DYNAMIC VOLTAGE SCALING FOR POWER CONSUMPTION REDUCTION IN REAL-TIME MIXED TASK MODEL

Arya Lekshmi Mohan[1] and Anju.S.Pillai[2]

[1]Department of Electrical and Electronics Engineering, Amrita University, Coimbatore India
aryalekshmim@yahoo.co.in

[2]Department of Electrical and Electronics Engineering, Amrita University, Coimbatore India
s_anju@cb.amrita.edu

ABSTRACT

The reduction in energy consumption without any deadline miss is one of the main challenges in real-time embedded systems. Dynamic voltage scaling (DVS) is a technique that reduces the power consumption of processors by utilizing various operating points provided to the DVS processor. These operating points consist of pairs of voltage and frequency. The selection of operating points can be done based on the load to the system at a particular point of time. In this work DVS is applied to both periodic and sporadic tasks, and an average of 40% of energy is reduced. The energy consumption of the processor is further reduced by 2-10% by reducing the number of pre-emption and frequency switching.

KEYWORDS

Dynamic Voltage Scaling, Pre-Emption, Frequency Switching, Earliest Deadline First

1. INTRODUCTION

The controlling action in a real-time embedded system is done by microprocessors. In a battery powered system the major part of power is consumed by the microcontroller. The most commonly using microprocessors are CMOS processors. The main characteristic of a CMOS processor is that its static power dissipation is very low. The dynamic power taken by a CMOS processor is given by

$$P=c_{eff}v^2f \qquad (1)$$

Where P is the power consumed by the processor, c_{eff} *is* the load capacitance, v is the operating voltage and f is the operating frequency. The power consumption of CMOS processor can be reduced either by reducing the voltage or by reducing the frequency.

There are different techniques evolved to reduce the power consumption of processors. DVS processor design is one among them. DVS processors mainly focusing on power management. In recent years, many DVS enabled processors came across, such as the Intel XScale, Transmeta

DOI: 10.5121/csit.2011.1206

Crusoe etc. They have discrete number of voltages and its corresponding frequency levels to switch in between. As the number of voltage levels increased the power consumption of the system will get decreased, in fact the tasks will get more choices to select frequency and voltage. The problem with increase in number of voltage and frequency level is that there is a chance of increased frequency switching.

A real-time task can be defined as a set of instructions that executes to perform a function such as I/O, peripherals etc. The characteristics of a periodic task are release time, deadline and its execution time, whereas the release time and deadline of a sporadic task is undefined. Release time is the time at which a task is ready for running. Execution time is the maximum time required for a task to complete. Deadline is the time before that particular task should complete its work. In a realistic scenario the real-time systems have to handle a number of periodic and sporadic tasks without crossing its deadlines. In this work we consider the system with both sporadic and periodic tasks. For simulation, the release time of sporadic task is generated randomly.

Dynamic voltage scaling technique has more to do with real time scenarios like wireless sensor networks which are battery powered. In such systems the sensor node has to monitor the environmental changes and send the information to a controlling station. In industries the sensor node continuously monitors the environmental parameters such as temperature, pressure etc, and sends them periodically to a controlling station; such an application can be named as a periodic task. In a sudden if fire occurs in that industry then it has to send that information with prior importance. Such a task can be defined as a sporadic one. A DVS processor can handle all these tasks with minimum power consumption.

When applying Dynamic voltage scaling in real time tasks, if the frequency of a processor decreases, its corresponding voltage also reduces and the task will take more time to complete the execution. Thus the power consumption reduces, with the assumption that the tasks will not miss any of its instances when it executes with maximum frequency.

The next phase of this paper delivers the contributions of other researchers in the field. Third phase explains the computational model. Fourth section describes in dynamic voltage scaling algorithm given by Pillai et.al.which has been implemented with modifications. Fifth phase deals with handling of sporadic tasks along with periodic tasks. Sixth phase conveys the simulation results.

2. RELATED WORKS

Over the last years, DVS has been adopted as one of the most effective technique for reducing energy consumption in embedded systems. An introduction to DPM and DVS is given in paper [1]. In that paper the authors discussed the trading-off between energy consumption and performance [1]. The paper by Johan Pouwelse et al. describes the importance of DVS in wearable computers. An implementation was done in Embedded Strong ARM 1100 processor which supports frequency scaling. The result of their experiment is given as energy per instruction at minimal speed is 1/5 of energy required at full speed [2].

