a soft real-time system for better QoS. Our proposed algorithm increases the deadline hit ratio and thereby improving the quality of the system.

4.1. Weighted FCFS Scheduling Algorithm

In the weighted FCFS method weight is assigned to each task based on the number of tasks on which it depends. First the tasks are inserted into the queue in the non decreasing order of the arrival time. Then the tasks in the queue are sorted according to the increasing order of their weight. The pseudo code for weighted FCFS is given in Figure 2.

4.2. Restriction Vector (RV) Algorithm

Resource reclaiming [17] refers to the problem of utilizing resources left unused by a task when it executes less than its wcet, because of data-dependent loops and conditional statements in the task code or architectural features of the system, such as cache hits and branch predictions, or both. Resource reclaiming is used to adapt dynamically to these unpredictable situations so as to improve the system's resource utilization and thereby improve its schedulability.

The resource reclaiming algorithm used is a restriction vector (RV) based algorithm proposed in [15] for tasks having resource and precedence constraints. Two data structures namely restriction vector (RV) and completion bit matrix (CBM) are used in the RV algorithm. Each task Ti has an associated n component vector, RV_i [1 . . . n], where n is the number of processors. RV_i[j] for a task Ti contains the last task in $T_{<i}(j)$ that must be completed before the execution of Ti begins, where $T_{<i}(j)$ denotes the set of tasks assigned to processor P_j that are scheduled in feasible schedule (pre-run schedule) to finish before Ti starts. CBM is an m X n Boolean matrix indicating whether a task has completed execution, where m is the number of tasks in the feasible schedule.

```
Weighted_FCFS ()
{
          call Assign_weight ()
          call InsertFCFS (queue,t)
          call sort (queue)
}


Assign_weight () //assign to each task
{
          for each task $t_i$ in queue // i = 1 to n ( number of current tasks
                                                      in the queue)
          {
             $t_i$.weight = Number of tasks on which $t_i$ is dependent
             // assign weight based on the precedence constraint of the task
          }
}



InsertFCFS (queue, t) // inserts the task t into the FCFS queue
{
          for each task $t_i$ in queue // i = 1 to n ( number of current tasks
                                                      in the queue)
          {
                        if(t.arrival_time < $t_i$.arrival_time)
                         {
                              queue_insert (queue, t, $t_i$)  // insert t before $t_i$
                          }
         }
          // if the task is not inserted
          if(not inserted) // add to the end of the queue
          {
                    queue_append (queue, t);
          }
}


Sort (queue)
{
          for each task $t_i$ in queue // i = 1 to n ( number of current tasks
                                                     in the queue)
          {
                    if ($t_i$.weight > $t_{(i+1)}$.weight)
                    {
                         swap ($t_i$, $t_{(i+1)}$) // interchange the tasks in the queue
                    }
          }
}
```

Figure 2. Pseudo Code for WFCFS Algorithm

4.3. Dynamic Slack Management Algorithm

The pseudo code of the dynamic slack management algorithm for energy efficiency is given in Figure 3.

```
Dynamic (t, p)
{
 for each processor p_j // j = 1 to pro ( number of processor)
 {
  for each task t_i in queue // i = 1 to n( number of current tasks in the queue)
   {
    //RV –precedence constraint vector, RV1- resource constraint vector
     if((RV_i = null) and (RV1_i = null)) // independent tasks
        {
                t_i.start_time = total_exec_time of p_j // initially 0
                if (t_i.start_time < t_i.arrival_time)
                        t_i.start_time = t_i.arrival_time
                t_i.exec_time = t_i.acet
                t_i.end_time = t_i.start_time + t_i.exec_time
                total_exec_time += t_i.exec_time
        }
        else
        {
                //t_k the task needed for task t_i either from precedence or resource constraint
                if(t_i.p = t_k.p) // Both task assigned to the same processor
                        t_i.start = t_k.end_time
                else
                        t_i.start = t_k.end_time + 1 //communication cost
                t_i.start_time = total_exec_time of p_j
                if (t_i.start_time < t_i.arrival_time)
                        t_i.start_time = t_i.arrival_time
                if(t_i.start_time < t_i.start)
                        t_i.start_time = t_i.start
                t_i.exec_time = t_i.acet
                t_i.end_time = t_i.start_time + t_i.exec_time
                total_exec_time+= t_i.exec_time
        }
        if((RV_i=null) and (RV1_i=null))
        {
                if(t_i is the last assigned task to p_j)
                        t_i.slack_time = deadline – t_i.end_time
                else
                        t_i.slack_time = Next assigned task in the same processor – t_i.end_time
        }
        else
        {
                t_i.slack_time = t_k.start_time – t_i.end_time
        }
   }
  }
}
```

