ENERGY EFFICIENT SCHEDULING TECHNIQUES FOR REAL-TIME EMBEDDED SYSTEMS

A Thesis

by

RAJESH BABU PRATHIPATI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2004

Major Subject: Computer Science

ENERGY EFFICIENT SCHEDULING TECHNIQUES FOR REAL-TIME EMBEDDED SYSTEMS

A Thesis

by

RAJESH BABU PRATHIPATI

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Rabi N. Mahapatra (Chair of Committee)	Wei Zhao (Member)
Krishna R. Narayanan (Member)	Valerie E. Taylor (Head of Department)

May 2004

Major Subject: Computer Science

ABSTRACT

Energy Efficient Scheduling Techniques for Real-Time Embedded Systems.

(May 2004)

Rajesh Babu Prathipati, B.Tech., R.E.C. Warangal, India

Chair of Advisory Committee: Dr. Rabi N. Mahapatra

Battery-powered portable embedded systems have been widely used in many applications. These embedded systems have to concurrently perform a multitude of complex tasks under stringent time constraints. As these systems become more complex and incorporate more functionality, they became more power-hungry. Thus, reducing power consumption and extending battery lifespan while guaranteeing the timing constraints has became a critical aspect in designing such systems. This gives rise to three aspects of research: (i) Guaranteeing the execution of the hard real-time tasks by their deadlines, (ii) Determining the minimum voltage under which each task can be executed, and (iii) Techniques to take advantage of run-time variations in the execution times of tasks. In this research, we present techniques that address the above aspects in single and multi processor embedded systems. We study the performance of the proposed techniques on various benchmarks in terms of energy savings.

To my Parents and the Real-Time Embedded Systems Research Community

ACKNOWLEDGMENTS

I would like to thank Dr. Mahapatra for all the help and guidance that he offered. The quality of this work has been improved tremendously by his persistence and unwavering interest. I would also like to thank him for giving me an opportunity to do research.

I would also like to thank Dr. Zhao for helping me in taking up this interesting work. This work wouldn't have been possible without his initial direction.

I would like to thank Dr. Narayanan for serving on my committee.

I would like to thank Subrata Acharaya and Nitesh Goyal for candid discussions and infinite patience in reading the manuscript to improve the work. Those lengthy discussions reinforced my understanding on this work. I would also like to thank Anand and Junyi for all the support they rendered during the final days.

Finally, I am grateful to my parents for the hard work and the ethics they have taught me. Their constant encouragement and the strong belief in my abilities had kept me going in turbulent times. I am greatly indebted to them for their sacrifices to create a better life for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The rapid progress in semiconductor technology has led to higher chip density and operation frequency, making today's systems more complex and power-hungry. The power consumption of microprocessors has increased almost linearly with area-frequency product over the years. Such high power consumption requires expensive packaging and cooling techniques given that insufficient cooling leads to high operating temperatures, which tend to exacerbate several silicon failures. To maintain the reliability of their products, and avoid expensive packaging and cooling techniques, manufacturers are now under strong pressure to reduce the power dissipation of their products.

The advent of portable systems has also emphasized the importance of efficient use of energy as a major design objective. This is due to several concerns. The convenience of using a portable system relies heavily on its recharging interval. A system that requires frequent recharging is inconvenient and hence limits the user's overall satisfaction in using the product. Clearly, power has become a major consideration in the design of today's applications due to portability, reliability, and cost concerns. For many years to come, the search for low power and voltage

---

The journal model is *IEEE Transactions on Parallel and Distributed Systems.*

techniques will continue. The answers to this ever-increasing demand are the system-level, architectural-level, and circuit-level low power techniques [29].

A. MICROPROCESSOR AND ENERGY CONSUMPTION

In recent years, processor performance has increased at the expense of drastically increased power consumption [18]. The average power dissipation, p_{av}, in microprocessors is due to three components:

$$p_{av} = p_{cap} + p_{leak} + p_{short} \quad \text{(1)}$$

where $p_{av,}$ $p_{leak,}$ and p_{short} represent capacitive switching power, leakage power, and short-circuit power respectively. The capacitive switching power, p_{cap}, depends on the transition activity ‘alpha’, switched capacitance C_l, the supply voltage v_{dd}, and the clock frequency ‘f’. It can be expressed as

$$p_{cap} = \alpha \, . \, C_l \, . \, V^2_{dd} \, . \, f \quad \text{(2)}$$

The leakage power is due to the sub threshold behavior of MOSFETs and is expressed as

$$p_{leak} = I_{leak} \, . \, V_{dd} \quad \text{(3)}$$

Where I_{leak} is the leakage current.

The short-circuit power is due to the flow of current from power supply to the ground for short moment during node transitions between logic states '0' and '1' [20] and is given by

$$p_{short} = 0.5 \cdot \alpha \cdot (t_r \cdot I_{short,max,r/f} + t_f \cdot I_{short,max,r/f}) \cdot V_{dd} \cdot f \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4)$$

The processor clock frequency, f, can be expressed in terms of supply voltage v_{dd}, and threshold voltage v_t, as follows:

$$f = k. \ (v_{dd} - v_t)^2 \ / \ v_{dd} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (5)$$

From equation (2), we can see that there is a quadratic dependence between the supply voltage and the capacitive switching power. Therefore, reducing the supply voltage is an effective means for the reduction of processor power consumption. In recent years, processors running on multiple supply voltages have become available [19]. The supply voltage of the processor may be adjusted dynamically for many applications to reduce the processor power consumption while meeting the deadlines [1].

B. MOTIVATION

High complexity and performance has led to increased power consumption. Recent studies have shown that the processor power consumption has increased almost linearly with the performance. Such high power consumption resulted in high operating temperatures, which exacerbated the chip failures. These reliability concerns have put a special emphasis on low power design. The portability and user convenience of battery-

powered portable systems has also emphasized the low-power design. System designers are on a look for even better low-power techniques to combat the high manufacturing costs and also to improve the reliability and user convenience of their systems. This need for low-power techniques has led to increased research.

C. CONTRIBUTIONS OF THIS THESIS

The contributions of this thesis are as follows:

1. A low-power scheduling technique for single processor is presented that considers task sets with arbitrary response times. While most of the approaches consider task deadline to be less than or equal to its period, we address the tasks that have no such restrictions.

2. Introduced a connection-based task execution approach for distributed systems to model low power. The proposed approach effectively distributes the slack available in the connection among the nodes to increase the worst-case delay tolerable by the messages of a connection at the nodes involved in the connection. The proposed approach also adapts the clock speed periodically to take advantage of run-time variations.

CHAPTER II

BACKGROUND

A. INTRODUCTION

An increasing amount of system functionality tends to be realized through software, which is leveraged by the high performance of microprocessors. As a result, system-level power reduction techniques, especially at operating systems level, are becoming relevant. The system level power reduction techniques can be broadly divided in to two types [2]. The first one is known as dynamic power management where the processor is put in power-down mode during idle intervals where only certain parts like clock generation and timing circuit are kept running. The second one is low-power scheduling technique where the processor speed is dynamically varied according to the computational demands. This method requires the use of variable supply voltage to reduce the processor power consumption. A number of techniques have been proposed to reduce the system-level power consumption.

A detailed taxonomy of the system-level power management techniques is given in section B. A detailed discussion of the energy efficient scheduling techniques that has been proposed so far for single processor systems is given in section C. A review of the research work on low power scheduling in multiprocessor systems is given in Section D. Section E briefly discusses the limitations of the existing work and the merits of the proposed techniques.

B. SYSTEM LEVEL POWER MANAGEMENT TAXONOMY

The high level taxonomy of system level power management techniques is shown in Figure 1. The techniques of dynamic power management and low power scheduling are applied to both the single and multi processor embedded systems with fixed or varying task set as shown in Figure 2 and Figure 3 respectively. Fixed task set is associated with the systems whose functionality remains constant through out the life

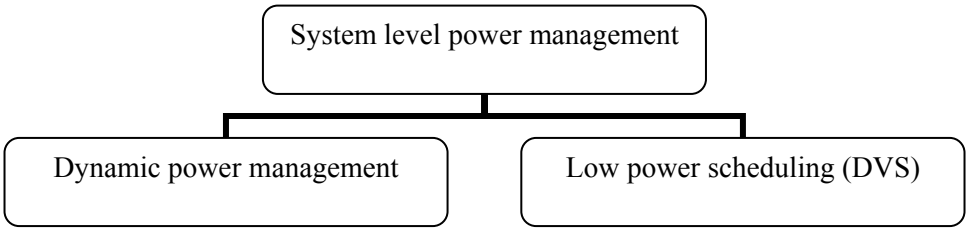

Figure 1: High-level classification of system level power management

time of the system. On the other hand, variable task set is associated with the multi-mode systems whose functionality can be varied dynamically. As the system mode changes, the applications and the corresponding task set associated with it also varies. Only limited work [12, 30] has been done so far that employed dynamic power management technique. All of this work has been concentrated on single processor

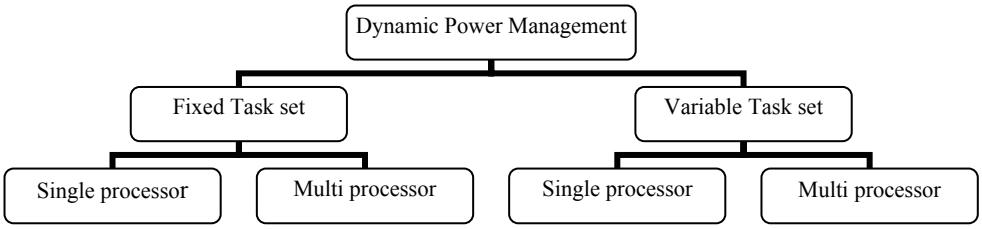

Figure 2: Broad classification of research work on dynamic power management

systems with fixed task set. Extensive research work has been carried out on low power scheduling in single processor embedded systems with fixed task set. As shown in Figure 3, this work can be broadly categorized into: 1) real-time systems with tasks whose response time is less than or equal to period. 2) real-time with tasks whose response time is arbitrary.

