ENERGY REDUCTION IN WEAKLY HARD REAL TIME SYSTEMS

By

ANKITA JAIN	071441
ISHA WALIA	071412
TALWINDER KAUR	071429

Under the Supervision of Mr. YASHWANT SINGH

May – 2011

Submitted in Partial Fulfilment of the Degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLGY (IT)

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT

TABLE OF CONTENTS

CERTIFICATE

This is to certify that the work titled **"ENERGY REDUCTION IN WEAKLY HARD REAL TIME SYSTEM"** submitted by **ANKITA JAIN (071441), ISHA WALIA (071412) , TALWINDER KAUR (071429)** in the partial fulfillment for the award of degree of Bachelor of Technology (IT) of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor _____

Name of Supervisor _____

Designation _____

Date _____

ACKNOWLEDGEMENT

It gives us great pleasure in presenting the project report for our project on 'ENERGY REDUCTION IN WEAKLY HARD REAL TIME SYSTEM'. We would like to take this opportunity to thank our internal guide Mr.Yashwant Singh for giving us all the help and guidance we needed. We are really grateful to her for her kind support throughout the analysis and design phase. We are also grateful to Brig. S.P Ghrera, Head of Computer Department, Jaypee Institute of Information and Technology for giving important suggestions.

ANKITA JAIN (071441)
ISHA WALIA (071412)
TALWINDER KAUR (071429)

DATE _____

ABSTRACT

We propose a scheduling algorithm to minimize the energy consumption in weakly hard real time systems, i.e., the (m, k)-model, which requires at least m out of k consecutive instances of a task meet their deadlines. Firstly, we recommend a strategy to partition real time jobs into mandatory and optional part in order to meet weakly hard real time constraint. Secondly, introduced an approach which can effectively reduce the energy by 15% along with DVS strategy which provide significant energy savings while maintaining real time deadline guarantees.

Real-Time systems are becoming pervasive. Typical examples of real-time systems include Air Traffic Control Systems, Networked Multimedia Systems, and Command Control Systems etc. In a Real-Time System the correctness of the system behaviour depends not only on the logical results of the computations, but also on the physical instant at which these results are produced. Real-Time systems are classified from a number of viewpoints i.e. on factors outside the computer system and factors inside the computer system. Special emphasis is placed on hard and soft real-time systems. A missed deadline in hard real-time systems is catastrophic and in soft real-time systems it can lead to a significant loss. Hence predictability of the system behaviour is the most important concern in these systems. Predictability is often achieved by either static or dynamic scheduling of real-time tasks to meet their deadlines. Static scheduling makes scheduling decisions at compile time and is off-line. Dynamic scheduling is online and uses scheduling test to determine whether a set of tasks can meet their deadlines.

In this we propose a different approach. We introduce the concept of Inverse RM among tasks as a new metric for expressing performance requirements. The meaning of this importance relationship is to express that in a schedule it is desirable to run a task in preference to other ones. This model is more intuitive and less restrictive than traditional utility-based approaches. By extensive simulation, we show that the new approach combined with the existing techniques algorithm outperforms the rate monotonic priority algorithm and EDF in several practical problems are discussed in the thesis.

Signature of Student

Name

Date

Signature of Supervisor

Name

Date

Signature of Student

Name

Date

Signature of Student

Name

Date

LIST OF FIGURES

LIST OF ABBREVIATIONS

SYMBOLS	DESCRIPTION
$e_{p,i}$	Computation required by the frequency dependent components of task τ_i
$e_{d,i}$	Computation required by the frequency independent components of task τ_i
rel_i^j	Release time of a job τ_i^j, i.e., $rel_i^j = j * p_i$
D_i^j	Absolute deadline of a job τ_i^j, i.e., $D_i^j = j * p_i + d_i$
ft_i^j	Finish time of a job τ_i^j
s_{ci}	Critical speed of the processor for the task τ_i
s_{ai}	Speed of the processor assigned to the task τ_i
s_{ai}^j	Speed of the processor assigned to the job τ_i^j
a_i^j	Frequency independent component a_i is associated with the task τ_j
$E_{dslp,i}^j$	Energy consumed per unit time by the device a_i associated with task τ_j in sleep state
$E_{dact,i}^j$	Energy consumed per unit time by the device a_i associated with task τ_j in active state
thd_i	DPD threshold of the device a_i

E_{pidle}	Energy consumed per unit time by the processor in the idle state
E_{pslp}	Energy consumed per unit time by the processor in the sleep state
E_{pi}	Energy consumed per unit time by the processor when running at a speed s_i ($E_{pi} = Cs_i^3$ where C is constant)
thp	DPD threshold of the processor
L	MK_hyperperiod
$\Re$	Pre-emption Overhead is context switching time required when a higher priority pre-empts a lower priority task
$E_{\Re}$	Energy consumed during each pre-emption

CHAPTER 1: INTRODUCTION

1.1 REAL TIME SYSTEM

A real time task is one for which quantitative expression of time is needed to describe its behaviour. This quantitative expression of time usually appears in the form of a constraint on the time at which the task produces results. The most frequently occurring time constraint is a deadline which is used to express that a task is required to compute its result within some tasks: deadline. "A real time task can be classified into hard, soft, or firm real time task depending on the consequences of a task missing its deadline." It is not necessary that all tasks of real time applications belong to the same category. It is possible that different tasks of real time systems can belong to different categories. There are various examples of real time systems which are illustrated in the next section. Real time systems are being implemented in various fields one of them is **INDUSTRIAL APPLICATION,** constitute a major usage area of real time systems. A few examples of industrial applications of real time systems are:

1.1.1 Automated Car Assembly

An automated car assembly plant is an example of a plant automated system. In an automated car assembly plant, the work product (partially assembled car) moves on a conveyor belt. Alongside the conveyor belt, several workstations are placed. Each workstation performs some specific work on the work product such as fitting engine, fitting floor, fitting wheel and spray painting the car etc. as it moves on the conveyor belt. An empty chassis is introduced near the first workstations. At each workstation, a sensor senses the arrival of the next partially assembled product. As soon as the partially assembled product is sensed, the workstation begins to perform its work on the work product. The time constraint imposed on the workstation computer is that the workstation must complete its work product moves away to the next workstation. The time bounds involved here are typically of the order of a few hundreds of milli seconds.

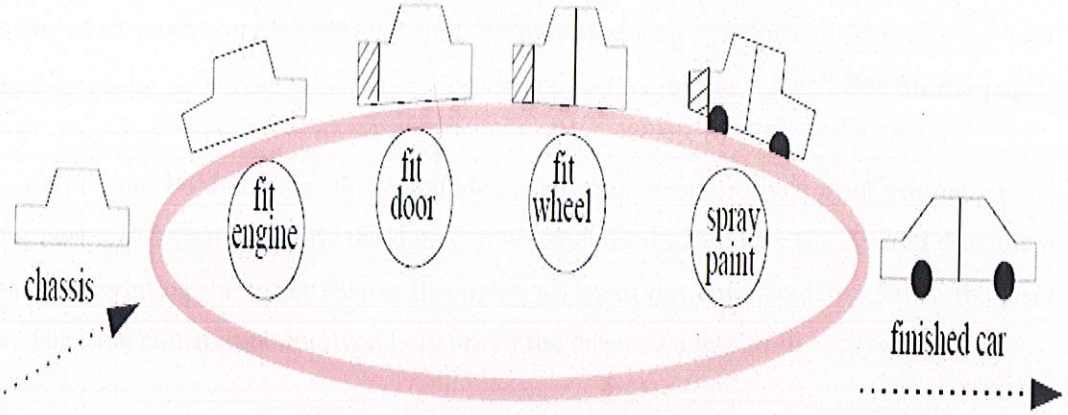


FIGURE 1.1

Figure 1.1 Automated Car Assemblies

1.1.2 Chemical Plant Control

Chemical plant control systems are essentially a type of process application. In an automated chemical plant, arel time computer periodically monitors plant conditions. The plant conditions are determined based on current readings of pressure, temperature, and chamical concentration of the reaction chamber. These paramerters are sampled periodically. Based on the values sampled at any time, the automation system decides on the corrective actions neccesary at that instant to maintain the chemical reaction at a certain rate. Each time the plant conditions are sampled, the automation system should decide on the exact instantaneous corrective actions required such as changing the pressure, temperature, or chemical concentration and carry out these actions within certain predefines time bounds. Typically, the time bounds in such a chemical plant control application range from a few micro seconds to several milli seconds.

1.1.3 Laser Printer

Most laser printers have powerful microprocessors embedded in term to control different activities associated with printing. The important activities that a microprocessor embedded in a laser printer performs includes the following : getting data from the communication

port(s) ,typesetting fonts , sensing paper jams , noticing when the printer runs out of paper , sensing when the user presses a button on the control panel , and displaying various messages to the user. The most complex activity that the microprocessor performs is driving the laser engine. The basic command that a laser engine supports is to put a black dot on the paper. However, the laser engine has no idea about the exact shapes of different fonts, font sizes, italic, underlining, boldface etc. the embedded microprocessor receives print commands on its input port and determines how the dots can be composed to achieve the desired document and manages printing the exact shapes through a series of dot commands issued to the laser engine. The time constraints involved here are of the order of a few milli seconds.

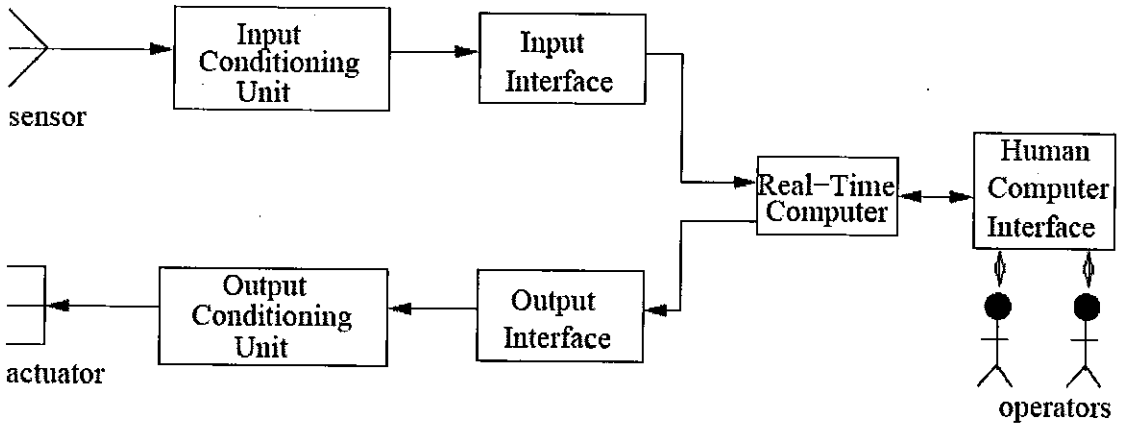


Figure 1.2 A Basic Model of a Real Time System

We need to have a basic conceptual understanding of the underlying hardware. We try to develop understanding of high level issues of the underlying hardware in a real time system. This figure shows a simple model of a real time systems in terms of its functional blocks. The sensors are interfaced with the input conditioning block, which in turn is connected to the input interface. The output interface, output conditioning and the actuator are interfaced in a complementary manner. The roles of the different functional blocks of a real time systems:

- **Sensor :** A sensor converts some physical characteristic of its enviornment into electrical signals. An example of a sensor is a phtot-volatic cell which converts light energy into

electrical energy. A wide variety of sensors are used. A temperature sensor typically operates based on the principle of a thermocouple and many other physical principle s exist.