Figure 3. Pseudo Code for Dynamic Slack Management Algorithm

5. SIMULATION AND ANALYSIS OF RESULTS

A simulator was developed to simulate voltage scalable processor which dynamically adjusts the processor speed according to the algorithm selected. A continuous voltage scaling model is used and hence the processor speed can be adjusted continuously from its maximum speed to a minimum speed which is assumed to be 25% of its maximum speed.

For simulation, scheduling task sets and task graphs are generated using the following approach:

- Task sets are randomly generated with parameters such as arrival time, acet, wcet and resource constraints.
- The wcet is taken randomly and acet is also randomly generated such that it is 40 to 100 percent of wcet.
- The overall deadline is generated such that it is always greater than or equal to the sum of acet of all the tasks in DAG.

Task graph is randomly generated using adjacency matrix where 0 represents the tasks that are not dependent on any other tasks and 1 represents the dependency, with varying breadth and depth.

5.1. Evaluation Parameters

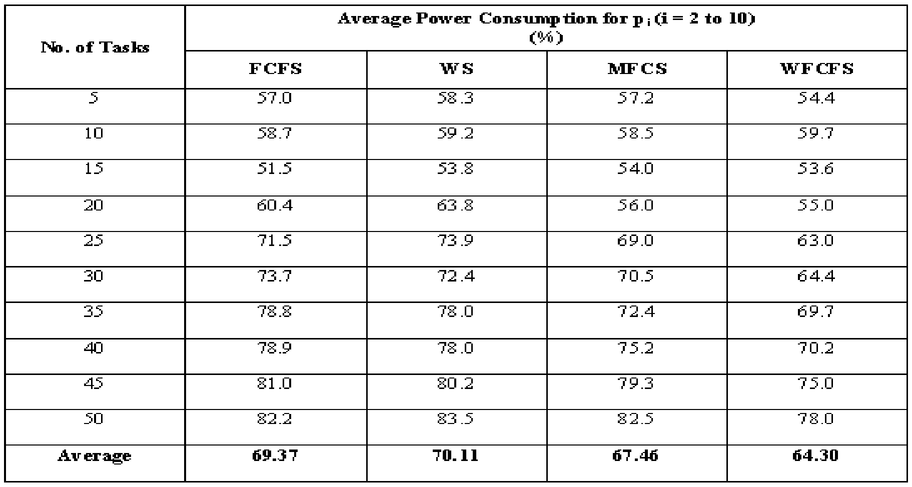
The performance parameters considered for evaluating and comparing our system with the existing systems are Power Consumption Ratio and Deadline Hit Ratio.

Power consumption ratio is the rate of power consumed by the system at scaled down voltage and frequency to the rate of power consumed at maximum operating voltage and frequency. Deadline hit ratio is defined as the number of input tasks that complete execution before their respective deadlines to the total number of tasks admitted in the system for execution.

5.2. Result Analysis

The power consumption ratio between the existing and the proposed algorithms were analyzed. For each set of tasks varied from 5 to 50, the number of processors is kept constant and the power consumption ratio for a minimum of ten randomly generated DAGs was calculated. The average of these values was plotted in the chart. This method was repeated by changing the number of processors and the comparison was made between the existing and the proposed algorithms.