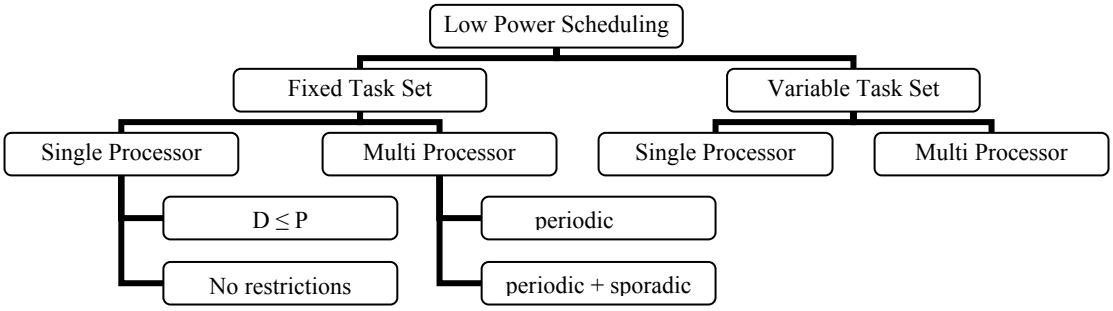

Figure 3: Broad classification of work on low power scheduling

Most of the work that has been done on single processors is targeted on the systems with tasks whose response time is less than or equal to the period. As shown in Figure 4, this work can be broadly categorized in to: 1) the techniques [24] that address multimedia systems that can afford to miss certain percentage of deadlines. 2) The techniques that address hard real time systems that can not afford to miss any deadline. Most of the work that has been done so far has addressed hard real-time systems with periodic and independent tasks [2,5,6,,8,9,27]. Only limited work has taken into account both the periodic and sporadic tasks [10,11,13]. There is no work that provides a general framework that encompasses both periodic task graphs and sporadic tasks. All

the work that has been done for multi processor systems has concentrated on fixed task systems with periodic and sporadic tasks [4,7,21,22,23,26].

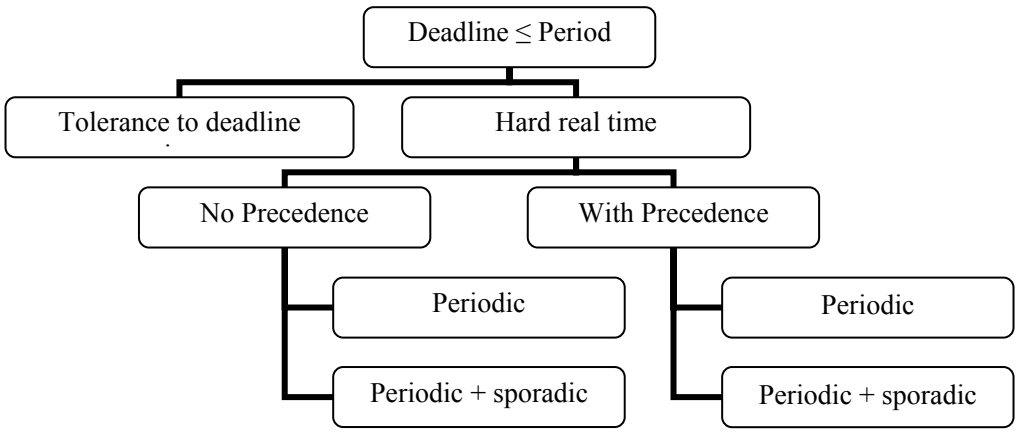

Figure 4: Broad classification of work on tasks with response time less than or equal to the period

C. SINGLE PROCESSOR

A dynamic power management technique that predicts the length of the upcoming idle interval is presented in [30]. It uses the idle interval by putting the processor in power-down mode. The drawback of this approach is it needs the traces of application to determine the predictive formula. In [9], a power optimization method that combines the off-line and on-line components for real-time embedded applications on a variable voltage processor is presented. The drawback of this method is that it does not maximally exploit the flexibility present in the schedule. It can be easily seen that the energy savings can be further improved. Swaminathan et al. has proposed a low-energy earliest deadline first heuristic [10] that tries to schedule the task with nearest

deadline first at a lowest possible voltage. The problem with this approach is that it does not guarantee that the required processor speed to meet the deadlines will never be greater than the processor clock speed even after accepting the task. In [6] Okuma et al. proposed a static voltage assignment technique for each task to minimize the total energy consumption. This technique assumes that the task set is statically order-scheduled in advance. Quan et al. [5] presented a technique that tries to determine the minimum constant voltage for each interval. All the jobs that start in an interval will be executed at the pre-determined voltage for that interval. An iterative slack distribution algorithm to minimize the system energy consumption is proposed in [13]. This algorithm takes in to account both periodic and sporadic tasks. An incremental on-line heuristic that dynamically adjusts the clock schedule when tasks enter and leave the system is proposed in [8]. This heuristic orders tasks according to how tight their deadlines are and how often tasks overlap. Hong et al. proposed an on-line scheduling algorithm [11] for scheduling the mixed workload of both sporadic and periodic hard real-time tasks on variable voltage processor. This technique employs an acceptance test on the fly to verify whether a sporadic task can be scheduled with out causing the other tasks which are already accepted to miss their deadline. The work in [12] presents several dynamic power management policies to enhance the battery-life time. These policies are broadly classified in to two: An *open-loop* policy that takes the decisions about shutting down the component independently from battery-voltage measurement. Closed-loop policies whose decision rules use to control the state of operation of the system are based on the observation of battery's output voltage. The work in [24]

presents several energy reduction techniques for soft real-time applications like multimedia that are able to tolerate occasional failures in meeting the deadlines. These techniques exploit this tolerance to occasional failure to reduce the system power consumption further. An analytical battery model that accurately predicts the battery lifetime under various discharge conditions has been proposed in [28]. In this model, the tasks are ordered in a sequence that minimizes the cost function to enhance the lifetime of the battery. A voltage allocation technique that produces a feasible task schedule for tasks with arbitrary arrival-time/deadline constraints is presented in [27]. This work also addresses the case in which tasks have non-uniform switched capacitances.

D. MULTI PROCESSOR

Even though there has been extensive work in the literature on the scheduling of real-time tasks in an energy-efficient manner on single processor embedded systems, relatively little work has been done in the area of multiprocessor embedded systems [26]. The work in [3] presents a method for the joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. Luo et al. proposed a power conscious algorithm [7] for jointly scheduling multi-rate periodic task graphs and aperiodic tasks in distributed real-time embedded systems. The algorithm first statically schedules the periodic task graphs and then creates the slots in the static schedule to accommodate hard aperiodic tasks. The work in [4] employs a list-scheduling algorithm to schedule the tasks. It employs a global shifting scheme, which distributes the slack

present in the schedule to flatten the discharge power-profile of the system. It then performs a series of local schedule transformations starting from an initially valid schedule to optimize the discharge current profile. The drawback with the above approaches is that they are only suitable to small-scale distributed real-time embedded systems. Zhu et al. proposed a slack reclamation scheme that allows slack sharing among processors to effectively reduce the energy consumption of the system [22]. The scheme employs a global queue and each processor selects the highest priority task for execution from the queue. The approach is limited to homogenous processors that share a common memory. Mishra et al. proposed greedy and gap-filling dynamic power management techniques to use the idle periods to execute tasks at reduced speed for energy savings [26]. The greedy technique allocates all the available slack on one processor to the next expected task running on that processor. The gap-filling technique fetches the future ready task and executes it. The execution of the out-of-order task will be preempted by the next expected task when it is ready. The approach is only applicable to task graphs with a common deadline. The work in [21] presents a static schedule algorithm that constructs a variable voltage schedule via critical path analysis and task execution order refinement. The approach uses static schedule generated by a list-scheduling algorithm. It then distributes the free slack time available in each critical path evenly among the tasks involved in the critical path to reduce the system power consumption. The approach also guarantees the precedence relationships between the tasks. A two-step iterative syntheses approach that partitions, schedules, and voltage scales multi-rate task graphs is presented in [23]. The approach considers the processing

element power profile during the voltage selection for each task. A power-aware scheduling algorithm that performs execution order optimization of scheduled events to increase the chances of scaling down voltages and frequencies of processing elements is presented in [25].

E. LIMITATIONS AND CONTRIBUTIONS

All of the work that has been done on single processor systems has been concentrated on systems with periodic and independent tasks. Very limited work exists that target the systems with periodic and sporadic tasks. A general frame work that encompasses periodic and sporadic tasks is lacking. In this work, we propose a technique that addresses periodic task graphs and sporadic tasks with precedence constraints. Further, all of the existing work targets the systems with tasks whose response time is less than or equal to the period. We take into account the tasks whose response time is arbitrary. The later case is particularly important for multimedia systems where the deadlines of the tasks are often greater than the period.

In this work, we have also introduced a connection based task execution approach to model low power in distributed embedded systems. In contrast to the existing work that addresses the system with fixed task set, the proposed techniques target the systems whose functionality can be configured dynamically.

CHAPTER III

ENERGY EFFICIENT SCHEDULING TECHNIQUES FOR SINGLE PROCESSOR EMBEDDED SYSTEMS

A. INTRODUCTION

As batteries power an increasing number of electronic systems, power efficient design of real-time embedded systems become important. Dynamic voltage scaling and power management represents the two system-level techniques to reduce the system power consumption and thereby extend the battery lifetime. Dynamic voltage scaling refers to dynamically varying the speed of a processor by changing the clock frequency along with the supply voltage [22]. Dynamic power management refers to the use of power-down modes when the processor is idle to reduce the processor power consumption [12].

Typically, the input specifications of these embedded systems applications are in the form of task graphs. A *task graph* is a directed acyclic graph in which each node is associated with a task and each edge is associated with the amount of data that must be transferred between the two connected tasks. The task graph is like a data flow graph but with higher functional granularity. Task graphs can be periodic or sporadic. Each task in a periodic task graph inherits the task graph's period. Each task in a periodic task graph can have a different deadline. For sporadic tasks, generally a minimum inter instance arrival interval, τ, denoting the minimum time interval between two consecutive instances of a sporadic task is specified [7]. A sporadic task can be invoked

for execution at any time and an execution slot must be available at the required time to meet the deadline. In our method, we create a periodic execution slot to serve the sporadic tasks with out causing the periodic task graphs and other sporadic tasks to miss the deadline.