- **Actuator :** An actuator is any device that takes its input from the output interface of a computer and converts these electrical signal into some physical actions on its enviornment . The physical actions may be in the form of motion, change of thermal, electrical. A popular actuator is a motor. Heaters are also commonly used along with several hydraulic .

- **Signal Conditioning Units** : The electrical signals produced by a computer can rarely be used to directly drive an actuator. The computer signals usually need conditioning before they can be used by the actuator. This is called output conditioning. For example, analog signals generated by a phtot-volatic cell are normally in the milli voltage range and need to be conditioned before processed by the computer. There some of the important types of conditioning such as : Voltage Amplification , Voltage level shifting .

- **Frequency Range Shifting and Filtering** : Frequency range shifting is often used to reduce the nise components in a signal. Many types of noise occur in narrow bands and the signal must be shifted from the noise bands so that noise can be filtered out.

- **Signal Mode Conversion** : A type of signal mode conversion that is frequently carried out during signal conditioning involves changing direct current into alternating current and vice-versa. Another type signal mode conversion that is frequently used is conversion of analog signals to a constant amplitude pulse train such that pulse rate is proptional to the voltage.

- **Interface Unit :** It commands from the CPU are delivered to the actuator through an output interface. An ouput interface converts the stored voltage into analog form and then outputs this to the actuator. The cpu selects a data register of the output interface and writes the necessary data to it. It takes care of the buffering and the handshake control aspects. Digital to analog conversion is frequently used in an output interface.

- **Analog to Digital Conversion :** Digital computers can not process analog signals. Therefore, analog signals need to be converted to digital form using a circuitry whose block diagram is shown above.Sampling is done by a capacitor circutory that stores voltage levels. These voltage levels can be discretized after a sampling into a step waveform.

1.2 HARD REAL TIME SYSTEMS

A real time task is one that is constrained to produce its results within a certain predefined time/unds. The system is considered to have failed whenever any of its hard real time tasks does not produce its required results before the specified time bound. The completion of an operation after its deadlines is considered useless; it may cause a complete failure or the events that occur with a strict deadline. When a process is considered hard real-time, it must complete its operation by a specific time. If it fails to meet its deadline, its operation is without value and the system for which it is a component could face failure. There are various examples of Hard Real Time Systems which are illustrated in the next section.

- A car engine control system. Such a system is considered hard, real–time because a late process could cause the engine to fail. Hard real-time systems are employed when it is crucial that a task or event is handled by a strict deadline. This is typically necessary when damage or the loss of life may occur as a result of a system failure.

- A system having hard real time tasks is a Robot .The robot cyclically carries out a number of activities including communication with the host system , logging all completed activities , sensing the environment to detect any obstacles present, tracking the objects of interest , path planning ,effecting next move etc.Now consider that the robot suddenly encounters an obstacle. The robot must detect it and as soon as possible try to escape colliding with it. It fails to respond to it too quickly (i.e. the concerned tasks are not completed before the required time bound) then it would collide with the obstacle and the robot would be considered to have failed. Therefore detecting obstacles and reacting to it are hard real time tasks.

- Another application having hard real time tasks is an anti-missile system. An anti-missile system consists of the following critical activities (tasks). An anti –missile system must first detect all incoming missiles, properly position the anti-missile gun, and then fire to destroy the incoming missile before the incoming missile can do any damage. All these tasks are hard real-time in nature and the anti-missile system would be considered to have failed, if any of its tasks fails to complete before the corresponding deadlines. For hard real-time tasks in practical systems, the time bounds usually range from several micro seconds to few milliseconds. It may be noted that a hard real-time task need not to be completed within the shortest time possible, but it is merely required that the task must complete within a specified time bound.

1.3 SOFT REAL TIME SYSTEMS

The few misses of deadline have no harm. It just decreases the overall performance of the system. When a system is considered soft, real-time, however, there is some room for lateness. For example, in a soft, real-time system, a delayed process may not cause the entire system to fail. Instead, it may lead to a decrease in the usual quality of the process or system. There are various examples of a Soft Real Time Systems Are illustrated in the next section:

- A soft real-time task is a task handling a request for a seat reservation in railway reservation application. Once the request is made for reservation, the response should occur within 20 seconds on the average. The response may either be in the form of a printed ticket or an apology message on account of unavailability of seats. Alternatively, we might state the constraint on the ticketing task as: At least in case of 95% of reservation requests, the ticket should be processed and printed in less than 20 seconds.

- Soft real-time task is web browsing .Normally, after an URL (Uniform resource locater) is clicked; the corresponding web page is fetched and displayed within a couple of seconds on the average. However when it takes several minutes to display a

requested page, we still do not consider the system to have failed, but merely express that the performance of the system has degraded.

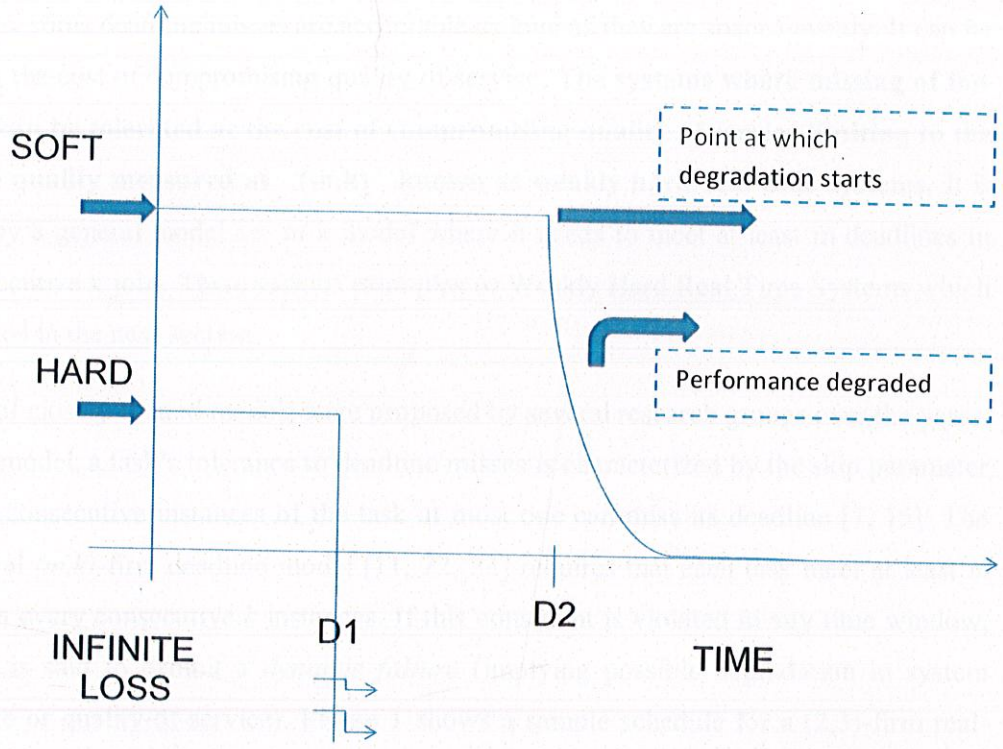


Figure 1.3 Graph for hard and soft real time system

As in the graph, D1 is the deadline for hard real time systems. The graph is plotted between utility and time. The task is completed after the deadline i.e. d1, it undergoes an infinite loss. In other case i.e. soft real time systems so here d2 is the deadline for soft real time systems where task misses its deadline d2 it leads to the performance degradation. It will not undergo severe consequences.

1.4 WEAKLY HARD REAL TIME SYSTEMS

It is the generalized case of real time systems. It is motivated by the observation for real time applications; some deadline misses are acceptable as long as they are spaced evenly. It can be tolerated at the cost of compromising quality of service. **The systems where missing of few deadlines can be tolerated at the cost of compromising quality of service limiting to the acceptable quality measured as (m,k) known as weakly hard real time systems.** It is measured by a general model i.e. m-k model where it needs to meet at least m deadlines in every consecutive k jobs. There various examples of Weakly Hard Real Time Systems which are illustrated in the next section:

A number of closely-related models were proposed by several research groups over the years. In the *skip* model, a task's tolerance to deadline misses is characterized by the skip parameter *s*: in any *s* consecutive instances of the task at most one can miss its deadline [7, 15]. The more general *(m,k)*-firm deadline model [11, 22, 23] requires that each task meet at least *m* deadlines in every consecutive *k* instances. If this constraint is violated in any time window, the system is said to exhibit a *dynamic failure* (implying possible degradation in system performance or quality-of-service). Figure 1 shows a sample schedule for a (2,3)-firm real-time task over six consecutive periods. The third instance is skipped, but the task meets its (2,3)-firm deadline constraint in the first execution window that encompasses the first three task periods, thanks to the timely completion of the first and second task instances.

- **MONITORING SYSTEMS:** Another example of a real-time task is a monitor. A task periodically performs a set of monitoring actions. The sampling period may be carefully computed (and probably overestimated) or decided as a rule of thumb. Again missing an occasional deadline means that the action from monitoring will be delayed by some bounded period of time. Provided that the effect of such a delay can be tolerated, deadlines can be missed.

- **MULTIMEDIA SYSTEMS**: A multimedia system is one that exhibits different media simultaneously. From the real-time point of view, we are interested in media that changes as a function of time, for instance, digital audio and video. These systems are generally considered soft systems as missing a deadline has no catastrophic consequences unless very high fidelity. Consider, for example, a digital video system. The system decodes some video streams and plays the frames at some rate (say, 30 frames per second). If it does not have enough time to finish decoding a frame, it is either skipped or only partially displayed. Not displaying a frame on time is considered a missed deadline. In this case, a missed deadline results in a lower quality of the service provided by the computer. If it is not known how missed deadlines are distributed over time, several missed deadlines may occur in a row, which will be manifested by a frozen screen for a short while.

1.5 ENERGY AWARE REAL-TIME SYSTEM

1.5.1 ENERGY MANAGEMENT IN REAL TIME SYSTEM

High performance is desired to satisfy peak computation requirement or meeting the timing constraints for real time applications. However, maintaining the peak performance all the time in a system may not be a wise decision since the computation requirement in a system generally has big variations and high performance generally implies high energy consumption. For example, the average workload for web server is only 10% to 20% of their peak workload. Thus, it is preferred to tune system performance according to run time computation requirement while lowering the system energy consumption. The rate of energy consumption (power) has increased over the years. Thus, efficient energy management is required to extend the operation time of battery operated devices (e.g., cell phones, PDAs and solar explorers) or to reduce the operation cost of energy hungry systems (e.g., server farms) and increase their reliabilities. In order to dynamically manage the power consumption in a computing system, dynamic power management techniques are used as described in the following section.

Energy conservation has come to be recognized as a critical issue in design of pervasive real-time embedded systems, particularly due to the proliferation of mobile systems with limited

power resources. For the sake of mobility, these portable computing and communication devices require low energy consumption to maximize the battery lifetime. As VLSI technology continues its remarkable advances, the power consumption has been increased exponentially. Current battery technology, with its 5% annual capacity increase [2], cannot effectively address this problem. In addition, serious concerns are raised with regards to managing the heat dissipation from the rapidly elevated power consumption. Left unchecked, the power consumption will threaten to curtail the availability of the future high performance portable devices and advanced multimedia functionality on these devices. Power aware scheduling has been proven to be an effectively way to reduce the power consumption. Rooted in the traditional real-time scheduling technology, the power aware scheduling techniques change the system computing performance accordingly based on the dynamically varied computation demand. Two main types of power management mechanisms are reported in the literature.