One of the disadvantages of Dynamic Voltage scaling is increasing the number of preemptions. To reduce that overhead Preemption-Aware DVS for hard Real-Time Systems is implemented. Two preemption control Methods are proposed in this paper they are accelerated-completion based technique and delayed preemption based technique. In that work the preemption is reduced by postponing the high priority tasks and forces low priority tasks to complete the execution [3]. Integration of preemption threshold scheduling and dynamic voltage scaling is given by Jejurikar et al. They present an algorithm to compute threshold preemption levels for tasks with given static slowdown factors. The proposed algorithm improves upon known algorithms in terms of time complexity. Experimental results show that preemption threshold scheduling reduces on an average 90% context switches, even in the presence of task slowdown. The system model for the experiment consists of periodic tasks only. [4]

Transition time is another overhead in the case of DVS processors. Transition time is the time taken by a task to switch from one frequency to another. There are two methods for transition elimination is given in the paper [5], they are offline and online. Another effort for reducing the frequency switching is done by Muhammad et al. [6] Frequency switching not only increase the power consumption but it wastes the processor cycles. By this algorithm an average of 30% of the frequency switching is reduced. The dynamic voltage scaling algorithm implementation and testing is done by Pillai et al. in [7]. In this paper the authors proposed three algorithms for voltage scaling and the proposed algorithms are implemented in 2.2.16 Linux kernel with hardware platform as AMD K6-2+ processor. [7].

An efficient work for task set generation is done by Bini et al. The main algorithms proposed in this work are Uunifast, Ufitting, Uscaling, Uunisort. Among these algorithms Uunifast is widely accepted one. These algorithms provide utilizations for each task in the task set and, from these utilizations a set of execution times and deadlines are generated. [8]

Sporadic task handling is another feature for Dynamic voltage scaling systems. A DVS algorithm, called DVSST, is introduced by Ala′ Qadi et al. In that algorithm they handled sporadic tasks in conjunction with the pre-emptive EDF scheduling algorithm [9]. An acceptance test for sporadic task instances is given in paper [10]. There are two steps in acceptance test. First step is processing and second step is slack updating. In this algorithm the acceptance of a sporadic instance is based on slacks available in the system. Generally the parameters of sporadic tasks are undefined. In paper [11], the authors provided one algorithm called Total Bandwidth Server algorithm for calculating the virtual deadline for the aperiodic tasks.

3. System Model

3.1. Task Model

The task set consists of both periodic and sporadic tasks. Let R (r_i, c_i, d_i) represents a periodic task with release time r_i, worst case computation time as c_i and relative deadline as d_i. The sporadic task can be represented by Si (A_i, C_i, D_i), where A_i is the arrival time of the task, C_i is worst case execution time and D_i is the deadline. Arrival time of the sporadic task is generated randomly for simulation.

3.2. Task set Generation

Task set is generated using an algorithm called UUnifast [8]. A task set with required number of tasks can be generated for a given utilization. The generated task set is scheduled under EDF scheduler and hence the maximum utilization is taken as 1.

3.3. Task scheduling Algorithm

The task scheduling policy is Earliest Deadline First; which is a dynamic scheduling algorithm. The necessary and sufficient condition for scheduling a task set under EDF is that the utilization should be less than or equal to 1.

$$U=\sum_{i=1:n} C_i/D_i, \quad (2)$$

In given equation 2, 'U' denotes the utilization, C_i denotes the computation time of ith task, and D_i represents the deadline of ith task. The assumptions are taken for EDF is that period of the task is equal to its relative deadline and all tasks are independent.

3.4. Processor Model

The processor selected for this work is a DVS enabled processor with a set of operating points. Operating points are a set of frequency- voltage pairs. Let subscript… f_1, f_2, f_3…f_n be the frequencies and v_1, v_2, v_3….v_n be the corresponding voltages. Based on the load at a particular point of time, one of these operating points can be selected provided any of the tasks will never miss its deadline when it is executed with its worst case execution time.