Most of the work that has been done so far has concentrated only on periodic and independent tasks [2,5,6,8,9,27]. A general framework encompassing periodic and sporadic tasks with precedence constraints is lacking. Also, the current work is limited to the systems with tasks whose response time is less than the period. However for certain multimedia systems where the deadline of tasks is greater than the period, the response time of the tasks could be larger than the period. This implies that there could be more than one instance of a task active at any point of time. The deadlines of all these instances must be guaranteed while determining the minimum speed at which a task can be run. However, the existing work assumes that there is only one active instance of a task while determining the minimum speed. Consequently, they are limited to the systems with tasks whose response time is less than or equal to the period. In this work, we take into account of the tasks whose response time could be greater than the period. We also consider both periodic task graphs with precedence constraints and sporadic tasks. The proposed scheduling method has two components: *off-line* and *on-line.* The "*off-line"* component determines the lowest possible clock speed for each task while guaranteeing the deadlines and precedence constraints. The *"on-line"* component dynamically adjusts the clock speed to take advantage of idle periods and run-time variations in the execution time of the tasks. The proposed approach is evaluated on

several real-time benchmarks like CNC controller [16], avionics [14] ... Experimental results have showed that the proposed approach yields significant energy savings.

The rest of the chapter is organized as follows: A brief discussion of the preliminaries is given in section B. The procedure to determine the minimum voltage for each task is given in section C. A brief discussion on dynamic power management is given in Section D. Section E presents an illustrative example. Section F presents the experiments conducted to evaluate the proposed approach. Section G concludes the work.

B. PRELIMINARIES

Each application-specific function executed by an embedded system is made up of several sequential and/or concurrent task graphs. Figure 5 shows an example of the input specification of typical embedded systems. As discussed earlier each node of a task graph represents a task. A periodic real-time task is characterized by (φ, P, e, D), where

- P – represents the period between the successive instances of a task
- e – represents the worst-case execution time demanded by the task.
- D – represents the deadline associated with the task.
- ϕ – represents the phase.

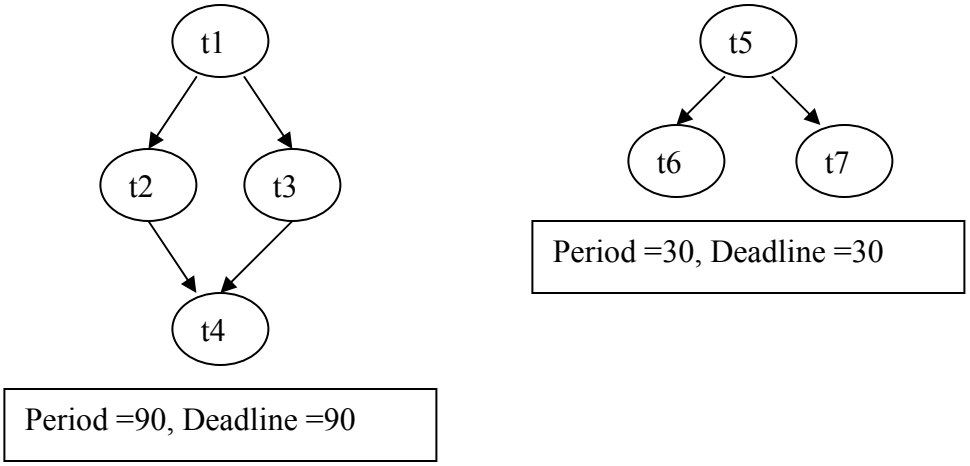

Figure 5: Typical input specification of real-time embedded systems

A sporadic task can arrive at the system for execution at any time and an execution slot must be available at the required time to meet the deadline. Let 'τ' be the minimum inter-instance arrival interval between two successive instances of a sporadic task, 'μ' be the worst-case execution time demanded and 'd' be the deadline. A periodic execution slot with period (d-μ), worst-case execution time 'μ', and deadline (d-μ) is created to serve the sporadic task. Clearly, this slot can guarantee that the sporadic task will meet its deadline. Likewise, a periodic execution slot is modeled for every sporadic task. The resultant system contains a set of periodic task graphs and a set of periodic execution slots corresponding to sporadic tasks. The periodic task graphs and execution slots are scheduled according to their assigned priorities. The assigned priorities should guarantee the precedence constraints between various tasks in a task graph. Figure 6 depicts a detailed step-by-step approach to assign priorities to tasks in a task graph and periodic slots corresponding to sporadic tasks.

1: **Input:** *A list G containing the periodic task graphs and execution slots*
2: **Output:** *A priority list σ with the tasks arranged in decreasing order of priorities.*
3: *While (G)*
4: *Initialize ActiveList A to NULL*
5: *Remove the task graph or execution slot 'T' with the smallest period from G.*
6: *Add the Level 1 nodes of T to ActiveList A*
7: *If A=0 then return σ.*
8: *While A≠0*
9: *Remove the node 'n' in A with least slack time.*
10: *Append 'n' to σ.*
11: *While (successor(n) ≠ 0) do*
12: *v := extract one node from successor(n)*
if (Pred(v) == 0) // if there is no predecessor in ActiveList
13: *Update the activelist 'A' with node 'v'.*
14: *End While.*
15: *End While*
16:***End While***

17: Output *σ.*

Figure 6: Algorithm 1 to assign priorities to task graphs and execution slots

The periodic task graphs and the execution slots corresponding to the sporadic tasks are arranged in the list in the increasing order of their period. First, a task graph with the smallest period is removed from the list and all the tasks in the task graph are assigned priorities. A node in a task graph is assigned a priority which is lesser than its predecessors and is greater than its successors. In this manner, all the precedence constraints between the nodes are guaranteed. The above procedure is repeated until all the task graphs are assigned priorities. The complexity of the above algorithm is of the order O(n), where 'n' is the total number of tasks.

C. LOW POWER SCHEDULING

Let $\Im = \{T_1, T_2, \ldots, T_N\}$ be the set of tasks that are arranged in the decreasing order of their priorities as determined by algorithm in Figure 6. Let $\{P_i, e_i, D_i\}$ represent the period, worst-case execution time, and deadline associated with the task T_i. A task set is called feasible if the deadline of each task is satisfied at all times. To minimize the energy consumption, lowest possible speed for each task that guarantees the feasibility of the task set is to be determined. In order to determine the lowest possible speed and the corresponding voltage for each task, schedulability analysis of the task set is required. According to the Critical Instant Theorem [17], if a task meets its deadline whenever the task is requested simultaneously with all the high priority tasks, then the deadline will always be met for all task phasing.

In other words, the task set $\Im = \{T_1, T_2, \ldots, T_N\}$ is schedulable if and only $t_i \leq D_i \; \forall \; i = 1, .. n$, where t_i is the response time of the task with priority 'i' when the task is requested simultaneously with the high priority tasks. The response time of real-time tasks can be broadly divided into two cases: 1) the response time of task whose deadline is less than or equal to the period and is given by

$$\sum_{k=1}^{i-1} \left\lceil t_i / P_k \right\rceil e_k + e_i \leq t_i \quad [31] \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (6)$$

The first term in the sum on the left hand side of the above inequality represents the amount of time the processor has spent in serving the tasks with priority higher than the

task T_i. The value of 't_i' that satisfies the above inequality gives the response time for task T_i. If the response time is less than the period, then only one instance of the task will be active at any point of time. In other words, an instance of a task finishes its execution before the next instance is released. Hence, in order to guarantee the feasibility of the task set, we just have to guarantee that the deadline of the critical instance of the task that is released simultaneously with all the high priority tasks. Most of the existing work in the literature that has been done so far belongs to this case. 2) The response time of a task whose deadline is greater than the period. In this case, more than one instance of a task could be active during critical instant. The response time of various active instances of the same task when the task is requested along with the high priority tasks is given by

$t_{i,j} = Rt_{i,j} - (j-1)P_i$, where

$$Rt_{i,j} \leq \sum_{k=1}^{i-1} \left\lceil Rt_{i,j} / P_k \right\rceil e_k + j*e_i \quad [31] \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (7)$$

where '$t_{i,j}$' is the response time for the 'j^{th}' instance of task T_i and $Rt_{i,j}$ represents the length of the interval between the start of the current level-Π_i busy interval and the time at which 'j^{th}' instance of a task has finished its execution. A level-Π_i busy interval is an interval where the processor is busy serving tasks with priority 'i' or high. The response time '$t_{i,j}$' for j^{th} instance of a task can be obtained by deducting the interval between the start of the current level-Π_i busy interval and the time instant at which the 'j^{th}' instance of task T_i is released from $Rt_{i,j}$. A task set is feasible if and only if $t_{i,j} \leq D_{i,j} \quad \forall \quad i$

=1,..n, and 'j' instances of t_i , where '$t_{i,j}$' is the response time for the 'j^{th}' instance of task with priority 'i'. Hence, in order to guarantee the feasibility of the task set, the deadlines of all the active instances must be guaranteed. There is no work in the literature that addresses this case. Even though it is imperative to guarantee the deadlines of the real-time tasks, there is no additional benefit in finishing the processing early. Instead, we can reduce the clock speed by extending the task execution time. Figure 7 depicts the algorithm that determines the lowest possible speed and the corresponding voltage for each task. The new speed determined for each task satisfies