1.5.2 DYNAMIC ENERGY MANAGEMENT:

Rate of energy consumption is proportional to supply voltage and frequency, i.e.,$P \propto V_{dd}^2 f$, where V_{dd} and f are the operating voltage and frequency. Hence, to minimize the dynamic power (rate of energy) consumption V_{dd} or f should be reduced. The dynamic power consumption scales up square times with the operating voltage hence, a small reduction in the voltage will be able to reduce the dynamic power consumption drastically. In addition, the processing frequency (speed) also has an almost linear relation with operating voltage. Therefore, reduction in the speed would reduce the power consumption cubically.

Thus, the major focus of dynamic energy management lies on the techniques to scale down the speed of the components is termed as dynamic voltage scaling (DVS). However, lowering the speed causes increased delay of CMOS circuit and requires the system to operate for longer duration. The more reduction in speed saves larger power consumption while elongating the execution time by large amount. The processor that supports switching between the voltage levels is termed as DVS processors. Consider for example, a DVS processor that can operate at two speed levels s_1 and s_2 with their respective power consumption as p_1 and p_2. In case s_1 is greater thans_2, the power consumption p_1 will be

more than p_2.In such system lowering the operating speed always saves energy and scheduling algorithms addressing such systems are referred to be processor energy aware.

Table 1.1 Researches in Energy Management

Author	DVS Technique	DPD Technique	Processor/System Energy
Weiser	Yes	No	Processor
Lee & Sakurai	Yes	No	Processor
Shin & Choi.	Yes	No	Processor
Kirovski & Potkonjak	Yes	No	Processor
Niu & Quan.	Yes	No	Processor
Aydin et. al.	Yes	No	System
Jejurikar & Gupta	Yes	No	System
Niu & Quan	No	Yes	System
Niu & Quan.	Yes	Yes	System
Kim & Roy	Yes	No	Processor
Present work	Yes	Yes	System

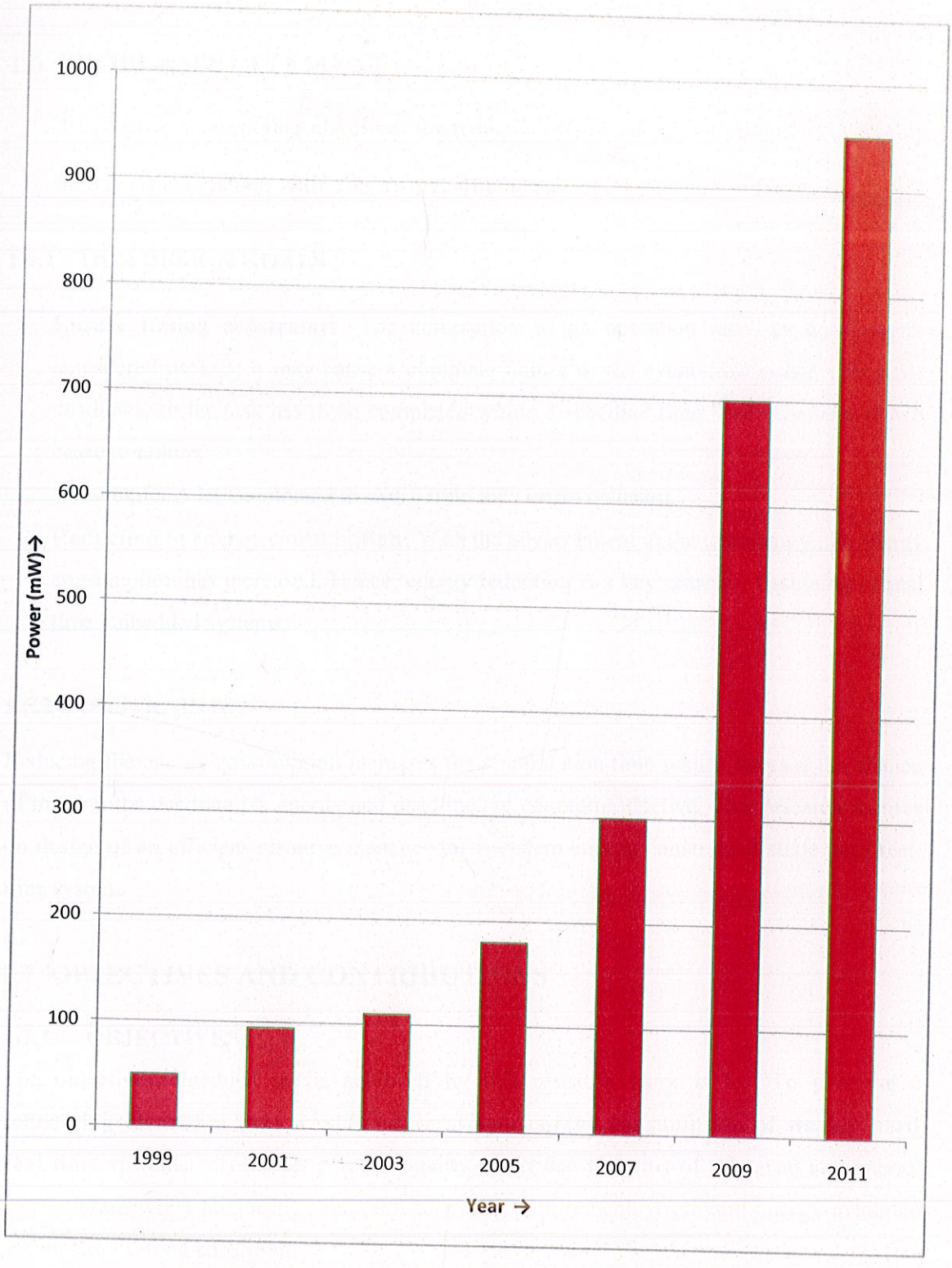


Figure: 1.5 Pie chart showing the saving of the Energy

1.6 PROBLEM STATEMENT

To propose a scheduling algorithm for reduction of the energy consumption in weakly hard real time systems with conservative deadline.

1.6.1 OUR DESIGN GOALS

- **Ensure timing constraints**: The completion of an operation after its deadlines is considered useless; it may cause a complete failure or the events that occur with strict deadlines, so the task has to be completed within a specified time limit otherwise it will cease to failure.

  Example**:** A late command to stop a train may cause collision.
- **Reduction in energy consumption:** With the advancement in the technology, the energy consumption has increased. Hence, energy reduction is a key issue for designing of real time embedded systems.

1.6.2 CONSTRAINTS

Reducing the energy consumption increases the computation time which increase the chance of missing the deadline i.e. energy and deadline are counterproductive. This research focuses on design of an efficient resource manager for a system energy constrained static hard real-time system.

1.7 OBJECTIVES AND CONTRIBUTIONS

1.7.1 OBJECTIVES

The objectives, intended to be achieved by this research work is to "**To propose a scheduling algorithm that would reduce system energy consumption of weakly hard real time systems.**" To achieve better quality of service in terms of tolerance and energy consumption while honouring minimum tolerance requirement, maximum energy available within strict timing constraint.

1.7.2 CONTRIBUTIONS

There is a need for design of an efficient resource manager that minimizes system energy consumption while giving better tolerance to static hard real time system with arbitrary deadline exposed to transient faults. Thus, issues of energy and real time constraints can be integrated into single framework to achieve reduced system energy consumption within deadline by the use of check pointing, tolerance patterns, pre-emption control, speed fine tuning, delay start, speed patterns, criticality and sensitivity In this work, study has been carried out on periodic task sets with arbitrary deadlines to reduce energy consumption. An energy reduction technique has been employed for reduction in energy consumption of the system with conservative deadlines. It is further extended for arbitrary deadline with the use of speed pattern. Finally energy issues are combined resulting reliable energy aware scheduling technique.

1.8 ORGANIZATION OF THESIS

The work presented in this thesis has been arranged in seven chapters.

Chapter-1 Gives an introduction to real time systems, types of real time systems, Energy aware systems and it presents the problem statement, motivation and constraints.

Chapter-2 it gives an overview about the scheduling of the real time systems , various types of scheduling, detailed method of clock driven , and priority based scheduling .

Chapter-3 surveys the methodology consisting of system model, existing techniques and proposed technique also it studies the problem of energy aware scheduling for conservative deadline. It presents a scheduling algorithm that offers lesser system energy consumption for weakly hard real time systems consisting of voltage scalable (processor) and non scalable components (peripheral device), modelled with $(\mathbb{m}, \mathbb{k})$ constraint.

Chapter-4 it highlights about the proposed modification i.e. Inverse rm and also the calculation of energy, comparison between the energies of Inverse rate monotonic and rate monotonic.

Chapter-5 it gives an overview about the feasibility analysis of the real time systems, whether the system is feasible or not, and limitations of the proposed modification.

Chapter -6 concludes the work and provides direction for future research, how to proceed further and what all can be done in the near future is explained in this section.

CHAPTER 2: REAL TIME SCHEDULING: AN OVERVIEW

In typical designs, a task has three states: 1) running (executing on the CPU), 2) ready (ready to be executed), 3) blocked (waiting for input/output). Most tasks are blocked or ready most of the time because generally only one task can run at a time per CPU. The number of items in the ready queue can greatly vary, depending on the number of tasks the system needs to perform and the type of scheduler that the system uses. On simpler non-preemptive but still multitasking systems, a task has to give up its time on the CPU to other tasks, which can cause the ready queue to have a greater number of overall tasks in the ready to be executed state. Whenever two processes with the same absolute priority are ready to run, the kernel has a decision to make, because only one can run at a time. If two processes are ready to run but have different absolute priorities, the decision is much simpler, and is described in Absolute Priority. Each process has a scheduling policy. For processes with absolute priority other than zero, there are two available: The most sensible case is where all the processes with a certain absolute priority have the same scheduling policy.

2.1 CLOCK DRIVEN SCHEDULING

Clock driven schedulers make their scheduling decisions regarding which task to run next only at the clock interrupt points. Clock driven schedulers are those for which the scheduling points are determined by timer interrupts. Clock driven schedulers are also called off line schedulers because these schedulers fix the schedule before the system starts to run. That is, the scheduler pre- determines which task will run when. Therefore, these schedulers incur very little run time overhead. However, a prominent shortcoming of this class of schedulers is that they cannot satisfactorily handle aperiodic and sporadic tasks since the exact time of occurrences of these tasks cannot be predicted. For this reason, this type of schedulers is also called a static scheduler.