4. Dynamic Voltage Scaling

Dynamic voltage scaling algorithm used in this work is a modification of look-ahead RT Dynamic voltage scaling algorithm proposed by Pillai et al [7]. Look-ahead algorithm calculates the number of cycles that must run within an interval and from that information the frequency required to run the task is calculated. This algorithm tries to run the high priority task with low frequency and low priority task with high frequency; in order to meet the deadline in time. This algorithm activates only at scheduling points. The algorithm utilizes the slacks to reduce the frequency. The actual execution time is assumed to be less than the worst case execution time. If a task runs with execution time less than the worst case execution time; then the tasks can run with a frequency less than the maximum frequency. In this paper the algorithm concerned only with periodic tasks. In this algorithm the pre-emption and frequency switching are considered to be negligible. This algorithm is applied only on periodic tasks.

At an instant if a low priority task is getting pre-empted by a high priority task then the new algorithm will check if sum of remaining computation time of previous task (pre-empted task) and the computation time of current task is less than or equal to difference in deadline of current task and current time, then the present pre-emption can be avoided. In most of the cases pre-emption causes frequency switching. If frequency of current task is different from the frequency of previous task then the algorithm will checks whether any task in the ready queue with frequency same as the previous task frequency. If such a task is there then the algorithm checks whether to execute that task for current instance. If it is possible then that task will take for execution otherwise not.

The algorithm can be briefly described by the code given below:

```
if previous_task ≠current_task
for i=1: no_of_task
if frequency(i)= frequency(previous)
if computation(current_task)+ computation(i) + t <deadline(current_task)
current_task=i
break;
else
current_task=current_task
end
end
if computation(previous_task ≠ 0)
if computation_current_task+computation_previous_task < = deadline_current_task – current_tick
current_task= previous_task
end
end
end
```

Figure.1. Pseudo code for frequency switching

4.1 Example task set for modified algorithm

Table 1. Task set for modified algorithm

TASK NO	EXECUTION TIME	DEADLINE
1	1	4
2	1	5
3	2	10
4	4	12

In the given example of the task set, first instance of Task1 and Task2 run with normalized frequency .5, and Task 3 runs with frequency of .75. At time t=4.2 ms the task T1 is ready for running with a frequency of .5 and there is another task Task4 having same frequency as the previous task that had already run (T3) , so the algorithm checks whether the Task4 can run without missing the deadline of Task1. The algorithm selects Task 4 and it executes till time, t=6.4ms. The same algorithm also checks for pre-emption. At t=12 Task 3 is pre-empted by Task 1 according to DVS algorithm. New algorithm verifies whether that pre-emption can be avoided, by reducing the pre-emption by continuing the execution of Task3.

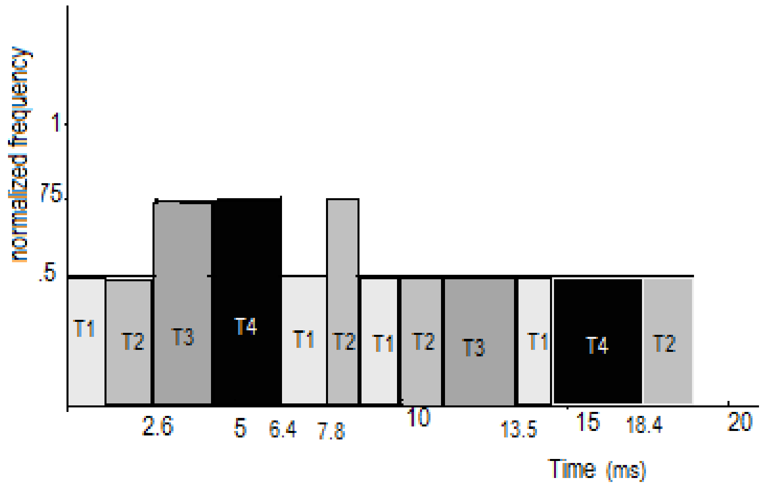

Figure 2. Example for frequency switching Reduction

5. Sporadic Task Handling

In a mixed task model the periodic task is handled along with the sporadic tasks. Sporadic tasks are tasks which occurs at random intervals. Sporadic tasks does not have predefined release time or deadline. A sporadic task can be represented by three parameters.

$$T_i = (e_i, g_i, d_i), \qquad (3)$$

Where e_i is the worst case execution time of the task, g_i denotes the minimum separation between two consecutive instances of task, d_i is the relative deadline. The minimum separation (g_i)

between two successive instances of the task implies that once an instance of a sporadic task occurs, the next instance cannot occur before g_i time units elapsed.