---

1: ***Input:*** *A "prioritylist" of tasks in the decreasing order of their priorities.*
2: ***Output:*** *A "voltagefactor" table containing tasks and their corresponding scaling factor.*
3: $limit\_interval \leftarrow 1;$
4: $voltagefactor[i] \leftarrow 1; \ \forall \ i = 1,2, ..n.$
5: $while\ limit\_interval \neq n$
6: $for\ i \leftarrow limit\_interval\ to\ n\ do$
7: *determine the response time of the first job in task* T_i *according to* $t^{l+1} = e_i + {}_{k=1}\Sigma^{I-1} \lceil t^l/p_k \rceil e_k$. *Solve this iteratively until* $t^{l+1} = t^l$. *let this final value of* t^{l+1} *be* $W_{i,1}(t)$.
8: $if\ (W_{i,1}(t) \leq p_{i,1})\ \&\&\ (W_{i,1}(t) \leq D_{i,1})\ then$
10: $S_i = \{ jT_k |\ k = 1,2,...i;\ j = 1,2,...min(D_{i,1}, p_{i,1})$
11: $scale[i] = max\{S_{i,j} / {}_{k=1}\Sigma^{I} e_k \lceil S_{i,j} / T_k \rceil\},\ j = 1,2,..\lfloor S_i \rfloor$
12: *else if* $(W_{i,1}(t) \leq p_{i,1})\ \&\&\ (W_{i,1}(t) > D_{i,1})$ *then task set is not schedulable.*
13: *endif*
14: $if\ (W_{i,1}(t) > p_{i,1})\ then$
15: *compute the length of level-*π_I *busy interval by solving the equation* $t = {}_{k=1}\Sigma^{i} \lceil t/p_k \rceil e_k$ *iteratively starting from* $t^{(1)} = {}_{k=1}\Sigma^{I} e_k$ *until* $t^{l+1} = t^l$. *The solution* t^l *is length of level-*π_I *busy interval.*
16: $for\ j \leftarrow 1\ to \lceil t^l/p_i \rceil\ do$
17: *find response time* $W_{i,j}$ *of* j_{th} *job by solving the equation* $t = W_{i,j}(t+(j-1)p_i)-(j-1)p_i$. *Where* $W_{i,j}( )$ *is given by* $W_{i,j}(t) = je_i + {}_{k=1}\Sigma^{I-1} \lceil t/p_k \rceil e_k$.
18: $if\ W_{i,j}(t) \in ((j-1)p_i , (j-1)p_i + D_i]\ then\ continue;$
19: *else "taskset is not schedulable"*
20: *endfor*

Figure 7: Algorithm 2 to determine the minimum possible speed for each task

21: scale[i] = min{$W_{i,j}$} for j =1,2,... $\lceil t/p_i \rceil$
22: endif
23: endfor
24: limit_factor ← min { scale[i] }, i= limit_interval, ...n.
25: limit_index = { i| limit_factor = min { scale[j] } , j = limit_interval,...n}
*26: voltagefactor[j] ← voltagefactor[j] * limit_factor, ∀j = limit_interval, ...n*
27: limit-interval ← limit_index
28: end while
29: output "voltagefactor"

Figure 7: continued

equations (6) and (7) and thereby guaranteeing the feasibility of the schedule. Initially, all the tasks are arranged in the list in the decreasing order of their priorities. The response time of a task at each priority level is determined using either equations (6) or (7). This response time is extended to the deadline to determine the factor by which the clock speed can be reduced with out causing the task at the current priority level and the other high priority tasks miss the deadline. The above procedure is repeated for each priority level to determine the minimum speed at which each task can be run. The complexity of the above algorithm is O(n), where 'n' is the number of tasks.

D. DYNAMIC POWER MANAGEMENT

Even if the tasks are scheduled at the speed determined in the above section, still there will be the idle intervals. These idle intervals arise due to two reasons: 1) the idle intervals that are inherent to fixed priority schedules and 2) the idle intervals that arise due to run-time variations in execution time of tasks. During system operation, the

execution time of each task frequently deviates from the worst-case execution time. Sometimes, the deviations could be very large. These idle intervals cannot be exploited by the speeds determined during the off-line phase. An on-line method that adjusts the clock speed dynamically to exploit the idle intervals is needed to reduce the power consumption of the system further. The scheduler maintains a queue called "readyqueue". All the tasks that are ready are kept in the "readyqueue" in the order of their priority. The task that is currently being executed is called "active-task". Figure 8 depicts an algorithm for on-line power manager.

```
---------------------------------------------------------------------------------------------------
1: Input: signal for the new task arrival or completion of the existing task
   and "voltagefactor" table.
2: Output: task to be scheduled or power mode for the idle interval.
3:  if (newtask has arrived)
4:       if priority(newtask) > priority(activetask) then
5:            scale the processor to the voltage level given by voltagefactor[newtask].
6:            switch the tasks.
7:      endif;
8:      else "insert the task in readyqueue"
9: endif
10:  if (activetask has finished) then
11:       if (readyqueue is empty) then
12:           determine the length of idle interval
13:          determine the optimal power-down mode
14:      endif
15:      else
16:              if the current task has finished early
17:                    b ← current_time
18:                   e ← estimated end time of current task
19:                t← min { (b-e),( deadline of the task in the ready queue – estimated
                        end time) }
20:              distribute this slack to the "next expected" task
21:          lookup the index for this task in the "voltagefactor" table and scale
             the processor to that level.
22:            endif
23:     endif;
24: endif.
25:   output the task to be scheduled or nothing
---------------------------------------------------------------------------------------------------
```

Figure 8: Algorithm 3 for on-line power manager

During the system operation, the scheduler is invoked when of the two conditions occur: 1) a new task is ready and 2) the current has finished its execution. When a new task is ready and the priority of the new task is higher than the current task, then the scheduler preempts the current task and schedules the new task. Other wise, the new task is kept in the "readyqueue". If the current task has finished its execution and there is no task in the "readyqueue", then the scheduler estimates the length of the idle interval. If the length of the idle interval is feasible, the processor is put in the power down mode. The above algorithm has a constant time complexity.

E. ILLUSTRATIVE EXAMPLE

Let $\Im = \{T_1, T_2, T_3\}$ be the set of tasks that are arranged in the decreasing order of their priorities as determined by algorithm in Figure 6. Table 1 shows the characteristics of the tasks. As shown in Table 1, the deadline of task T_3 is greater than its period. Hence the response time for task T_3 could be larger than the period. Figure 9 shows the time valid schedule of the tasks for the length of the hyper period. As shown in Figure 9, there is more than one instance of task T_3 that is active at any point of time.

Table 1: Characteristics of tasks

Task	**Period**	**Execution time**	**Deadline**
T_1	10 sec	4sec	10sec
T_2	15 sec	3 sec	15 sec
T_3	20 sec	8 sec	30 sec

The deadlines of all these instances must be guaranteed while determining the minimum speed at which task T_3 can be run. The response times for the tasks and their corresponding speeds as determined by Algorithm 2 in Figure 7 on Intel SA-1100 processor specifications is given in Table 2.

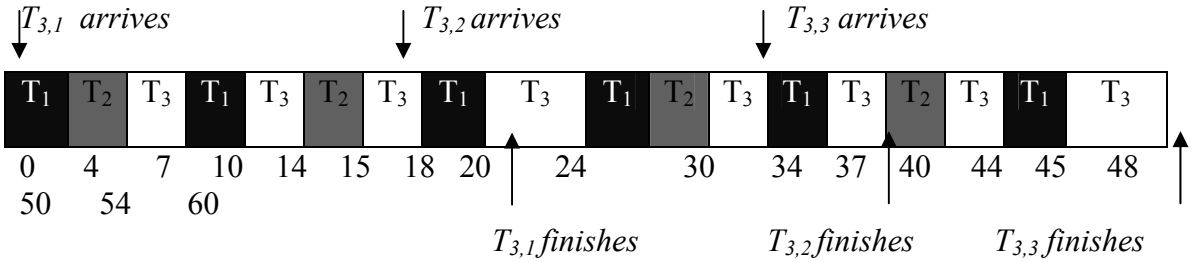

Figure 9: Time valid schedule of the tasks for the length of hyper period

The first instance of task T_3 has become a limiting instance. The tasks T_1, T_2, and T_3 must be scheduled at the speed 0.933 for all the tasks to meet the deadlines. The tasks are run at the above determined speed and the on-line power manager in Figure 8 is employed to take advantage of idle intervals. Simulation results have indicated an energy savings of 62.27 %.

Table 2: Response time and speed factors

Task	# of active instances	response time for active instances	Clock speed on SA-1100
T_1	1	4	0.4
T_2	1	7	0.466
T_3	3	(26,25,20)	(0.933,0.866,0.666)

F. CASE STUDY AND EXPERIMENTAL RESULTS

An event driven simulator has been developed to evaluate the merit of the proposed technique. The simulator runs on the Linux operating system. An instance of each task is generated at an interval equal to its period using timing information generated by "system clock" functions. The performance of the proposed technique is evaluated using traces of various real-time applications on Intel StrongArm SA-1100 embedded processor specifications. The StrongArm SA-1100 is a dynamically voltage scaleable processor with eleven frequency and corresponding voltage levels. The clock and voltage specifications of StrongArm processor are given in Appendix A. The test cases used to evaluate the proposed approach consist of three synthetic cases and five real-world examples. Table 3 shows the specifications of various test cases.

Table 3: Characteristics of various test cases

Test Cases	# periodic task graphs	# sporadic tasks	# tasks with D > P
Synthetic I	3	1	2
Synthetic II	5	3	4
Synthetic III	10	5	8
CNC [16]	8	---	---
INS [15]	6	---	---
Avionics [14]	14	1	---
MP3 Player	1		4
GSM Decode	1		---

The “mp3 player” consists of tasks “scale factor”, “Huffman decode”, “Dequantize sample”, and “sub band synthesis” that are executed in a sequential order. “GSM decode” consists of tasks “RPE decoding”, “long term synthesis filter”, “short term synthesis filter”, and “post processing” that are executed in a sequential manner. Table 4 depicts the characteristics and priorities of tasks that are assigned by Algorithm 1 for mp3 player and GSM decoder. The tasks are scheduled by on-line power manager

Table 4: Characteristics of various tasks in mp3 player and gsm decoder

Task	Priority	Period	Execution time	deadline
Scale factor	1	20 msec	1024 μsec	150 msec
Huffman decode	2	20 msec	1890 μsec	150 msec
Dequantize sample	3	20 msec	4580 μsec	150 msec
Subband synthesis	4	20 msec	2675 μsec	150 msec
RPE decoding	5	18000 msec	921 msec	18000 msec
LT synthesis filter	6	18000 msec	2079 msec	18000 msec
ST synthesis filter	7	18000 msec	689 msec	18000 msec
Post processing	8	18000 msec	311 msec	18000 msec

according to their priorities in a preemptive manner. The actual execution demanded by a task will be less than the worst-case execution time due to run-time variations. This will cause the idle intervals. The on-line power manger exploits the idle intervals by putting the processor in power down mode. The overhead in performing the shutdown

sequence in Intel SA-1100 is 90 microseconds and the worst-case overhead in varying the clock speed between various levels is assumed to be 60 microseconds. The tasks are simulated for the length of hyper period, which is 18000 milliseconds. The same schedule repeats over and again for the subsequent hyper periods. Table 5 depicts the actual execution time demanded by each task during the length of the hyper period and the clock speed at which they are executed. During the period of the simulation, the processor entered the power down mode 876 times. The percentage associated overhead energy consumption due to context switches and shut down sequences in comparison with total energy consumption is 0.01 %. The total energy savings are 91.1%.