Table 1. Comparison of Power Consumption



No. of Tasks	Average Power Consumption for p_i (i = 2 to 10) (%)			
	FCFS	WS	MFCS	WFCFS
5	57.0	58.3	57.2	54.4
10	58.7	59.2	58.5	59.7
15	51.5	53.8	54.0	53.6
20	60.4	63.8	56.0	55.0
25	71.5	73.9	69.0	63.0
30	73.7	72.4	70.5	64.4
35	78.8	78.0	72.4	69.7
40	78.9	78.0	75.2	70.2
45	81.0	80.2	79.3	75.0
50	82.2	83.5	82.5	78.0
Average	**69.37**	**70.11**	**67.46**	**64.30**

The average power consumption for the existing and proposed algorithms is shown in Table 1. The value in each row is average of values obtained by varying the number of processors from 2 to 10. If the number of processors in the distributed embedded system is two the algorithms were not able to schedule more than 10 tasks, as all task set cases above 10 missed their deadline. From the table it is seen that the average power consumption of the proposed algorithm is 5 to 6 percent more than the exixting ones. This comparison is presented as a chart in Figure 4.

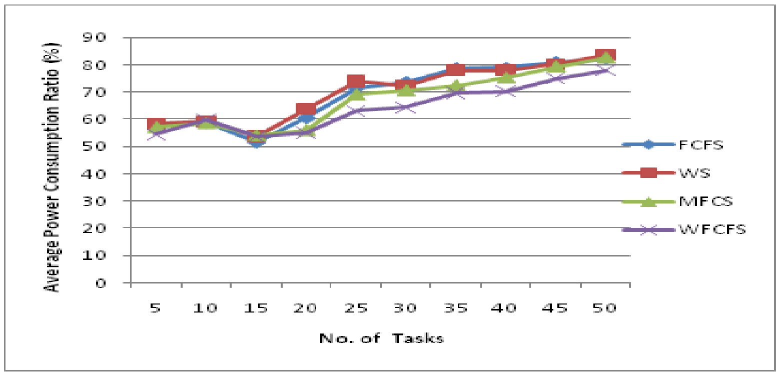


Figure 4. Comparison of Power Consumption Ratio

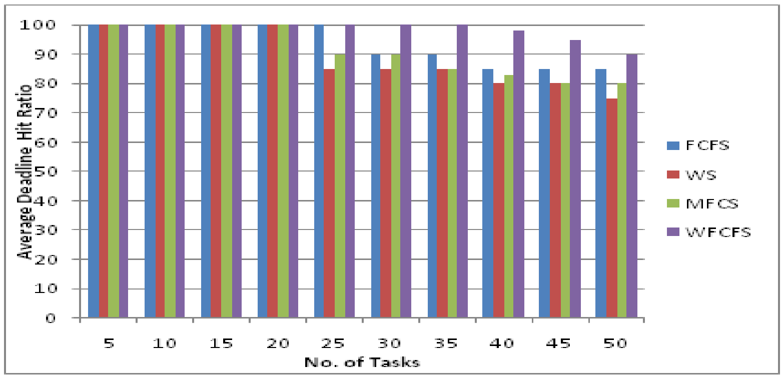


Figure 5. Comparison of Deadline Hit Ratio

For all cases from p = 2 to 10, proposed algorithm WFCFS with the dynamic slack management technique consumed less power than the existing MFCS, FCFS and WS algorithms with the SRP slack management technique, when resource contraint is added to the real-time application in addition to the precedence and time constraints.

The deadline hit ratio is also compared among the existing and proposed algorithms by varying the number of tasks from 5 to 50 and the number of processors from 2 to 10. The averge deadline hit ratio is shown in Figure 5. The maximum number of tasks is taken to be 50, because real-time applications with dependent tasks above 50 is very very rare. The number of processors is also limited to 10, as this number itself rare for a distributed embedded system. Upto p = 6 the proposed algorithm gives better deadline hit ratio compared to the existing ones. For p>=7, all the algorithms resulted in 100% deadline hit ratio.

6. CONCLUSION

In this work, an energy efficient real-time scheduling algorithm for distributed embedded systems is presented. This scheduling algorithm is capable of handling task graphs with precedence and resource constraints in addition to timing constraints. The major contribution of this work is the development of improved FCFS scheduling algorithm called Weighted FCFS. The proposed dynamic slack distribution technique efficiently utilizes the available slack and in turn increases the efficiency of the system. It also exploits the idle intervals by putting the processor in power down mode to reduce the power consumption. The proposed algorithm increases the deadline hit ratio when compared to the existing ones and thus increases the reliability. Simulation results show 5 to 6 percent less power is consumed by the proposed algorithm compared with the existing algorithms.