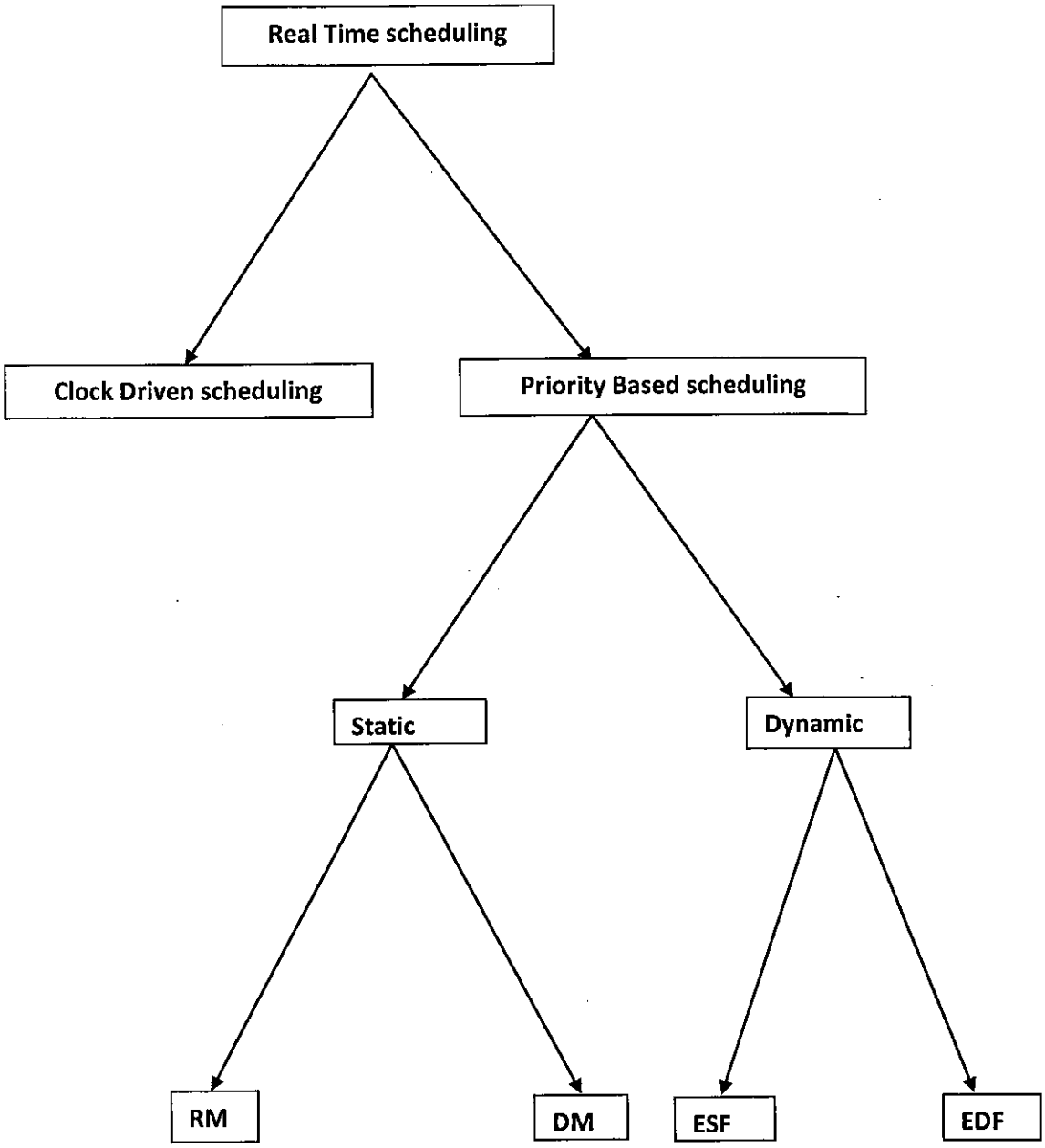


Figure 2.1 Types of Real Time Scheduling

2.2 PRIORITY DRIVEN SCHEDULING

It is based on the priority of the jobs. When the higher priority is ready it pre-empts the lower priority which is running. There are two types:-

2.2.1 STATIC APPROACH: Priorities are assigned to tasks once for all and every job of a task will have same priority

2.2.1.1 RATE MONOTONIC ALGORITHM (RM)

Rate monotonic algorithm is a dynamic pre-emptive algorithm based on static priorities. The rate monotonic algorithm assigns static priorities based on task periods. Here task period is the time after which the tasks repeat and inverse of period is task arrival rate. For example, a task with a period of 10ms repeats itself after every 10ms. The task with the shortest period gets the highest priority, and the task with the longest period gets the lowest static priority. At run time, the dispatcher selects the task with the highest priority for execution. According to RMA a set of periodic, independent task can be scheduled to meet their deadlines, if the sum of their utilization factors of the n tasks is given as below.

$$\sum\left(\frac{C_i}{T_i}\right) \leq U(n) = n\,(2^{1/t} - 1)$$

C_i =worst – case task execution time of task

T_i =period of $task_i$

U (n) = Utilization bound for n tasks

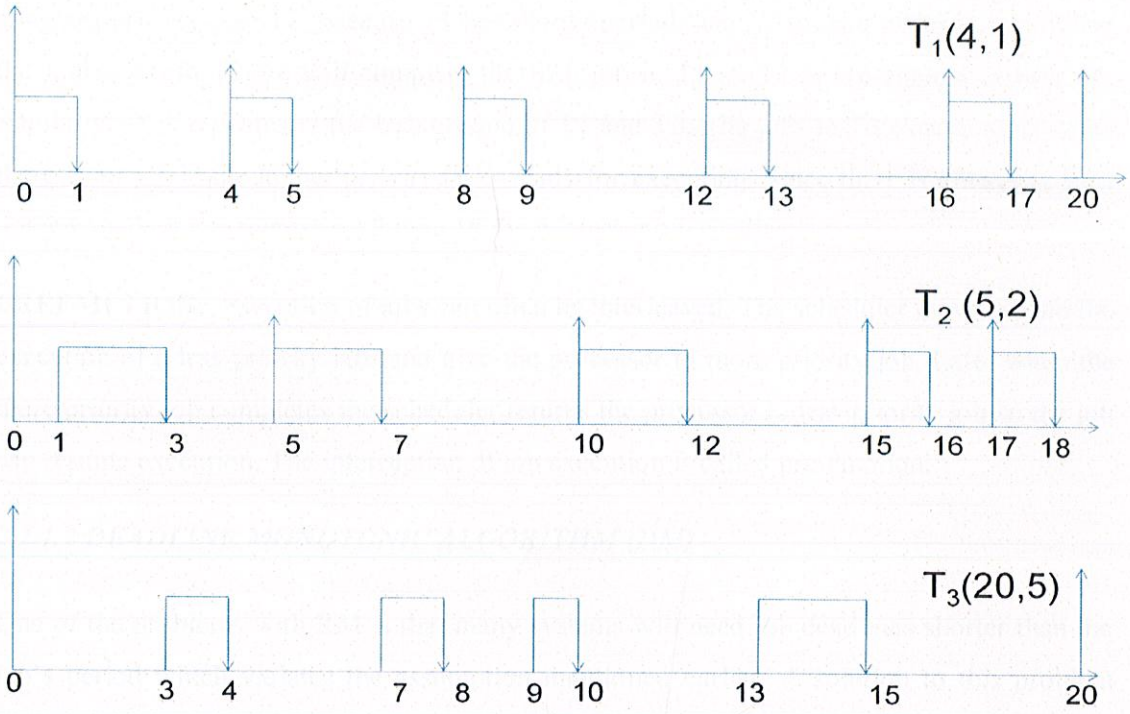


Figure 2.2 Scheduling for Rate monotonic

In this example we have three tasks T_1, T_2, T_3

T_1 (4, 1)

T_2 (5, 2)

T_3 (20, 5)

Where 4, 5, 20 are the period and 1, 2, 5 are the execution time of the task. It has a hyper period 18 i.e. The Total period of the task this is calculated by taking LCM of the periods.

First we will release T1 because it has shorter period (in rate monotonic algorithm shorter period has higher priority) after execution of first job of T1 we will release T2 who has next shorter period and its jobs execute in the background of T1 so for this reason the execution of the first job in T2 is delayed until the first job of T1 completes and the fourth job in T2 is pre-empted at the time 16 when the fifth job in T1 is released because here the job in T1 has

a highest priority than T2 because T1 has shorter period than T2 so here instead of complete the fourth job in T2 we will complete the fifth job in T1 so T1 is pre-empted at time 16. Similarly T3 is executes in the background of T1 and T2. The jobs in T3 execute only when there is no job in the higher priority tasks ready for execution. Since there is always at least one job ready for execution until time 18 the processor never idles.

PREEMPTION: execution of jobs can often be interleaved. The scheduler may suspend the execution of a less priority jabs and give the processor to more priority job. Later when the more priority job completes the scheduler returns the processor to less priority job so the job can resume execution. The interruption of job execution is called pre-emption.

2.2.1.2 DEADLINE MONOTONIC ALGORITHM (DM)

One of the problems with RM is that many systems will need job deadlines shorter than the job's period which violates the assumption mentioned earlier. A solution to this problem arrived in 1982 with the introduction of the Deadline Monotonic (DM) algorithm. With DM, a job's priority is inversely proportional to its relative deadline. That is to say, the shorter the relative deadline, the higher the priority.

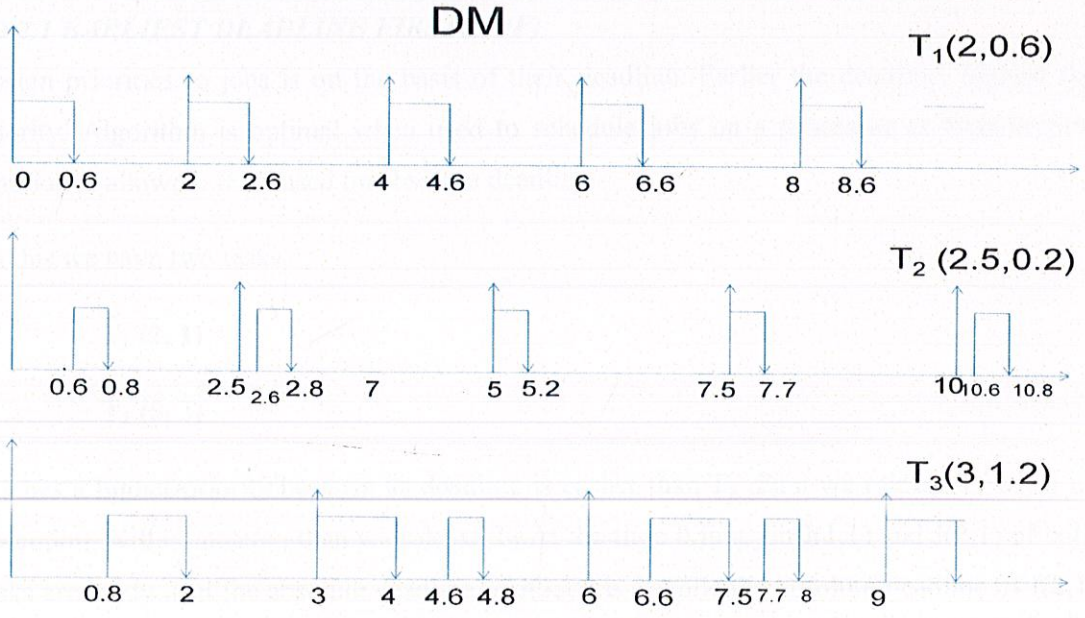
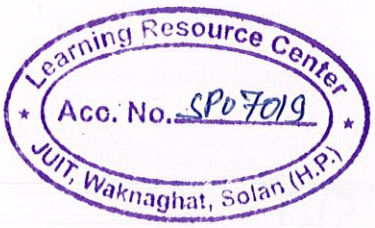


Figure 2.3 Scheduling of Deadline Monotonic

In this example we have three tasks T_1, T_2, T_3



T_1 (2, 0.6)

T_2 (2.5, 0.2)

T_3 (3, 1.2)

Where 2, 2.5, 3 are the period which is same as the deadline of the task and 0.6, 0.2, 1.2 is the execution time of the task. It has a hyper period 10 i.e. Total period of the task which is calculated by taking LCM of the periods. In this T1 has the higher priority because it has a shorter deadline, so first we release T1, when its execution time completes then we release T2 who has next higher priority. It job executes at the background of T1.So for this reason the execution of the first job in T_2 is delayed until the first of T_1 completes. The 2^{nd} job in T_3 is pre-empted at time 4, when the 3^{rd} job in T_1 is released because it has higher priority than T_3. Similarly the 3^{rd} job in T_3 is pre-empted at time 7.5.Since there is always at least one job ready for execution until hyper period, the processor never get idle until that time.