Table 5: Actual execution time and clock speed

Task	Total time actually demanded	Clock speed on SA-1100
Scale factor	312 msec	0.5333
Huffman decode	458 msec	0.533
Dequantize sample	1839 msec	0.533
Subband synthesis	908 msec	0.533
RPE decoding	502 msec	0.533
LT synthesis filter	1191 msec	0.533
ST synthesis filter	593 msec	0.533
Post processing	143 msec	0.533

A comparison of % energy savings between the proposed technique and the technique "VLPS" in [5] is given in Figure 10. As shown in the figure, the proposed technique yields higher energy savings compared to the technique proposed in [5]. The technique in [5] tries to determine the set of intervals in a hyper period that can be run at a minimum constant voltage. All the jobs in an interval are run at a constant voltage. It fails to take maximum advantage by not exploiting the flexibility available in the system to the maximum extent. The system energy savings can be further improved by

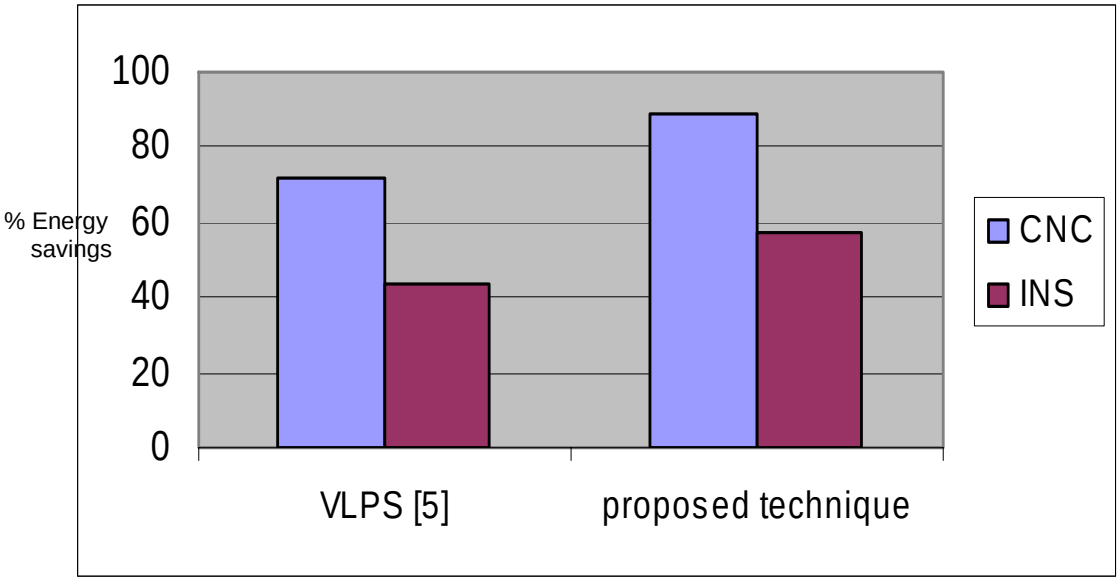

Figure 10: Comparison of energy savings with various low power techniques

running each job at a minimum possible speed. The proposed technique yields higher energy savings by determining the minimum speed for each task. The energy savings due to the proposed technique on various test cases is shown in Figure 11.

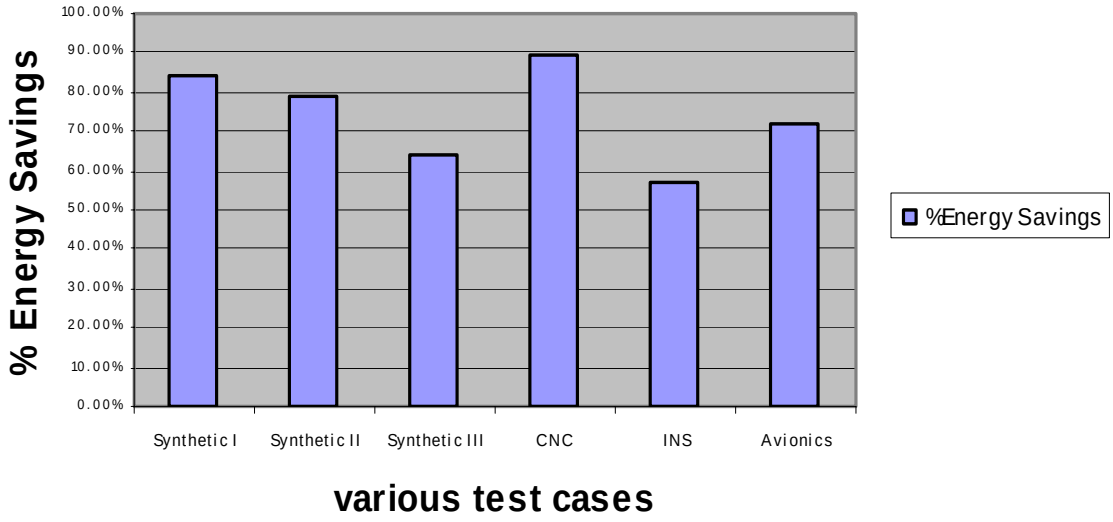

Figure 11: % Energy savings with the proposed technique on various test cases

G. CONCLUSIONS

In this work, an energy efficient scheduling technique for single processor real-time embedded systems is presented. The scheduling algorithm presented is capable of handling both sporadic and periodic task graphs with precedence constraints. The static power reduction component determines the minimum voltage at which each tasks can be run. The Online power reduction component dynamically adjusts the clock speed to take advantage of run-time variations in execution time of the tasks. It also exploits the idle intervals by putting the processor in power down modes to reduce the processor power consumption. The proposed approach is evaluated on various standard benchmarks. Experimental results show that proposed approach yields significant energy savings.

CHAPTER IV

ENERGY EFFICIENT SCHEDULING TECHNIQUES FOR MULTI PROCESSOR EMBEDDED SYSTEMS

A. INTRODUCTION

Many embedded command and control systems used in manufacturing, chemical processing, tele-medicine, and sensor networks are mission-critical. These systems are usually involved with applications that must accomplish certain tasks before pre-specified deadlines. An application in these systems comprise of chain of task set that are implemented in a distributed fashion. The nodes in the distributed system are involved with computation and exchange of messages between them in order to accomplish real-time tasks. The messages sent between these nodes have deadlines by which they must be processed. It is imperative to make the computation in the above systems energy efficient while simultaneously guaranteeing the end-to-end message deadlines.

A connection among chain of tasks is always established to realize computation and message communication in an application. To guarantee the message deadlines of a particular connection, a bound on the worst-case end-to-end delay experienced by the messages must be derived. This objective is accomplished by computing the worst-case delay suffered by messages at each node involved in the connection. The worst-case end-to-end delay is obtained by summing up the worst-case delays at each node. A connection is admitted if the worst-case end-to-end delay is less than or equal to its end-

to-end deadline. The success of this seemingly straightforward approach hinges on to critical issues. The first issue pertains to the description of a connection's traffic. The term *traffic descriptor,* Γ(I), [32] has been adopted for a method that is used to provide relevant information about a connection's traffic. A traffic descriptor should adequately describe connection's traffic not only at the source node but also throughout the nodes involved in the connection. This traffic descriptor specifies the maximum processing time demand by the messages of a connection at any point in the network during an interval of length I as Γ(I). The second issue pertains to the analysis of the worst-case delay suffered by a connection's message at each node. The scheduling policy employed at the node also affects the analysis. We consider First Come First Serve (FCFS) and Weighted Round Robin (WRR) scheduling disciplines.

While the systems must process the messages by their deadlines, there is no additional benefit in finishing the computation early. Rather by making the computation at these nodes energy efficient, the battery lifetime can be increased. The strategy to accomplish this objective is based on three key observations: 1) the processing of the messages of a connection at any node can be delayed or extended up to the worst-case delay at that node. 2) The *slack in the connection is the difference between the end-to-end deadline and the end-to-end worst-case delay of the messages of a connection.* This slack can be utilized to increase the worst-case delay tolerable at the computational nodes involved in the connection. 3) The actual processing time demanded by the messages of a connection during the run-time varies and is less than the worst-case specification. In this work, a heuristic that effectively distributes the slack available in a

connection among the nodes involved in the computation of an application is presented. A technique to adapt the clock speed according to the run-time variations to make the computation at the node energy efficient is also presented. The proposed technique has following advantages: 1) it copes very well with the variable message delays that are typical to distributed system. 2) It is applicable to both the systems with fixed task set and the systems that operate in modal fashion. 3) It yields significant energy savings.

The rest of the chapter is organized as follows: Section B presents motivational examples. A brief description of the system model and other preliminaries is given in Section C. section D presents the slack distribution technique. An elaborate discussion on the service rate determination is given in Section E. Section F presents the experimental results. Section G concludes the work.

B. MOTIVATIONAL EXAMPLES

This section presents two examples that motivated the work in this paper.

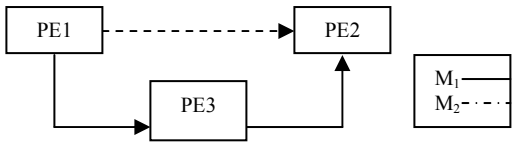

Figure 12: Typical distributed embedded systems with 3 nodes & 2 connections

Example 1: Let us consider a simple distributed embedded system consisting of three nodes and two connections: M_1 and M_2 as shown in Figure 12. The connection M_1 involves computation of nodes PE1, PE2, and PE3. The application M_2 involves computation of nodes PE1 and PE2. Table 6 gives the specification for connection M_1.