If an instance of a sporadic task is getting the high priority by the earliest deadline first policy, the algorithm will conduct an acceptance test based on the execution time, available slacks and deadline of sporadic job, and decide whether to accept or reject the job. Accepting a task implies that the task will place in a ready queue and will take for execution according to priority and will complete without affecting any deadline of periodic task. Acceptance of sporadic task directly relates to the amount of slack remaining for that interval. In this work more priority is given to periodic tasks.

The deadline for sporadic task is obtained by using TBS (Total Bandwidth Server) algorithm. This is based on the assigned bandwidth, and offer excellent performance with retaining the optimality of dynamic-priority scheduling [11]. When a sporadic task occurs, the server assigns the possible deadline to sporadic task, which is a virtual one. Once the sporadic task is assigned the virtual deadline, it is scheduled with other periodic tasks according to the EDF algorithm. The virtual deadline of sporadic task is determined based on the remaining utilization of server. Suppose that a system has a bandwidth Usrv, assuming that the kth sporadic task S_k arrives on the system at a_k , with an execution time Ck, then its virtual deadline V_k is calculated by the equation given below, where $v_0=0$ by definition [11].
The virtual deadline V_k is given as

$$V_k=\max\{a_k,v_k-1\}+(C_k/Usrv), \qquad (4)$$

5.1. Sporadic Acceptance Test

Consider a system having n number of periodic tasks and m number of sporadic tasks. Sporadic task is represented by S_i (A_i, C_i, D_i) and periodic task is represented by R (r_i, c_i, d_i). Whenever a sporadic request occurs then the acceptance test for sporadic task is done. Acceptance test have two main steps. One is pre-processing step and other is slack updating step [10].

During pre-processing step the system computes the available slacks to incorporate the sporadic request [10]. Before the system begins execution, the scheduler computes the slack δ_i, for each periodic request $R_i(r_i,c_i,d_i)$, which is the maximum amount of time available before d_i to execute the sporadic request before causing R_i to miss its deadline. Let H_i denotes all the periodic requests whose deadline is before d_i.

$$\delta_i=d_i-c_i-\sum c_j \qquad (5)$$

Where c_j is the computation time of all the periodic tasks R_i whose deadline is before d_i.
Acceptance test relies on information on slack of each request in the system. It has two steps [10].
1. When a sporadic request Si arrives, first determine the amount of slack available in the system before Di. If the amount of slack is less than the computation time of current sporadic request Si is rejected, since there is not enough time to schedule Si before its deadline.
2. If there is enough slack, then check if accepting Si would cause any request in the system whose deadline is after D_i to miss its deadline. Si is accepted if no other deadline is missed or rejected otherwise.

Let Δk be the slack, then if Δk>1 then the sporadic request at that instant can be accepted; otherwise rejected.

$$\Delta k=\delta p+ (D_k-dp)-I-Ck-SP-\sum_{Dk<di} fi-\sum_{Di<Dk} Ci-\sum_{Dk<di} Fi \qquad (6)$$

Where δp – initial slack of periodic request R_p whose deadline is closer to the deadline of sporadic request D_k, and whose deadline is earlier than D_k.
D_k - deadline of currently requested sporadic task.
Dp- Deadline of periodic request Rp whose deadline is closer to the deadline of sporadic request Dk, and whose deadline is earlier than Dk.
I - The amount of idle time before the current sporadic request.
Ck is the worst case execution time of currently requested sporadic request.
fi – the sporadic task requests at 'tc ', then the current requests of each periodic task can have deadline later than dp and yet start executing before 'tc' but not yet completed.
SP is the amount of time that is consumed by the sporadic requests which are completed.
$\sum_{Di<Dk}$ (Ci) This amount of time is reserved for sporadic requests which have not been completed and have deadlines before Dk.
$\sum_{Dk<di}$ (Fi) This amount of time has been consumed by sporadic requests that have not completed and have deadlines after Dk.