2.2.2 DYNAMIC APPROACH: The priorities of tasks may change from request to request; different jobs will have different priorities.

2.2.2.1 EARLIEST DEADLINE FIRST(EDF)

Assign priorities to jobs is on the basis of their deadline. Earlier the deadline, highest the priority. Algorithm is optimal when used to schedule jobs on a processor as long as pre-emption is allowed. It is based on absolute deadline.

In this we have two tasks

T_1 (2, 1)

T_2 (5, 3)

T1 has a higher priority because its deadline is earlier than T_2 .First we release T_1 when its execution will complete , then we release T_2.At the time 0,first job J(1,1) and J(2,1) of both tasks are ready .But the absolute deadline of J(1,1,) is 2 while the absolute deadline of J(2,1) is 5 .So consequently J(1,1).

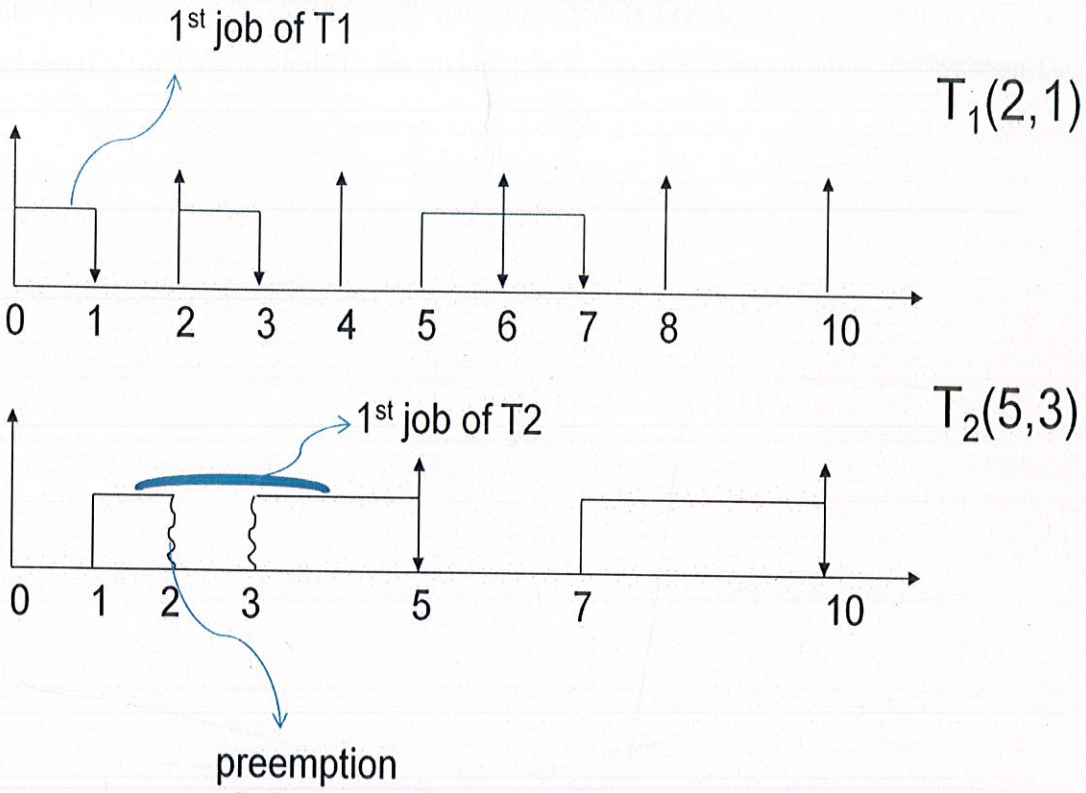


Figure 2.4 Scheduling of Earliest Deadline First

In this we have two tasks

T_1 (2, 1)

T_2 (5, 3)

T1 has a higher priority because its deadline is earlier than T_2 .First we release T_1 when its execution will complete , then we release T_2.At the time 0,first job J(1,1) and J(2,1) of both tasks are ready .But the absolute deadline of J(1,1,) is 2 while the absolute deadline of J(2,1) is 5 .So consequently J(1,1) has a higher priority and executes .When J(1,1) completes J(2,1) begins to execute. At time 2 the 2nd job T_1 is released and its deadline is 4, earlier than the deadline of job 1 of T_2. Here 2nd job of T_1 pre-empts the first job of T_2 and execute. When job

2^{nd} of T_1 completes , the processor ten executes Job 1 of T_2 .At time 4 job 3 of T_1 is released its deadline is 6 which is later than the deadline of job 1 of T_2 , hence processor continue to execute the first job of T_2. At time 5 T_2 has completed its first job now the deadline of T_2 is 10 and deadline of T_1 is 6. Here T_1 has higher priority because it has deadline earlier than T_2 so this way jobs continue to execute in tasks T_1 and T_2.

CHAPTER 3: METHODOLOGY

There are existing techniques such as DVS and DPD. The DVS is the most common and effective approach in low-power embedded system design. The basic idea is to apply low supply voltage on tasks to utilize the system idle time. In this work, we present an approach that reduces the configuration energy by applying the DVS on the reconfiguration process. DPD is a switching to sleep mode or low power mode of unused component of a system.

3.1 SYSTEM ENERGY MODEL

This chapter aims to minimize the system energy consumption for a system having independent periodic task set $T = \{\tau_1, \tau_2, \tau_3 \dots \tau_n\}$ that guarantee minimum QoS defined by $(\mathbb{m}, \mathbb{k})$. The priority of a job is assigned based on the earliest deadline first policy. The system consists of two types of components namely, processor (termed as frequency dependent component) and peripheral devices (termed as frequency independent component). While formulating the energy model following considerations are taken into account:

- The frequency independent components are represented by set $A = \{a_1, a_2, a_3 \dots a_N\}$ where a_i represents a memory or peripheral device. The power management policies reported in uses only two states (active and sleeping) for a frequency independent component with no recourse conflicts. Same consideration is taken in this work.

- The frequency dependent components (DVS processor) can operate at $\mathcal{N} + 1$ discrete voltage levels, i.e., $V = \{v_{slp}, v_1, v_2, v_3 \dots v_{\mathcal{N}}\}$ where a voltage level v_i is associated with its corresponding speed s_i from the set $S = \{s_{slp}, s_1, s_2, s_3 \dots s_{\mathcal{N}}\}$.The speed s_1 is the lowest operating speed level measure at voltage v_1 whereas maximum speed $s_{\mathcal{N}}$ at the voltage level $v_{\mathcal{N}}$. A processor can lie in one of the three possible states namely active, idle and sleep. In the active state the processor can run at any of the speed levels between s_1 to$s_{\mathcal{N}}$, while in the idle state and sleep state it will function at speed s_1 and s_{slp} respectively.

Each task $\tau_i \in T$ has attributes $< e_i(s_j),\ p_i,\ d_i,\ \mathbb{m}_i, \mathbb{k}_i >$ where p_i and d_i are the period and relative deadline of the task respectively. The computation time at speed s_j is $e_i(s_j) = \mathbb{e}_{p,i}/s_j + \mathbb{e}_{d,i}$ where$\mathbb{e}_{p,i}$ and $\mathbb{e}_{d,i}$ are the computation requirement for frequency dependent component at the speed s_j and independent component respectively. Beside these temporal characteristics minimum QoS requirement is represented by a pair of integers$(\mathbb{m}_i, \mathbb{k}_i)$, such that out of $\mathbb{k}_i$ consecutive releases (jobs) of τ_i at least $\mathbb{m}_i$ releases must meet their deadline. The tasks relative deadline is $d_i \leq p_i$termed as conservative deadline are used and we consider same constraint in this chapter.

3.2 TERMS USED

In the following section some terms which we use are described.

- **Job**: - Unit of work that is scheduled and executed by the system.
- **Task: -** Set of relative jobs which acting to consume resources and producing one or more results. The jth instance of a task Ti would be denoted as Ti (j).
- **Release time of a job: -** The response time of a job τ_i^j at speed s is the sum of its own time requirement and its higher priority jobs pre-empting it.
- **Execution time of a task**: - The amount of time required completing the execution of any job.
- **Deadline: -** The instant of time at which the task must complete its execution.
- **Relative deadline**: - The maximum allowable response time of the job. (1 hour)
- **Absolute deadline**: - It is equal to interval of time between release times and actual instant at which deadline occurs. (Till 5:00 p.m.)
- **MK_hyperperiod:**- It can be defined the point after which all the task in the set are in phase and $(\mathbb{m}, \mathbb{k})$ pattern as for each task is restarted i.e., the situation at time $t = 0$ is restored. Mathematically, $L = LCM(\mathbb{k}_1 p_1, \mathbb{k}_2 p_2, \mathbb{k}_3 p_3 \ldots \mathbb{k}_n p_n)$ where, LCM stands for least common multiple.
- **Critical speed of the task: -** Speed at which system energy requirement is least for a task is called the critical speed. Each task in the system has its own critical speed because its computation demand and set of associated components may differ.

3.3 REAL-TIME SCHEDULING WITH (m, k) DEADLINE

To schedule a real-time task set with (m, k)-firm deadline involves two sub-problems: (i) mandatory/optional partitioning problem, i.e., to determine if a job should be mandatory or optional, and (ii) scheduling problem, i.e., to schedule these jobs properly to guarantee their deadlines. As proven in, both problems are NP-hard problems. Some related real-time scheduling results for (m, k)-firm guarantee. For ease of our explanation, we use *patterns* to denote the mandatory/optional partitions. A pattern is an infinite binary sequence associated with each task such that a job is mandatory if its corresponding bit is "1" and optional otherwise.

The mandatory/optional partition decision can be made off-line or on-line. Two known static mandatory/optional partitioning strategies are reported in literature. The first one is called *the deeply-red pattern* or *R-pattern.* The (*m*, *k*)-pattern defined with formula (2) has the property that mandatory jobs are marked evenly, and is therefore referred as the *evenly distributed pattern* (or *E-pattern*).

The most significant advantage of applying static patterns is that they enable the application of theoretic real-time techniques to analyze system feasibility. Analytical schedulabilty results are available for both fixed-priority and EDF scheduling policies, based on either R-pattern or E-pattern. The problem, however, is its poor adaptively in dealing with the run-time variations, which is inherent in many real-time applications. Dynamic mandatory/optional partitioning, on the other hand, is more flexible and therefore can accommodate run-time variations more effectively. The problem is how to ensure the deadlines of all the mandatory jobs. A number of dynamic mandatory/optional partitioning heuristics are proposed with no guarantee for the deadlines of mandatory jobs at all.

Currently, two dynamic techniques published can ensure the (m, k)-guarantee. proposed a Bi-Modal Scheduler, which runs jobs at two modes: normal mode and panic mode. A task is first executed at the normal mode and promoted to the panic mode if the dynamic failure will occur if it stays in the normal mode to shift the E-pattern dynamically when an optional job meets its deadline.