Let us consider the node PE3. We have chosen PE3 for simplicity. The techniques to deal with more complex scenarios involving several connections are presented in section D. Let 'f' be the normal operational clock frequency of this processor. The maximum processing time, $\Gamma(\Delta)$, demanded by the messages of connection M_1 at node PE3 is 5 sec and the messages suffer a worst-case delay of 5 sec. The connection M_1 can be admitted since its end-to-end worst-case delay (65 sec) is less than the end-to-end deadline (95 sec). At any node involved in the connection, messages must be processed by their worst-case delays to guarantee the end-to-end deadlines. Hence, the node PE3 must operate at its maximum speed. The slack available in the connection M_1 is 30 sec: (95- (25+35+5)). This slack can be distributed among the nodes to increase the upper bound on the delays tolerable by the messages of connection M_1. The new worst-case delay tolerable by the messages of connection M_1 at node PE3 is 15 seconds assuming that the slack is distributed equally among the nodes PE1, PE2, and PE3. Hence the new clock frequency at node PE3 is 0.33f. An efficient method for distributing the slack is presented in section D.

Table 6: Traffic specification for an arbitrary interval of length, $\Delta = 9$ sec

Nodes	$\Gamma(I)$	Worst-case delay	End-to-end deadline
PE1	6 sec	25 sec	95 sec
PE2	8 sec	35 sec	
PE3	5 sec	5 sec	

Example 2: Let us consider the node PE3 from previous example. Table 7 shows the actual computational time demanded by the messages of connection M_1 at node PE3 during an arbitrary interval of length 'Δ' =9sec. We can take advantage of these run-time variations in the processing time demand to reduce the energy consumption at the node. We will determine the service rate that is needed to guarantee the processing of the messages by their worst-case delay, 'd', at the beginning of each interval Δ. At the beginning of the first interval $[t_0,t_0+\Delta]$, the clock speed is set to 0.33f as determined from Example 1.

Table 7: Run-time variations in processing time demands

Interval	**Actual Computation time demanded**
$[t_0, t_0+\Delta]$	4 sec
$[t_0+\Delta, t_0+2\Delta]$	4 sec
$[t_0+2\Delta, t_0+3\Delta]$	2 sec
:	

At the beginning of the second interval, though the actual processing time demanded during the previous interval was 4 sec, it was possible to serve only $(\Gamma(\Delta)/d)*\Delta$, 3 time units. The remaining 1 sec of computational demand must be processed with in $(d-\Delta)$ time units, i.e., 6. The processor must be able to serve this traffic and the expected processing time demand by the messages of the upcoming

interval, i.e., 9, by their delays. Hence the required operational frequency is, (1/6 + 5/15), i.e., 0.46f. However in practical systems, the clock frequency cannot be scaled continuously. So, the processor will be put in the lowest suitable frequency mode higher than this factor. Figure 13 shows the clock speed for the four consecutive intervals d of the original schedule and new schedule.

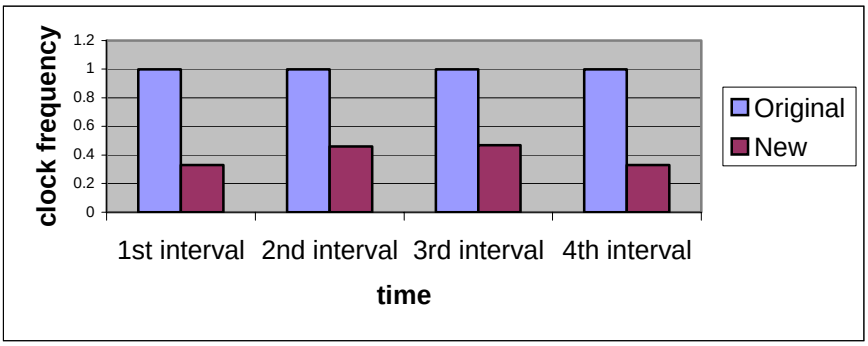

Figure 13: Clock frequency for original and new schedules

C. SYSTEM MODEL

Let us represent the set of connections supporting the currently active applications by $M \equiv (M_1,...M_n)$. A vector triplet, $(\vec{P}_i, \vec{C}_i, \vec{D}_i)$ is used to describe any individual connection M_i served over various nodes. Let us use the symbol '∇' to represent an unspecified quantity. This means that there is no restriction on the value of the corresponding parameter. We use the set {∂/η} to denote one of the conditions ∂ or η but not both. The vectors $\vec{P}_i, \vec{C}_i, \vec{D}_i$ are described below:

• The nodes involved in any arbitrary connection M_i communicate with each other by exchanging messages. Even though the messages are generated periodically at the start

node, they will not adhere to any strict periodic pattern at the subsequent nodes in the system. This is because of the variable delays they suffer in the network. This kind of behavior is accounted by not making any assumption on the periodicity of the messages at the nodes other than the start node. The vector $\vec{P}_i$ is used to represent the period with which the messages of connection M_i arrives at various nodes. It is described by ($P_{i1}, \nabla, \ldots\ldots \nabla$), where $P_{i,j}$ represents the period of the messages of connection M_i at a node 'j'.

- The worst-case processing time of the messages of connection M_i is described by the vector $\vec{C}_i \equiv (C_{i1}, .. C_{in})$ where C_{ij} represents the worst-case processing time due to computation and communication at node 'j'. The total worst-case processing time of the message is given by $\sum_{k=1}^{n} C_{ik}$.

- The vector $\overrightarrow{D_i}$ represents the deadlines for the messages of connection M_i at various nodes and is described by $\left(\{D_{i1}/\nabla\}, .. D_{in}\right)$. A message may or may not have a local deadline at a particular node, but it will have an end-to-end deadline, which is the deadline at the last node. The message has to be processed by that deadline D_{in} .

- The maximum demand function $\prod^{i}(I)$ is used to represent the processing time demanded by the messages of connection M_i over duration of length I.

D. SLACK DISTRIBUTION

As shown in Figure 12, any arbitrary connection M_i passes through a sequence of nodes. Let us represent the sequence of the nodes serving connection M_i by $H_i = <$ s(i,1),s(i,2),….,s(i,j),..s(i,k)>, where k is the total number of nodes serving connection M_i and s(i,j) denotes the node-id of the j^{th} node in the connection's path. Let us assume that $d_{i,s(i,,1)}$, $d_{i,s(i,2)}$,…,$d_{i,s(i,k)}$ are the upper bounds on the delays suffered by M_i. Let D_{ik} be the end-to-end deadline for the connection M_i. The total slack available in the connection is given by $TS_i = D_{ik} - \sum_{j=1}^{k} d_{i,s(i,j)}$. It is evident, from motivational example 1, that this slack can be distributed among the nodes serving connection M_i to reduce the energy consumption. Instead of distributing this slack equally, let us allocate more slack to the nodes where it will have maximum effect. Let us use the term *service rate* to denote the ratio of current clock speed to the normal clock speed of a processor. An efficient way of distributing the slack among the nodes of a connection is to allocate slack according to the service rate of the nodes. The amount of slack allocated to a node is proportional to the ratio of the service rate of the node to the sum of the service rates of the nodes involved in the connection. This is subject to the condition that the new worst-case delay is bounded by the local deadline at that node. The slack distribution algorithm is invoked at a node during connection admission. Admission of a new connection involves two phases: 1) connection setup and 2) reply. During connection setup phase, a connection is established between the pre-specified nodes involved in the computation of an application. A connection is established only if: i) The worst-case

delay suffered by the messages of a connection at each node is less than the local deadline, if there is a local deadline. ii) The worst-case end-to-end delay is less than the end-to-end deadline. Otherwise, the connection is rejected. If a connection is rejected, the behavior of the system is implementation dependent. A particular implementation might try to handle the exception, while another might abort the operation and reset. Algorithm 4 as shown in Figure 14 gives the pseudo code for connection setup and reply phases. The slack distribution algorithm (Algorithm 5) as shown in Figure 15 will be triggered during connection reply phase at every node involved in the connection. The following are the brief description of the system parameters at an arbitrary node s(i,j) involved in connection M_i:

- tsrate – sum of the service rates of the first 'j' nodes involved in the connection.
- $srate_i$ – service rate for connection M_i at s(i,j).
- tdelay – sum of the worst-case delay suffered by the messages of connection M_i at the first 'j' nodes.
- n – number of incoming connections at the node s(i,j).
- tslack – total slack to be distributed among first 'j' nodes.
- $slack_i$ – slack allocated to node s(i,j) for connection M_i.
- $delay\_wrr_i$ – worst-case delay for connection M_i at s(i,j) under WRR policy
- delay_fcfs – worst-case delay for connection M_i at s(i,j) under FCFS policy.

```
Procedure connection_request(tdelay_i, tsrate_i)
calculate the worst-case delay for connection M_i at s(i,j).
delay = worst-case delay determined as above
tdelay_i +=delay.
if  (tdelay_i > local_deadline)connection_ reply(REJECT,0);
determine the service rate for M_i at s(i,j). // see section 4.2.
srate_i = service rate determined as above.
tsrate_i =+ srate_i.
if (j==k) // this is the last node in the connection
     tslack_i = deadline – tdelay;
    if (tslack_i >= 0)
        connection_reply(ACCEPT,tslack);
   else
       connection_reply(REJECT,0);
else
  connection_request(tdelay_i,tsrate_i);
end connection_request
Procedure connection_reply(res,tslk)
if (res == REJECT) take an implementation dependent action.
else
   slack_distribution(tslk);
end connection_reply
```

Figure 14: Algorithm 4 for connection establishment

```
Procedure slack_distribution(tslk)
slack_i  = (tslk * srate_i)/tsrate
if  (tdelay+slack_i  > deadline)
       slack_i = deadline –tdelay;
tslk = tslk – slack_i;
#ifdef WRR
 delay_wrr_i  += slack_i; // update the delay for connection M_i.
#ifdef FCFS
   update_delay(slack_i)
end slack_distribution
Procedure update_delay(slack_i)
for (i =1;i <=n; i++)
     if (slack_i < 0) return;
     else
        if (min > slack_i) min = slack_i;
delay_fcfs = delay_fcfs+min;
end update_delay
```

Figure 15: Algorithm 5 for slack distribution

E. PERIODICAL SERVICE RATE DETERMINATION

In this section, a technique to determine the lowest possible service rate that is necessary to guarantee the upper bound on the delays of the messages of incoming connections at a given node is presented. The technique also dynamically adapts the clock speed to take advantage of the run-time variations. For this purpose, the actual processing time demanded by the messages of incoming connections at a given node during previous intervals is observed and this feedback is incorporated while determining the service rate for the current interval. The new service rate should guarantee the processing of the messages that will arrive in the upcoming interval by their delay bounds. In addition, the new service rate should also guarantee the processing of the messages that arrived during previous intervals and are waiting in the queue by their delay bounds.

a) First Come First Serve (FCFS): The following notation to describe the system parameters when the scheduling policy at the node is FCFS:

- f(t) - represents the actual processing time demanded by the messages that already arrived at the node before the time instant 't'.
- Q_t^j - represents the processing time demanded by the unprocessed messages, left in the queue, which arrived during the interval $(t-j\Delta,t-(j-1)\Delta)$ at time instant 't'.
- S_t^j - represents the required service rate at time instant 't' to guarantee the processing of the messages in the queue which arrived during the interval $(t-j\Delta,t-(j-1)\Delta)$ by their worst-case delay bounds.