5.2. Example for calculation of virtual deadline

Consider two periodic tasks and one sporadic task given in table. Task 1 and Task 2 are periodic tasks and Task 3 is a sporadic task. Task 3.1 is the first instance of sporadic request and task 3.2 is the second instance of sporadic request. The minimum inter arrival time for sporadic task instance 1 is randomly taken as 8, and for second instance as 7.Here in this example the arrival time for first instance is obtained randomly as 3,and the virtual deadline is 8 from the below example. So finally deadline for first instance of the sporadic task is taken as 3+8=11.
The virtual deadline for sporadic task is computed as:
Vd1=max{0,3+(execution_sporadic/(1-utilization_of_periodic))} (7)

The arrival time for next instance is randomly taken as 13. Then the virtual deadline for next instance is obtained as
Vd2=max{11,13+(execution_sporadic/(1-utilization_of_periodictasks))} (8)

5.3 Example for Sporadic Task Handling

Table2. Periodic and sporadic task parameters

Task number	Arrival time	Execution time	Relative deadline
Task 1	0	2	4
Task 2	0	2	8
Task 3	8	2	13
Task 4	3	1	17

In the table given above the tasks 1 and task2 are periodic tasks and tasks 4 and 5 are sporadic tasks. The arrival time of sporadic tasks are 8 and 3 milli seconds respectively. The minimum inter arrival time for task 3 and 4 are 7 and 8 respectively. Although these parameters are undefined, in simulation these parameters are generated randomly. Figure 3 shows the trace for the task scheduled under EDF, and all tasks scheduled without missing the deadline and power calculated is 215 mW. In figure 4, the scheduled task after applying DVS is given. In that figure most of the slacks are utilized and the power measured is 125.65 mW. In figure 5, the output trace for modified algorithm is given and power consumption reduced to 120 mW. In that trace preemptions are avoided. At t=12 a frequency switching occurred after applying DVS, but in modified algorithm the frequency switching due to the pre-emption of task 3 is avoided.