3.4 EXISTING TECHNIQUES USED

3.4.1 DYNAMIC VOLTAGE SCALING (DVS)

It is based on adjusting the processor voltage and frequency on the fly. Power requirement depends on operating frequency as well as voltage i.e. the dynamic processor power is strictly increasing convex function of the processor speed. The DVS reduce the processor speed to the extent it is possible to obtain higher energy saving. The speed of a frequency dependent component is said to be reduced if it is either operating at lower voltage or frequency. The task response time increases with the reduced processor speed leading of the following consequences:

- A release may miss its deadline when it is feasible at higher speed.
- The longer execution time will be able to decrease the dynamic energy consumption of the processor.
- Frequency is dependent component remains active for longer time and increase energy consumption.
- Longer execution time implies more losses in energy due to leakage current.

Power requirements are one of the most critical constraints in mobile computing applications, limiting devices through restricted power dissipation, shortened battery life, or increased size and weight.

The design of portable or mobile computing devices involves a trade off between these characteristics. For example, given a fixed size or weight for a handheld computation device/platform, one could design a system using a low-speed, low-power processor that provides long battery life, but poor performance, or a system with a (literally) more powerful processor that can handle all computational

loads, but requires frequent battery recharging. This simply reflects the cost of increasing performance. For a given technology, the faster the processor, the higher the energy costs (both overall and per unit of computation). It focuses on the energy consumption of the processor in a portable computation device for two main reasons. First, the practical size and weight of the device are generally fixed, so for a given battery technology, the available

energy is also fixed. This means that only power consumption affects the battery life of the device. Secondly, we focus particularly on the processor because in most applications, the processor is the most energy-consuming component of the system. This is definitely true on small handheld devices like PDAs [3], which have very few components, but also on large laptop computers that have many components including large displays with backlighting. As a result, the design problem generally boils down to a trade off between the computational power of the processor and the system's battery life. One can avoid this problem by taking advantage of a feature very common in most computing applications: the average computational throughput is often much lower than the peak computational capacity needed for adequate performance.

3.4.2 DYNAMIC POWER DOWN (DPD)

The dynamic power down technique suggest to switching to sleep state (low energy consuming state) of the idle components to reduce the energy consumption. For switching from active state to sleep state and back from sleep state to active state will incur an overhead called the DPD overhead. Thus, switching to sleep state too often may be counterproductive and estimated a threshold value for shutting down the components by comparing the energy consumption required in idle state with energy consumption on power down state and wakeup. If the energy consumption for switching to sleep state is less than or equal to the energy consumption in idle time then switching to sleep state is preferred over leaving the component idle. Suppose P_{idle} is the power consumption per unit time when the processor is idle for t time period and would consume E_{shut} energy when it is shut down and wakeup for the same period then if $(P_{idle} * t) \geq E_{shut}$ then this shutdown will lead to a positive energy gain. DPD has been widely adopted in real time scheduling. A majority of DPD techniques have been proposed for soft real-time systems, where task deadlines can be missed albeit with reduced quality levels. The multimedia applications are judiciously scheduling the real time tasks and switching to sleep state the processor (employ dynamic power down DPD approach) reduces the energy consumption while guaranteeing the QoS requirements for weakly hard real time system.

Table 3.1 DVS Processors Specifications

DVS Processor	CPU Frequency(MHz)	Voltage (V)	CPU Power (maw)
Intel Scale PXA60	200	1.0	178
	300	1.1	283
	400	1.3	411
Transmeta Crusoe]	300	1.2	130
	400	1.225	190
	533	1.35	300
	600	1.5	420
	667	1.6	530
IBM PowerPC 405LP	33	1.0	40
	100	1.0	120
	133	1.3	280
	200	1.5	630
	266	1.8	1000

3.5 PARTITIONING TECHNIQUES

Selection of m jobs from a window k consecutive jobs for execution .Referred to as mandatory and represented by 1.

3.5.1 DEEPLY RED PATTERN (RED PATTERN)

This pattern was proposed by Koren & Shasha where first '$\mathbb{m}_i$' releases out of '$\mathbb{k}_i$' consecutive releases of task τ_i are mandatory. Mathematically, this can be described as

$$\pi_i^j = \begin{cases} 1, & 0 \leq j \bmod \mathbb{k}_i < \mathbb{m}_i \\ 0, & otherwise \end{cases} \qquad j = 0, 1, \ldots . \mathbb{k}_i - 1$$

When π_i^j is 1, release τ_i^j is mandatory while it is optional in case 0 is assigned toπ_i^j. We refer this pattern as Red_Pattern. Advantage of applying this pattern to a task set for energy minimization is that it aligns the optional jobs together so that a component has a better opportunity to switch into sleep state to save energy. For a task whose critical speed is higher than or equal to the highest possible speed ($s_{\mathcal{N}}$) the operating speed should never be scaled down. Assigning Red_Pattern to such a task helps to extend the idle interval for switching to sleep state. However, for a task whose critical speed is lower than $s_{\mathcal{N}}$ Red_Pattern overloads the system leading to large size busy intervals and need more energy to be feasible.

3.5.2 EVENLY DISTRIBUTED PATTERN (EVEN PATTERN)

In this evenly distributed pattern in which the first release is always mandatory and the distribution of mandatory and optional is even i.e., alternating. Mathematically, this can be described as

$$\pi_i^j = \begin{cases} 1, & if\ j = \left\lfloor \left\lceil \frac{j * \mathbb{m}_i}{\mathbb{k}_i} \right\rceil * \frac{\mathbb{k}_i}{\mathbb{m}_i} \right\rfloor \\ 0, & otherwise \end{cases} \quad for\ j = 0, 1, \ldots \mathbb{k}_i - 1$$

We refer it to as Even_Pattern.

3.5.3 REVERSE EVENLY DISTRIBUTED PATTERN (REV_PATTERN)

This pattern is a reverse of the Even_Pattern, hence the first release is always optional and the distribution of mandatory and optional is alternating. Mathematically:

$$\pi_i^j = \begin{cases} 0, & if\ j = \left\lfloor \left\lceil \frac{j*(\mathbb{k}_i - \mathbb{m}_i)}{\mathbb{k}_i} \right\rceil * \frac{\mathbb{k}_i}{(\mathbb{k}_i - \mathbb{m}_i)} \right\rfloor \\ 1, & otherwise \end{cases} \quad j = 0, 1, \ldots \mathbb{k}_i - 1$$

We refer it as Rev_Pattern.

3.5.4 HYBRID PATTERN (HYD_PATTERN)

In which instead of assigning same pattern to all the tasks in the task set, they assigned different type of patterns (Red_Pattern or Even_Pattern) to each task. For example, task τ_1 is partitioned into mandatory and optional according to Red_Pattern while τ_2 and τ_3 could be assigned Red_Pattern or Even_Pattern. Thus, yielding 2^n possible combination of pattern assignment where n is the number of the tasks in the task set.

3.5.5 MIXED PATTERN (MIX_PATTERN)

The hybrid pattern allows a task in the task set to be scheduled by Red_Pattern or Even_Pattern. In both cases at least the first release of each task is mandatory (if not more e.g., $(\mathbb{m}, \mathbb{k}) = \{(3,5),\ (4,7)\}$ first two releases of both the task are mandatory with the Hyd_Pattern) and are in phase hence, will overload the system, forcing it to be feasible with high energy requirement. Therefore, to improve the performance of Hyd_Pattern suggested a mixed pattern (Mix_Pattern) which combines the Hyd_Pattern with the Rev_Pattern yielding 3^n possible combination of pattern assignment. By including the Rev_Pattern the Mix_Pattern would give fairer chance to a task for executing at lower speed (the second release of both the task in the above example would be mandatory while the first may or may not be so. Since the second release of a task would usually be out of phase with the other releases and will not overload the system as hybrid pattern does). Thus, Mix_Pattern is the superset of all the above suggested patterns. This work uses Mix_Pattern.

CHAPTER 4: PROPOSED MODIFICATION: INVERSE RM

Our main focus is to reduce the energy consumption of a real time system and to enhance the performance of the system. Reduction in energy consumption increases the computation time which therefore increases the chance of missing the deadline i.e. energy and deadline are counterproductive. So, we have introduced an approach which can effectively reduce the energy by 15% along with DVS strategy which provide significant energy savings while maintaining real time deadline guarantees. It is inverse of rate monotonic scheduling where shorter the period higher is the priority but in case of inverse RM higher is the period, higher is the priority of the task set. It includes peripheral devices and energy is calculated using the existing techniques such as DVS and DPD. Let's observe the scheduling and energy consumption of task set T using Inverse RM:

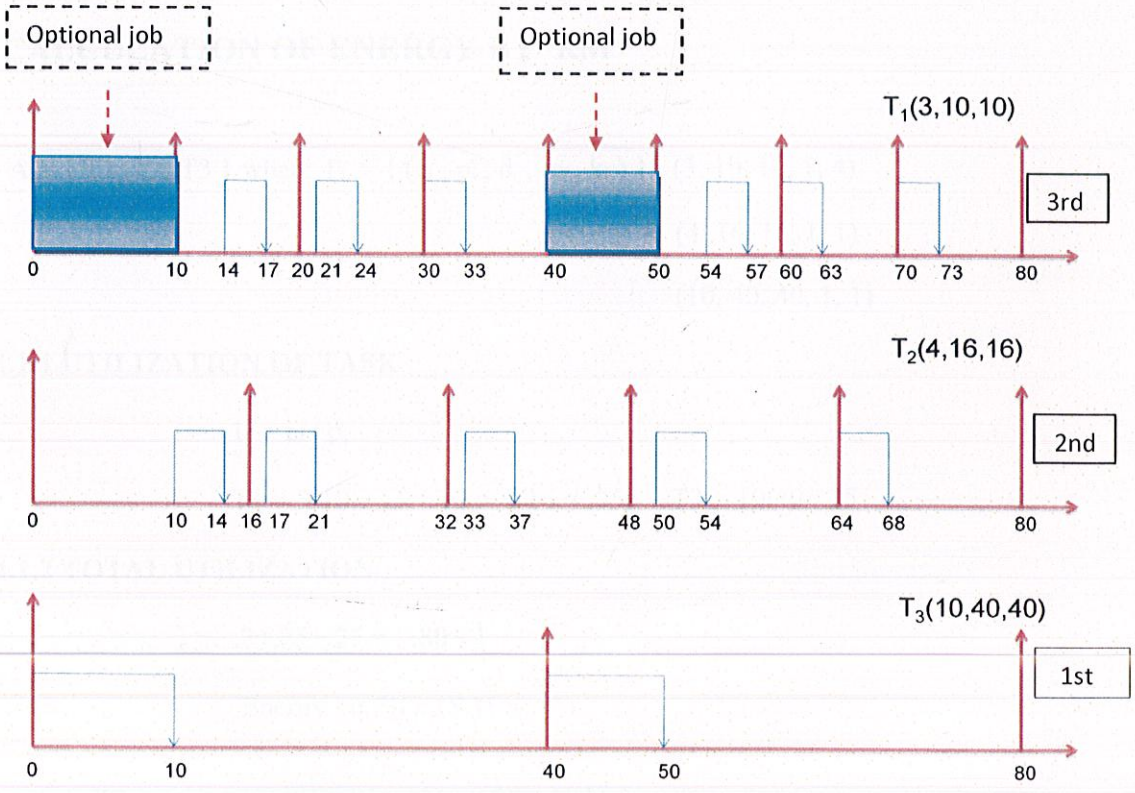


Figure 4.1 Scheduling of Inverse RM

Now above is an instance where we assign priorities based on their periods. Higher is the period; higher is the priority of the task set. In this example we have three task set T1 (3, 10, 10 , 1,4) , T2 (4, 16, 16, 1, 1) , T3(10 , 40, 40 , 1 ,1) which includes execution time e , period p, deadline d , m jobs which can miss deadline , k consecutive jobs. Here 3, 4, 10 are the execution time of the task T1 ,T2, T3 and 10, 16, 40 are the period and deadline of the tasks T1, T2, T3. It has the hyper period 80 i.e. total period of the task which is calculated by taking the L.C.M. of the periods. As release is available at time 0, three of the tasks are available but as T3 has the higher period so it will have the highest priority whereas T2 will have the 2nd priority and T1 had the 3rd priority. Now T3 will take 10 units of time, the first job of the task T3 is executed then T2 will start executing, it will execute till the next release of job. After the execution of task T2, T1 will execute its 2nd job as 1st job is set to be optional by using m-k model where m jobs can miss their deadlines accordingly. Since, 1st and the 4th release is made optional of the task T1, will execute 3 units of every job. It will reduce the energy consumption of the system.