- let $\{M_1,..M_k\}$ be the incoming connection set at a given node and $\{ \Gamma^1, \Gamma^2, ... \Gamma^i, .. \Gamma^K \}$ be the corresponding input traffic descriptor set. Let Γ_t^i represent the input traffic descriptor function for connection M_i at time't'.

- Let $\Gamma_t(\Delta) = \sum_{i=1}^{k} \Gamma_t^i(\Delta)$ be the total maximum processing time demanded by the incoming connection set at a given node for the upcoming interval 'Δ' at time instant 't'.

- d^{FCFS} - represents the worst-case delay suffered by the messages of incoming connection set when the scheduling policy at the node is FCFS.

- Let 'n' be the ratio between d^{FCFS} and 'Δ'. The value for 'Δ' is chosen such that 'n' is always greater than 1.

- Let $S_t = \sum_{i=1}^{n} S_t^i$ represent the required service rate at the node at time 't'.

- Let 't_s' be the system start time. We assume that the queue is zero at t_s.

At the system start time, the service rate is set according to

$$S_{ts} \geq \frac{\Gamma_{ts}(\Delta)}{d^{FCFS}} \quad \text{.....} \quad (8)$$

Clearly, this service rate will guarantee the processing of messages that will arrive in the upcoming interval 'Δ' by their delay bounds. Moreover, this service rate will always be less than or equal to 1. In practice, the actual processing time demanded by the

arrived messages will be less than the maximum processing time demand because of the run-time variations. We will take advantage of run-time variations by periodically adjusting the service rate. At the beginning of each interval, a new service rate is determined. This service rate should guarantee the processing of messages that will arrive in the upcoming interval and the messages that already arrived in previous intervals by their delay bounds. At any point of time, there will be unprocessed messages from at most 'n-1' outstanding immediate predecessor intervals left in the queue. The new service rate at the beginning of every interval is determined according to

$$S_t = \sum_{j=1}^{k} S_t^j + \frac{\Gamma_t(\Delta)}{d^{FCFS}} \quad \text{(9)}$$

and the corresponding queue is determined according to

$$Q_t = \sum_{j=1}^{k} Q_t^j \quad \text{(10)}$$

where $k = \{t - \max(t_s, t-(n-1)\Delta)\}/\Delta$.

The service rate S_t^j should be such that it must process the outstanding messages that arrived during the interval (t-jΔ,t-(j-1)Δ) by their remaining delay bound. i.e., (d^{fcfs}-jΔ).

$$S_t^j . (d^{FCFS} - j\Delta) \geq Q_t^j \quad \text{(11)}$$

$$Q_t^j = \begin{cases} \Psi \quad \text{if } \Xi > 0 \\ \\ \text{Else } \Psi + \Xi \end{cases} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \qquad (12)$$

where $\Psi = f(t-j\Delta) - f(t-\overline{j-1}\Delta)$ and $\Xi = Q_{t-j\Delta} - \sum_{k=1}^{j} \int_{t-k\Delta}^{t-(k-1)\Delta} S_{t-k\Delta}$

THEOREM 1: The service rate, S_t, determined by equation (9) will guarantee the processing of the messages by the upper bound on their delays.

Proof: To prove that S_t is a valid service rate, the following cases must be satisfied.

Case 1: The unprocessed messages arrived during any arbitrary previous interval (t-jΔ, t-(j-1)Δ) will be processed by the upper bound on their delays by the service rate S_t. i.e.,

$$S_t * (d - j\Delta) \geq Q_t^j$$

$$\Rightarrow (S_t^1 + .. + S_t^j + .. + S_t^n) * (d - j\Delta) \geq Q_t^j$$

$$\Rightarrow (S_t^1 + .. + S_t^n) * (d - j\Delta) + Q_t^j \geq Q_t^j \text{ By substituting equation (11)}$$

which is vaild $\because S_t^i * (d - j\Delta) \geq 0 \quad \forall i$

Case 2: The messages that will arrive in the upcoming interval 'Δ', along with the messages that are left in the queue will be processed by their deadlines by the service rate S_t. i.e., $S_t * d \geq \sum_{j=1}^{n-1} Q_t^j + \Gamma_t(\Delta)$

$$\Rightarrow (S_t^1 + .. + S_t^j + .. + S_t^n) * d \geq \sum_{j=1}^{n-1} Q_t^j + \Gamma_t(\Delta)$$

$$\Rightarrow (\sum_{i=1}^{n-1} \frac{Q_t^i}{d - i\Delta}) * d + \Gamma_t(\Delta) \geq \sum_{j=1}^{n-1} Q_t^j + \Gamma_t(\Delta)$$

which is valid $\because$ $i\Delta \geq 0 \quad \forall i$

This proves the theorem. ■

b) Weighted Round Robin (WRR): The following notation to describe the system parameters when the scheduling policy at the node is WRR:

- $f^i(t)$ represents the actual processing time demanded by the messages of connection M_i that already arrived at the node before the time instant 't'.
- $Q_t^{i,j}$ represents the processing time demanded by the unprocessed messages from connection M_i, left in the queue, which arrived during the interval (t-jΔ,t-(j-1)Δ) at time instant 't'.
- $S_t^{i,j}$ represent the required service rate at time instant 't' to guarantee the processing of the messages from connection M_i in the queue which arrived during the interval (t-jΔ,t-(j-1)Δ) by their worst-case delay bounds.
- let $\{M_1,..M_k\}$ be the incoming connection set at a given node and $\{ \Gamma^1, \Gamma^2, ... \Gamma^i, .. \Gamma^K \}$ be the corresponding input traffic descriptor set. Let Γ_t^i represent the input traffic descriptor function for connection M_i at time 't'.
- $\Xi_i(T)$ to represent the length of the interval during which the messages from connection M_i will be processed in a time interval 'T'.

- d_i to represent the worst case delay suffered by the messages from connection M_i at a given node.
- Let 't_s' be the system start time. We assume that the queue is zero at t_s.
- let 'n' be the the ratio between d_i and 'Δ'.
- Let $S_t^i = \sum_{j=1}^{n} S_t^{j,j}$ represent the required service rate at the node at time 't' to process the messages from connection M_i.

At system start time, the service rate is set to

$$S_{ts}^i \geq \frac{\Gamma_{ts}^i(\Delta)}{\Xi_i(d_i)} \quad \text{.....} \quad (13)$$

Clearly, this service rate will guarantee the processing of messages from connection M_i that will arrive in the upcoming interval 'Δ' by their delay bounds. At the beginning of each interval, a new service rate is determined. This service rate should guarantee the processing of messages from connection M_i that will arrive in the upcoming interval 'Δ' and the messages that already arrived during previous intervals by their delay bounds. The new service rate at the beginning of every interval is determined according to

$$S_t^i = \sum_{j=1}^{k} S_t^{i,j} + \frac{\Gamma_t^i(\Delta)}{\Xi_i(d_i)} \quad \text{.....} \quad (14)$$

and the corresponding queue is determined according to

$$Q_t^i = \sum_{j=1}^{k} Q_t^{i,j} \quad \text{..... (15)}$$

where k = $\{t - \max(t_s, t-(n-1)\Delta)\} / \Delta$

The service rate $S_t^{i,j}$ and the corresponding processing time demanded by the outstanding messages that arrived during the interval (t-jΔ,t-(j-1)Δ) are given by

$$S_t^{i,j}.(d_i - j\Delta) \geq Q_t^{i,j} \quad \text{..... (16)}$$

$$Q_t^{i,j} = \begin{cases} \chi \quad \text{if } \bar{u} > 0 \\ \text{Else} \quad \chi + \bar{u} \end{cases} \quad \text{..... (17)}$$

where $\chi = f^i(t - j\Delta) - f^i(t - \overline{j-1}\Delta)$ and $\bar{u} = \left(Q^i_{t-j\Delta} - \sum_{k=1}^{j} \int_{t-k\Delta}^{t-\overline{k-1}\Delta} S^i_{t-k\Delta} \right) > 0$

THEOREM 2: The service rate, S_t^i, will guarantee the processing of the messages of connection M_i by their upper bounds on delays.