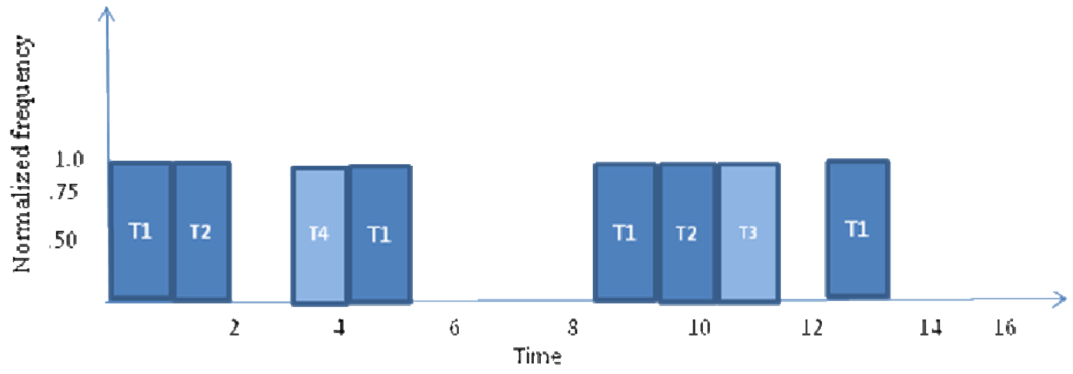

Figure 3 Output Trace For Task Set Scheduled Under EDF

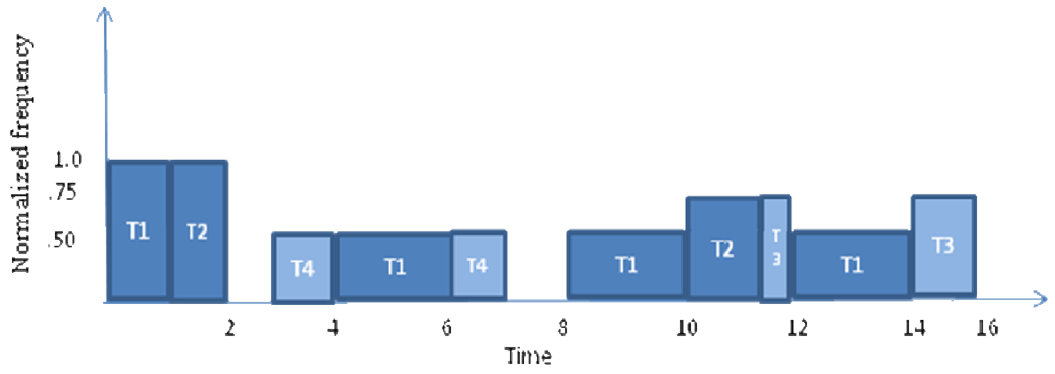

Figure 4 Output Trace For Task Set Scheduled Under DVS

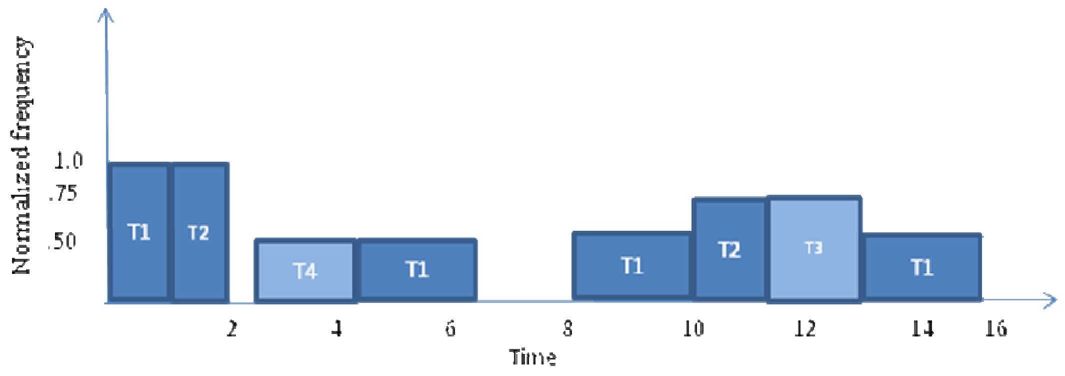

Figure 5 Output Trace For Task Set Scheduled Under Modified Algorithm

6. Simulation Results

The simulation is done with 5 different task sets with both periodic and sporadic tasks, and the average power saving calculated from the simulation result is 40%.

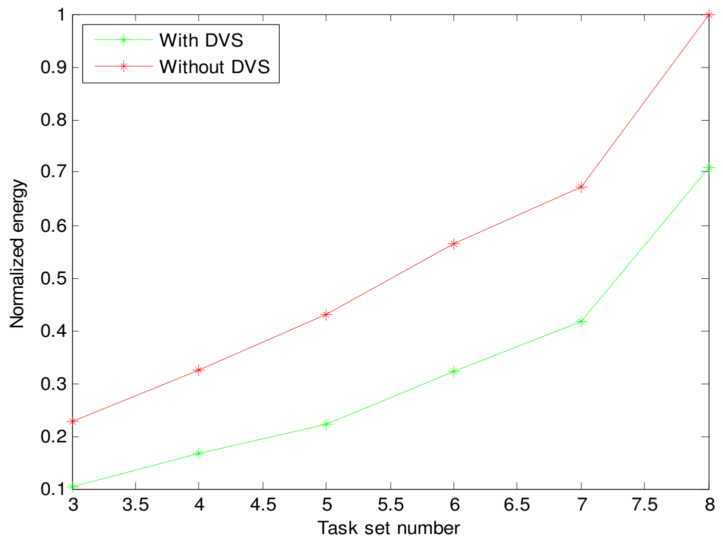

Figure 6. Normalized energy plot for mixed task model

Task pre-emption requires about .2 milli joules energy for context switching [4]. In figure 6 Normalized energy Vs utilization graph is plotted. From this figure it is clear that power consumption is further reduced with the modified algorithm. About 2-10% of power is reduced in the modified work as compared to the previous algorithm. The simulation is done for periodic tasks only. Figure 7 shows normalized energy after reducing the pre-emption.