4.1 CALCULATION OF ENERGY BY RM

$A = \{ T1, T2, T3 \}$ where $T_i = \{ (e_i , p_i , d_i , m_i , k_i) \}$: (3, 10, 10, 1, 4)

(4, 16, 16, 1, 1)

(10, 40, 40, 1, 1)

4.1.1 UTILIZATION OF TASK

$U = ei / p_i$

$T1 = 3/10 = .3$; $T2 = 4/16 = .25$; $T3 = 10/40 = .25$

4.1.2 TOTAL UTILIZATION

$.3+.25+.25 = .80 < 1$

$Energy = e_i * [(\gamma_p(S_i) + \gamma_d]*n$

Where e_i = execution time of the task

S_i = operating speed

γ_p = speed of the processor

γ_d = speed of the device

n = no. of jobs

Power = Energy/ unit time: $e\gamma$

Frequency scalable (Processor): $\gamma_p \alpha S^3$ where s : operating speed

Dynamic voltage scaling DVS: $\gamma_p(Si) \alpha Si^3$

Energy $= e_i * [(\gamma_p(S_i) + \gamma_d] * n$

T1 = 3*[(10)^3+50]*6 = 18900

T2 = 4*[(10)^3 +100)*4] + [(10^3) *4 + (55-48)*100] = 74304

T3 = 10*[(10)^3*2+[(60-43)+(27-7)]]*150=25550

Total Energy = T1+T2+T3 = 118754

4.2 CALCULATION OF ENERGY BY INVERSE RM

A = { T1, T2, T3 } where T_i = { (e_i , p_i , d_i , m_i , k_i) } : (3, 10, 10, 1, 4)

(4, 16, 16, 1, 1)

(10, 40, 40, 1, 1)

4.2.1 UTILIZATION OF THE TASK

$U = e_i / p_i$

T1 = 3/10 = .3;

T2 = 4/16 = .25;

T3 = 10/40=.25

4.2.2 TOTAL UTILIZATION

.3+.25+.25 = .80 <1

Energy $= e_i * [(\gamma_p(S_i) + \gamma_d] * n$

Where

e_i = execution time of the task

S_i = operating speed

γ_p = speed of the processor

γ_d = speed of the device

n = no. of jobs

Power = Energy/ unit time: $e\gamma$

Frequency scalable (Processor): $\gamma_p \alpha S^3$

Where s: operating speed

Dynamic voltage scaling DVS: $\gamma_p(Si) \alpha Si^3$

Energy $= e_i * [(\gamma_p(S_i) + \gamma_d] * n$

T1 = 3*[(10)^3+50]*6 = 18900

T2 = 4*[(10)^3+100]*5 = 22000

T3 = 10*[(10)^3+150]*2 = 23000

Total Energy = T1+T2+T3 = 63900

By implementing inverse RM, energy is reduced by 37.7%

Hence proved algorithm is feasible.

The reduction in energy consumption of the system can be calculated. The calculation of energy reduction by RATE MONOTONIC and INVERSE RATE MONOTONIC is as follows: In this we have taken three tasks { T_1, T_2, T_3 },with the respective execution time,period and deadline Ti = {(e_i , p_i , d_i) } also here (m_i , k_i)refers in this m number of jobs can be missed out of k consecutive total jobs.

RATE MONOTONIC SCHEDULING:

In the RM, utilization of the task can be calculated as execution time of any job per the period of that very job for a particular job that is occurring (U= e_i / p_i) .Similarly we calculate the utilization for all the three tasks. The summation of the three tasks should be less than 1, for a job or a task to be feasible. Thereby we calculate the energy of the task by Energy =ei *[(γ_p(S_i)+ γ_d]*n

where, e_i = execution time of the task

Si = operating speed

γ_p = speed of the processor

γ_d = speed of the device

n = no. of jobs

Also γ_p α S_3 where s: operating speed so therefore γp (Si) α S_i^3.Therefore the energy calculated for RM: Total Energy = T1+T2+T3 = 118754

Similarly for the calculation for **INVERSE RM SCHEDULING:**

The utilization of the task is calculated here in the same way as done in RM.Therfore the calculation for the energy reduction of INVERSE RM is calculated to be 63900.

So by implementing the inverse RM the percentage reduction of energy can be calculated as Energy calculated by RM - Energy calculated by Inverse RM /Total energy calculated by inverse RM which is calculated and the output is 37.7% reduction in energy consumption.

CHAPTER 5: FEASIBILITY ANALYSIS AND LIMITATIONS

5.1 FEASIBILITY ANALYSIS

It is done in order to check whether the task set is feasible or not i.e. Can execute all the jobs residing in a task set T. A weakly-hard real-time system is schedulable if it can be guaranteed that at run-time all weakly-hard temporal constraints are always satisfied. Given a task set ordered by priorities, the feasibility analysis requires:

- Computing the worst-case μ-pattern of each task at its corresponding priority level.

- Checking if the μ-pattern satisfies the weakly-hard constraint.

Naturally, a task set can have tasks with either hard deadlines or weakly-hard constraints. All tasks must be tested with the response-time equation but only the infeasible ones with weakly-hard constraints require the two above steps. A function W(α, k) that returns true when the j^{th} task in the priority ordering α meets the weakly-hard constraints is described in . Function W (α, k) starts checking the strict feasibility of the j^{th} task according to the FPS feasibility test;

• If feasible, it will not miss any deadline and therefore it is also weakly-hard feasible;

• If infeasible and with a strict hard deadline, it is infeasible;

5.2 CALCULATION OF FEASIBILITY ANALYSIS

$$R_i^r = e_i + \sum_{n=1} (Rir^{-1} / P_h) * e_h$$

Where R_i^r = response time

e_i = execution time

P_h = higher period

e_h = higher execution time

For instance:

Task set:

T1 (3, 10, 10)

T2 (4, 16, 16)

T3 (10, 40, 40)

FOR TASK T3 = 10 It will execute for 10 units.

T2 = 4 + [4/40] *10 = 4+10 = 14

= 4 + [7/ 40] * 10 = 4 +10 = 14

T1 = 3 + [3/40] * 10 + [3 / 16] * 4 = 3+ 10 + 4 = 17

= 3 + [17/40] * 10 + [17 / 16] * 4 = 3+ 10 + 8 = 21

= 3 + [21/40] * 10 + [21 / 16] * 4 = 3+ 10 + 8 = 21

Therefore, the task set T1, T2, T3 is feasible in nature but it also has some limitations.

5.3 LIMITATIONS:

Every task i.e. feasible by RM may not be feasible by inverse RM.

CHAPTER 6: RESULTS AND CONCLUSION

In this, we present a dynamic scheduling algorithm to minimize the system wide energy consumption with (m, k)-guarantee. The system consists of a core processor a number of peripheral devices, which have different power characteristics. Energy consumption is critical in the design of pervasive real-time computing platforms. The power consumption for peripheral devices, as a significant part of the overall power consumption, must be taken into consideration to reduce the system wide power consumption. Along with the adopted single known mandatory/optional partitioning strategy, we propose to incorporate different partitioning strategies based on the power characteristics of the devices as well as the application specifications. We introduce a feasibility condition, and based on which, we propose an algorithm to performance the mandatory/optional job partitions. In this , we presented a dynamic DPD and DVS approach(calculation) to reduce the system wide energy consumption while guaranteeing the QoS requirement, which are modelled as the (m,k)-constraints. The energy saving performance of our approach comes from the facts that we dynamically change the mandatory/optional job settings, and merge the idle intervals effectively by delaying the execution for mandatory jobs. Our experimental results demonstrate that our approach can greatly reduce the number of idle intervals and thus the power consumption, while still providing (m, k)-firm guarantee. We also propose a novel pre-emption control scheme, which can be well incorporated into our dynamic scheduling algorithm. Extensive experiments have been performed and demonstrate the effectiveness of our approach.

Our results indicate that the energy-consumption of the real time systems along with the greedy algorithms when the system is significantly energy-constrained is reduced by 15% and enhance the performance .

6.1 FUTURE SCOPE OF THE WORK

In the future, we would like to expand this work beyond the deterministic/absolute real-time paradigm presented here. In particular, we will investigate other scheduling technique with probabilistic or statistical deadline guarantees. We will also explore integration with other energy conserving mechanisms.

Bibliography

Buttazzo, G. C. ,Rate Monotonic vs. EDF. Real-Time Systems, 9(1) , 5–26.
Chaeseok Im, S. H. (June 11–13,2004). Dynamic Voltage Scaling for Real-Time Multi-task. LCTES'04 (2005). .

E. D. Jensen, C. D., A Time-Driven Scheduling Model for Real-Time Operating Systems. In IEEE Real-Time Systems Symposium , pages 112-122 (1985).

G. Bernat, A. B., Weakly Hard Real-Time Systems. IEEE Transactions on Computers, 50(4) , 308–321 (2001).

J. A. Stankovic, R. R. Real-Time Operating Systems. Real-Time Systems,28(2-3),237–253(2004).

J. Xu, D. L., Priority Scheduling Versus Pre-Run-Time Scheduling. Real-Time Systems, 18(1) , 7-23(2000).

L. Niu, G. Q., Energy minimization for real-time systems with (m,k)-guarantee. IEEE Trans. on VLSI, Special Section on Hardware/Software Codesign and System Synthesis , 717–729 (July 2006).

Nasro MIN-ALLAH, Y.-J. W.-S., (Enhanced Rate Monotonic Time Demand Analysis. IJCSES International Journal of Computer Sciences and Engineering Systems, Vol.1, No.3 , 149-153 (July 2007).

Niu, L.,Energy-Aware Dual-Mode Voltage Scaling forWeakly Hard Real-Time Systems. SAC'10 , 321-322 (March 22-26, 2010).

S. Agrawal, R. S., A Preemption Control Technique for System Energy Minimization of Weakly hard real time systems. SNPD(2008).

Santhi Baskaran, P. T. ,Dynamic scheduling of skippable periodic tasks with energy efficiency in weakly hard real time system. International Journal of Computer Science & Information Technology (IJCSIT), Vol 2, No 6 , 100-110 (December 2010).