Fault tolerant issues may be considered in the future work for such real-time distributed embedded systems. Another important future work may be extending this algorithm for a heterogeneous distributed real-time system with varying capacity processors and communication links.

REFERENCES

[1] Rabi N. Mahapatra and Wei Zhao, (2005) “An Energy-Efficient Slack Distribution Technique for Multimode Distributed Real-Time Embedded Systems”, IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 7.

[2] Changjiu Xian, Yung-Hsiang Lu and Zhiyuan Li, (2008) Dynamic Voltage Scaling for Multitasking Real-Time Systems with Uncertain Execution Times, In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 8.

[3] Subrata Acharya and Rabi N. Mahapatra, (2008) A Dynamic Slack Management Technique for Real-Time Distributed Embedded Systems, In: IEEE Transactions on Computers, vol. 57, no. 2.

[4] W. Yuan and K. Nahrstedt, (2003) Energy-efficient soft real-time CPU scheduling for mobile multimedia systems**,** In: ACM Symposium on Operating Systems Principles, pp. 149-163.

[5] C. Shen, K. Ramamritham and J.A. Stankovic, (1993) Resource reclaiming in multiprocessor real-time systems, In: IEEE Transactions on . Parallel and Distributed Systems, vol. 4, no. 4, pp. 382-397.

[6] T. Ishihara and H. Yasuura, (1998) Voltage Scheduling Problem for Dynamically Variable Voltage Processors, In: International Symposium on Low Power Electronics and Design, pp. 197-202.

[7] L. Benini, A. Bogliolo, and G. De Micheli, (2000) A Survey of Design Techniques for System-Level Dynamic Power Management, In: IEEE Transactions on VLSI Systems, pp. 299-316.

[8] P. B. Jorgensen and J. Madsen, (1997) Critical path driven co synthesis for heterogeneous target architectures, In: International Workshop on Hardware/ Software Code, pp. 15–19.

[9] M. T. Schmitz and B. M. Al-Hashimi, (2001) Considering power variations of DVS processing elements for energy minimization in distributed systems, In: International Symposium on System Synthesis, pp. 250–255.

[10] C. Lu, J.A. Stankovic, G. Tao, and S.H. Son, (2002) Feedback Control Real- Time Scheduling: Framework, Modeling, and Algorithms, In: Real-Time Systems Journal, Special Issue on Control-theoretical Approaches to Real-Time Computing, pp. 85-126.

[11] Santhi Baskaran and P. Thambidurai, (2010) Power Aware Scheduling for Resource Constrained Distributed Real Time Systems, In: International Journal on Computer Science and Engineering Vol. 02, No. 05, pp. 1746 -1753.

[12] C.M. Krishna and Shin K. G., Real-Time Systems, (1997) Tata McGraw-Hill.

[13] Subrata Acharya and Rabi N. Mahapatra, (2008) A Dynamic Slack Management Technique for Real-Time Distributed Embedded Systems, In: IEEE Transactions on Computers, Vol. 57, No. 2.

[14] L. Sha, R. Rajkumar, and J.P. Lehoczky, (1990) Priority Inheritance Protocols: An Approach to Real-Time Synchronization, In: IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175-1185.

[15] Peng Li, Haisang Wu, Binoy Ravindran and E. (2006) Douglas Jensen, A Utility Accrual Scheduling Algorithm for Real-Time Activities with Mutual Exclusion Resource Constraints, In: IEEE Transactions on Computers, Vol. 55, No. 4.

[16] G. Manimaran, C. Siva ram Murthy, Machiraju Vijay, and K. Ramamritham, (1997) New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems, In: Journal of Parallel and Distributed Computing, Vol. 44, No. 2, pp. 123-132.

[17] C. Shen, K. Ramamritham and J.A. Stankovic, (1993) Resource reclaiming in multiprocessor real-time systems, In: IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 4, pp. 382-397.