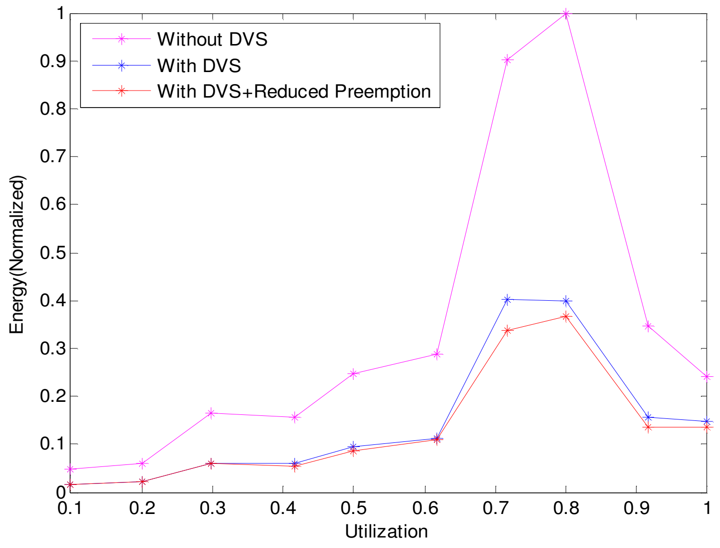

Figure 7. Plot for Normalized Energy Vs utilization

7. Conclusions

The dynamic voltage scaling technique is applied to the processors to alleviate the excessive power consumption. In real time systems in addition to periodic tasks sporadic task may occur. The handling of sporadic task without any deadline miss has prime importance. A modified algorithm for dynamic voltage scaling is discussed above for the mixed task set. An average of 40% of power consumption is reduced in mixed task model by reducing the pre-emption and frequency switching. Frequency switching in the system is reduced by 35% by applying the new algorithm. Frequency switching due to pre-emption is avoided in improved algorithm.

ACKNOWLEDGEMENTS

We would like to thank God Almighty for giving us this opportunity to complete this work. We would like to extend our gratefulness to Dr. T.N Padmanabhan Nambiar for providing the valuable support and help in this work.

REFERENCES

[1] Niraj K. Jha," Low Power System Scheduling and Synthesis", IEEE/ACM *International Conference on Computer Aided Design, 2001.*

[2] Johan Pouwelse, Koen Langendoen and Henk Sips "Dynamic Voltage Scaling on a Low-Power Microprocessor", *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on Oct. 2003, Volume 11 Issue:5, page(s): 812 – 826,2003.*

[3] Woonseok Kim, Jihong Kim, Sang Lyul Min," Preemption-Aware Dynamic Voltage Scaling in Hard Real-Time Systems", *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'04), 2004.*

[4] Ravindra Jejurikar and Rajesh Gupta, "Integrating Preemption Threshold Scheduling and Dynamic Voltage Scaling for Energy Efficient Real-Time Systems", *Proceedings of the Ninth International Conference on Real-Time Computing Systems and Applications, 2004.*

[5] Bren Mochocki, Xiaobo Sharon Hu, Gang Quan, "Transition-Overhead-Aware Voltage Scheduling for Fixed-Priority Real-Time Systems", *ACM Transactions on Design Automation of Electronic Systems, Vol. 12,No. 2, Article 11, April 2007.*

[6] Farooq Muhammad, Bhatti M. Khurram, Fabrice Muller, Cecile Belleudy, Michel Auguin, "Precognitive DVFS: Minimizing Switching Points to Further Reduce the Energy Consumption", *Proceedings of the 14th Real-Time and Embedded Technology and Applications Symposium, 2008.*

[7] Padmanabhan Pillai and Kang G. Shin,"Real-Time Dynamic Voltage Scaling for Low Power Embedded Operating Systems", *Symposium on Operating Systems Principles'01, 2001.*

[8] Enricobini, Giorgioc Buttazzo, "Biasing Effects in Schedulability Measures", *Proceedings of the 12th 16th Euromicro Conference on Real-Time Systems (ECRTS'04), IEEE, 2004.*

[9] Ala′ Qadi, Steve Goddard, Shane Farritor, "A Dynamic Voltage Scaling Algorithm for Sporadic Tasks", *24th IEEE International Real-Time Systems Symposium (RTSS'03) 2003.*

[10] Too-seng Tia, Jane W.S.Liu, Jun Sun, Rhan Ha, "A linear-time optimal acceptance test for scheduling of hard real-time tasks", *IEEE Transactions on Software engineering, November15, 1994.*

[11] Shinpei Kato and Nobuyuki Yamasaki, "Scheduling Aperiodic Tasks Using Total Bandwidth Server on Multiprocessors", *Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing - Volume 01, 2008.*