Smriti Agrawal, R. S. A Preemption Control Approach for Energy Aware Fault Tolerant Real Time System. International Journal of Recent Trends in Engineering, Vol. 1, No. 1 , 382-383 (May 2009).

Smriti Agrawal, R. S., (A Preemption Control Technique for System Energy Minimization of Weakly Hard Real-Time Systems. Studies in Computational Intelligence,Volume 149 , 201-215 2008.

Sprunt B,. Priority-driven preemptive I/O controllers for realtime systems. Proceddings of 15th International Symposium on Computer Architecture , 152-159 (1988).

T. A. AlEnawy, H. A. (Proceedings of the 16th EuroMicro Conference on Real-Time Systems (ECRTS'04)). Energy-Constrained Real-Time. June 2004.

P. R. Goossens1, Overview of Real-time Scheduling Problems, Universite Libre de Bruxelles, (2004).

J. P. Lehoczky, L. Sha, Y. Ding, The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior, in IEEE Real-Time System Symposium, pp.166-171 (1989).

P. Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 6, pp. 549 – 559, (June 1999).

S. Hua and G. Qu, Energy-Efficient Dual-Voltage Soft Real-Time System with (m, k)-Firm Deadline Guarantee, CASES'04, Washington, DC, USA, (Sep. 22–25, 2004).

R. Jejurikar and R. Gupta, Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems, ISLPED, (2004).

S. Agrawal, R. S. Yadav, Ranvijay, A Pre-emption Control Technique for System Energy Minimization of Weakly Hard Real-time Systems, SNPD (2008).

APPENDIX-A

THE CODE

CALCULATION FOR THE UTILIZATION AND FEASIBILITY OF A TASK

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
class task
{
 int cp, p, d, m, k, cd, Sa,e, rel_no;
 float res;
 public:
    void read_data(int);
    void accp(int);
    void RM_sa(int , int[]);
}
void main()
```

```
{
 task T[10];
 int n;
 int speed[5] = {5,10,15,25,30};
 cout<<"enter the no of tasks"<<endl;
 cin>>n;
 for(int i=0;i<n;i++)
 T[i].read_data(30);
 T->accp(n);
 T->RM_sa(n , speed);
}
void task :: read_data( int Smax)
{
 cout<<"\n enter the computation and devices speed"<<endl;
 cin>>cp>>cd;
 cout<<"\n enter the period of the task"<<endl;
 cin>>p;
 cout<<"\n enter the deadline of the task"<<endl;
 cin>>d;
 cout<<"\n enter the no of jobs which can miss deadline"<<endl;
 cin>>m;
```

```
 cout<<"\n enter the no of jobs"<<endl;
 cin>>k;
 Sa = Smax;
}
void task :: accp(int n)
{
 float u1=0, u2=0 ;
 for(int i=0;i<n;i++)
 u1= ((this[i].cp/ this[i].Sa)+ this[i].cd);
 u2+= (u1/this[i].p);
  if(u2 < 1)
    {
        cout<<" \n The task is acceptable : " <<endl ;
    }
    else
    cout<<"\n The task set is not acceptable"<<endl;
}
void task :: RM_sa(int n, int speed[])
{
  int e, i ,a , b ;
  int S = 0;
```

```
 int not_feasible = 1;
 while( not_feasible)
 {
  this[i].Sa= speed[S];
  not_feasible = this->accp(n);
  }
for(i=0;i<n;i++)
{
 a= (this[i].cp/ this[i]. Sa) + this[i].cd;
 b=a;
 for(int j=0;j<i;j++)
 {
  this[j].e = (this[j].cp/this[j].Sa)+ this[j].cd;
  b= b+ ceil(a/this[j].p)* this[j].e;
 }
 while ( a!= b)  {
  b=a;
  for(j=0;j<i;j++)
  {
  this[j].e = (this[j].cp/this[j].Sa)+ this[j].cd;
  b= e+ceil(b/this[j].p)* this[j].e;
```

```
  }
  if(this[i].res > this[i].d)
  {
   not_feasible ;
   not_feasible = this->accp(n);
  }
  else
  cout<<" the task set is feasible"<<endl;
}
}}
```

PRIORITY QUEUE

```
#include <iostream.h>
#include <stdlib.h>
class p_queue
{
int x;
p_queue * next;

public : p_queue* insertion (int);
p_queue* deletion (int * );
void display ( ) ;
p_queue::p_queue()
{
cout<<"\n constructor called ";
```

```
        x=-1;
        next=NULL;
        }
}
main()
        {
        p_queue *head ;
        head = new p_queue;
        int x;
        //cout<<"enter the value u want to insert "<<endl;
        head = head -> insertion(5);
        head -> display();
        head = head -> insertion(16);
        head = head -> insertion(1);
        head = head -> insertion(12);
        head = head -> insertion(18);
        head = head -> insertion(10);
        cout <<"deletion"<<endl;
        head = head -> deletion(& x);
        cout<<"\n deleted element is "<<x;
head -> display();
        }
        p_queue*  p_queue :: insertion (int y)
        {
        if(x==-1)
        {
        x=y;
        return this;
        }

        p_queue * temp;
```

```
        temp = new p_queue;
        temp -> x = y;
        p_queue * ptr = this;
cout<<"\n before if" <<ptr -> x;
        if (ptr -> x > y)
        {
        cout<<"\n in if part";
        temp -> next = this;
        cout<<temp;
        return temp;
        }
        else
        {
        while (ptr -> next! = NULL)
        {
        if (((ptr -> next) -> x < y))
        {
        cout<<"\n in else part "<< ((ptr -> next) -> x);
        ptr = ptr ->next;
        }
        else
        break;
        }

        temp -> next = ptr -> next;
        ptr -> next = temp;
        cout<< "\n new "<<this;
        }
        return this;
        }
        p_queue * p_queue :: deletion(int * data)
```

```
        {
        p_queue *temp = this -> next;
        * data = this -> x;
        return ( this -> next) ;
        }
void p_queue :: display ()
        {
        p_queue * temp = this;
        cout<<" \n priority queue is as follows :"<<endl ;
        while ( temp != NULL)
        {
        cout << temp -> x<<endl;
        temp=temp->next;
        }
}
```

SCHEDULING OF THE TASK

```
        #include<iostream.h>
        #include<conio.h>
        float task1( float ,int  ,int );
        float task2 (float , int , int);
        float task3 (float , int  , int);
        void main()
        {
         clrscr();
         float e1 , e2, e3;
         int p1 , p2, p3;
         int d1 , d2, d3;
         float u1 , u2, u3 , t , P ;
         int a , b, c ,d ,i, j , n1 , n2 , n3;
```

```
int sp , sd , S ,Ei, Er , ch ;
cout<<"ENTER THE EXECUTION TIME FOR ALL THE THREE TASKS :";
cin>> e1 >>e2 >>e3 ;
cout<<"\n ENTER THE PERIOD FOR ALL THE THREE TASKS:";
cin>> p1 >> p2 >> p3;
cout<<" \n ENTER THE DEADLINE FOR ALL THE THREE TASKS :";
cin>> d1 >> d2 >> d3;
u1 = task1( e1 , p1 , d1 );
cout<<"\n UTLIZATION FOR THE FIRST TASK IS : " << u1 ;
u2 = task2 ( e2 , p2 , d2);
cout<<"\n UTILIZATION FOR THE SECOND TASK IS:" << u2 ;
u3 = task3 ( e3 , p3 , d3 );
cout <<"\n UTILIZATION FOR THE THIRD TASK IS : " << u3 ;
t = u1 + u2 + u3 ;
if (t<= 1 )
{
 cout <<" \n TASKS ARE FEASIBLE ";
}
else
{
 cout <<" \n TASKS ARE NOT FEASIBLE" ;
}
cout<<"\n TOTAL UTILIZATION IS  " << t ;

a = 0;
b = 0;
while (a! = 1)
{
 b++;
 c = b*d3;
 if((c%d1 == 0)&&(c%d2 == 0))
```

```
{
 d=c;
 a++;
}
}
for(i=0 ; i<d; i++)
{
 a[i] =0 ;
}
for(i=0; i<d; i++)
{
if (a[i] == 0)
{
cout<<"\n i =" << i;
if (( i%d3==0)&&( i%d1==0)&&(i%d2==0))
{
 cout<<" \n COMBINE" ;
 cout<<i ;
 for (j = 0 ; j< e3 ; j++)
 {
  a[i] = 3;
  i++;
 }
  i-- ;
 }
 else
 if ((i%d1 == 0)&&(i%d2!=0)&&(i%d3==0))
 {
  for (j=0; j<e3;j++)
  {
  a[i] = 3;
```

```
i++;
}
i--;
}
else
if((i%d1!=0)&&(i%d2==0)&&(i%d3==0))
{
 for(i=0;i<e2;i++)
 {
    a[i]=2;
    i++;
    }
    i--;
    }
    else
    if((i%d1!=0)&&(i%d2!=0)&&(i%d3==0))
    {
     for(j=0;j<e3;j++)
     {
     a[i] = 3;
     i++;
     }
     i--;
     }
     else
     if((i%d1!=0)&&(i%d2!=0)&&(i%d3!=0))
     {
     for(j=0;j<e2;j++)
     {
      a[i] = 2;
      i++;
```

```
}
i--;
}
else
if((i%d1==0)&&(i%d2!=0)&&(i%d3!=0))
{
for(i=0; i<e1 ; i++)
{
cout<<"\n a"<<i<<" = " << a[i];
}
cout<<" \n OPERATING SPEED : ";
cin>> S;
cout<<"\n SPEED OF THE PROCESSOR ";
cin>> sp;
cout<<"\n SPEED OF THE DEVICE ";
cin>> sd ;
cout <<"\n NUMBER OF JOBS FOR TASK1" ;
cin>> n1 ;
cout<<"\n NUMBER OF JOBS FOR TASK2" ;
cin>>n2;
cout<<"\n NUMBER OF JOBS FOR TASK3 " ;
cin>> n3;
cout<<"\n ENTER THE CHOICE";
cin>> ch;
switch(ch)
{
case 1 : cout<<" \n CALCULATION BY INVERSE RM ";
        E1 = e1* [ (sp(S)^3 +sd] *n1;
        E2 = e2* [ (sp(S)^3 +sd] *n2;
        E3 = e3* [ (sp(S)^3 +sd] *n3;
        Ei = E1 + E2 + E3;
```

```
                cout<<"THE ENERGY BY INVERSE RM :" <<Ei ;
                break;
        case 2 : cout<< "\n CALCULATION BY RM :";
                E1 = e1* [(sp(S)^3 +sd] *n1;
                E2 = e2* [(sp(S)^3 +sd] *n2;
                E3 = e3* [(sp(S)^3 +sd] *n3;
                Er = E1+ E2 + E3;
                cout<<"THE ENERGY BY RM :"<<Er;
                break;
        default : cout <<"\n NOT FEASIBLE " ;
                 break;
        }
        P = ((Ei / Er)*100);
        cout<<"\n PERCENTAGE BY WHICH ENERGY IS REDUCED "<< P;
        getch();
 }
float task1 ( float e1 , int p1 , int d1 )
{
 float a1;
 a1 = e1/p1 ;
 return a1 ;
}

float task2 ( float e2 ,int p2 , int d2 )
{
 float a2;
 a2 = e2/p2 ;
 return a2 ;
}

float task3 ( float e3 , int p3 , int d3 )
```

```
        {
         float a3;
         a3 = e3/p3;
         return a3 ;
}
```