Proof: To prove that S_t^i is a valid service rate, we must prove two cases:

Case 1: The unprocessed messages arrived during any arbitrary previous interval (t-jΔ, t-(j-1)Δ) will be processed by the upper bound on their delays by the service rate S_t^i.

$$i.e., S_t^i * (d - j\Delta) \geq Q_t^{i,j}$$

$$\Rightarrow (S_t^{i,1} + \ldots + S_t^{i,j} + \ldots + S_t^{i,n}) * (d - j\Delta) \geq Q_t^{i,j}$$

$\Rightarrow (S_t^{i,1} + ... + S_t^{i,n}) * (d - j\Delta) + Q_t^{i,j} \geq Q_t^{i,j}$ by substituting equation (16)

which is valid $\because S_t^{i,k} * (d - j\Delta) \geq 0 \; \forall$ k

Case 2: The messages that will arrive during the upcoming interval and the messages that are in the queue will be processed by their deadlines by the service rate S_t^i.

i.e., $S_t^i * d \geq \frac{\Gamma_t^i(\Delta)}{\Xi_i(d_i)} + \sum_{j=1}^{n-1} Q_t^{i,j}$

$\Rightarrow (S_t^{i,1} + ... + S_t^{i,n}) * d \geq \sum_{j=1}^{n-1} Q_t^{j,j} + \Gamma_t^i(\Delta)$

$\Rightarrow (\sum_{j=1}^{n-1} \frac{Q_t^{i,j}}{d - j\Delta}) * d + \Gamma_t^i(\Delta) \geq \sum_{j=1}^{n-1} Q_t^{i,j} + \Gamma_t^i(\Delta)$ which is valid $\because k\Delta \geq 0 \; \forall$ k

This proves the theorem. ■

F. EXPERIMENTAL RESULTS

An event driven simulator has been developed to evaluate the proposed technique. The distributed embedded system is simulated through various independent processes running on Linux and communicating through “Socket” interface. A global centralized server is used to supply the timing information to all the processes involved in the simulation. The performance of the proposed approach is evaluated using traces of various real-time applications on Intel PXA250 XScale embedded processor. The Xscale is a dynamically voltage scaleable processor with four frequency and corresponding voltage levels. The clock and voltage specifications of Xscale processor

are given in Appendix. The test cases used to evaluate the proposed approach consist of three synthetic cases and two real-world examples. Table 8 shows the specifications of

Table 8: Characteristics of various test cases

Test Cases	Number of Nodes	Number of Connections	Number of Modes
Synthetic I	3	10	2
Synthetic II	5	20	2
Synthetic III	10	30	3
Multimedia	4	4	3
DSP	16	31	1

various test cases. The Multimedia test case has applications of MPEG, JPEG, MP3, and ADPCM running in three different modes. The DSP application operates in a single mode. Table 9 shows the mode configurations for various test cases.

Table 9: Mode configurations for multimedia and synthetic test cases

Test Cases	Mode 1 (nodes, connections)	Mode 2 (nodes,connections)	Mode 3 (nodes,connections)
Synthetic (10,30)	(9,20)	(9,25)	(10,30)
Multimedia (4,4)	(3,2)	(3,3)	(4,4)

A comparison of various slack distribution schemes is depicted in Figure 16. As shown in the figure, the slack distribution technique “srate” yields more energy savings compared with any other approaches. The “greedy” technique of allocating all the available slack to the first node in the connection had eventually lead to severe connection drop rate. The “wextime” technique of distributing slack according to the

worst-case execution time of tasks at the nodes involved in the connection had a marginal effect on the overall energy savings of the system. Even though the "equal" technique of distributing slack equally among the nodes involved in the connection fared well, it is inferior to the "srate" technique. This indicates that reducing the service rate of a node with higher utilization by a factor will have a greater impact on the system-wide energy savings than reducing the service rate of a node with lower utilization by the same factor. Hence the proposed technique is superior compared with "greedy" [26], "wextime" [26], "equal".

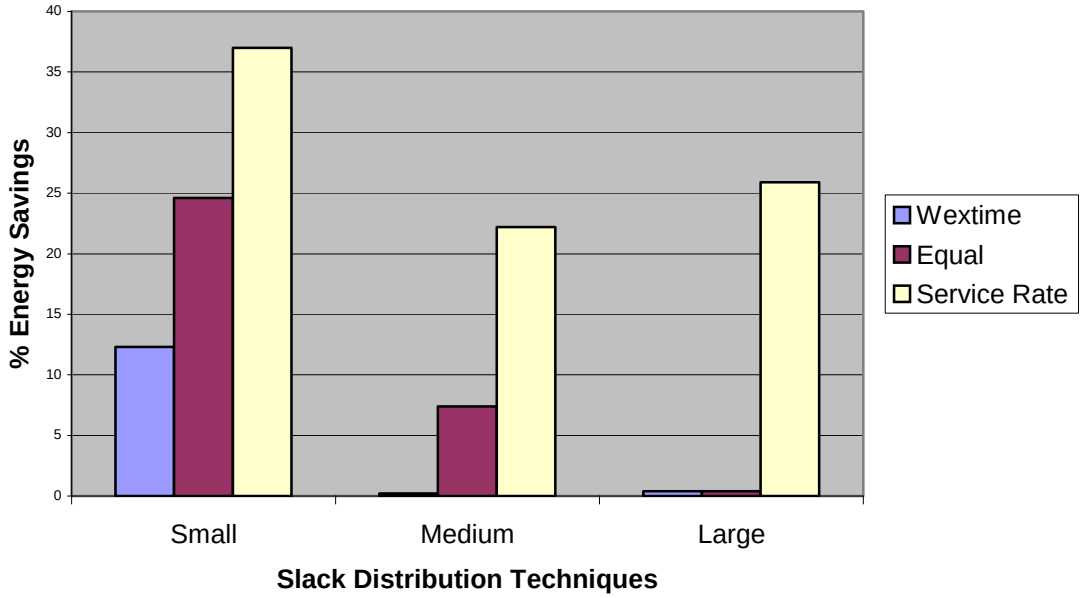

Figure 16: Comparison of energy savings with various slack distribution approaches

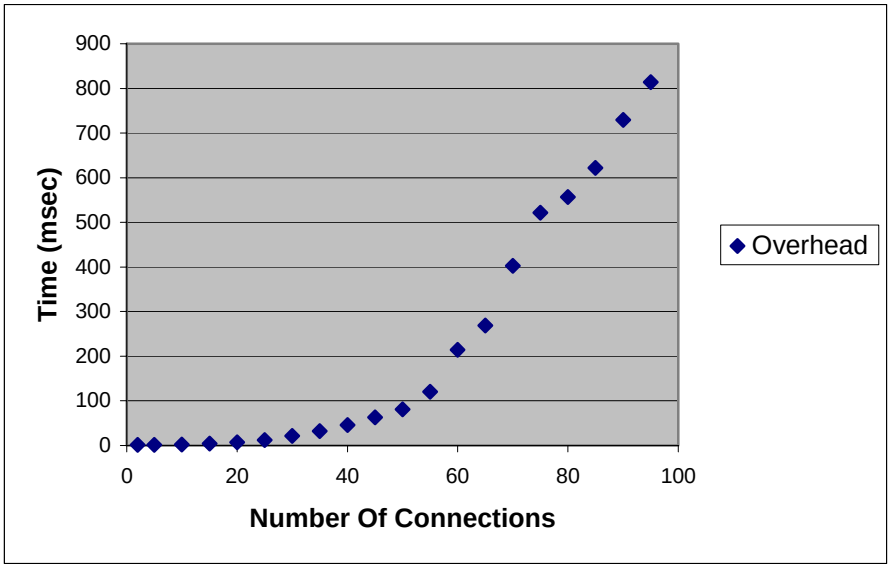

Figure17: Connection setup overhead vs. number of connections

The overhead of connection establishment procedure is shown in Figure 17. The overhead increases slightly with the number of connections initially and after a certain point the overhead increases drastically with the slight increase in the number of connections. This overhead is measured on Xscale processor. The overhead of periodic service rate determination is shown in Figure 18. As shown in the above figure, the overhead of periodic service rate determination grows in a "super linear" fashion with the size of the incoming connection set at the node. Even though the overhead of periodic service rate determination is negligible for a large incoming connection set (size < 35), it will have a significant impact on the system with a very large incoming connection set. The "super linear" nature of periodic service rate determination is due to the fact that the node has to deal with "large" number of messages and computations associated with them as the size of the incoming connection set increases. The overhead of periodic service rate determination may also become a limiting factor in connection

acceptance for a system with huge incoming connection set having very small processing time demand.

The comparison of energy savings with all nodes employing FCFS scheduling policy and all nodes employing WRR scheduling policy is shown in Figure 19 on synthetic II test case. The advantage of the FCFS scheduling policy is it is easy to implement and has a lesser overhead in determining the clock frequency periodically compared with the WRR scheduling policy. With WRR scheduling policy, the periodic service rate determination algorithm will be called on each individual connection separately at the beginning of each interval. This means that the overhead for determining the service rate with WRR scheduling policy is roughly the amount, which is the size of the incoming set, multiplied by the overhead of determining the service

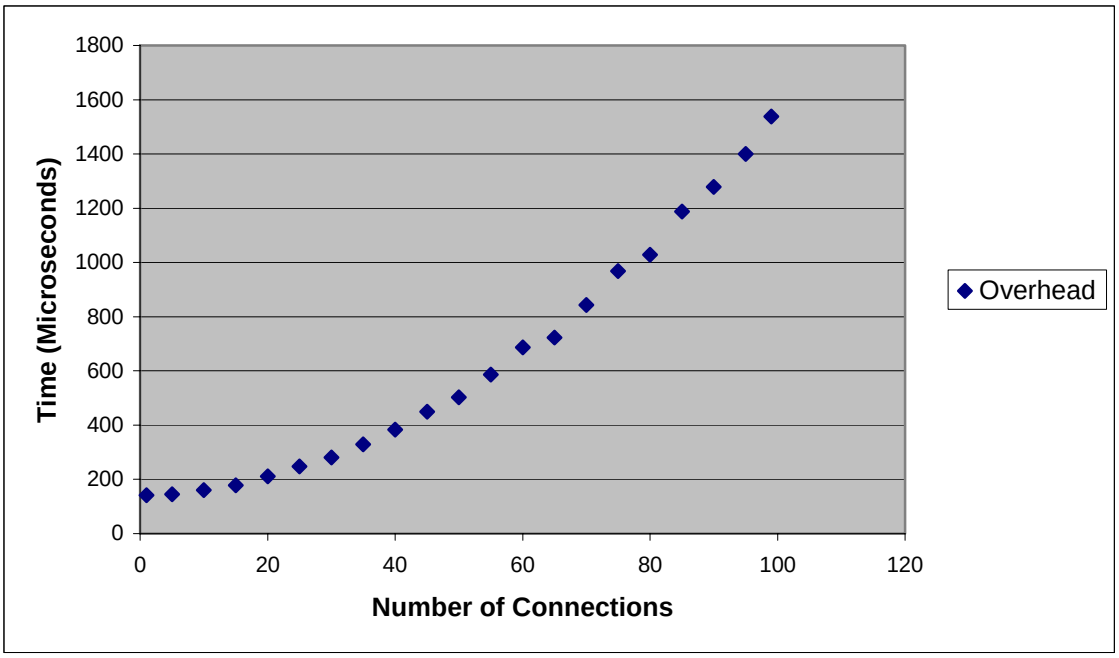

Figure 18: Overhead of periodic service rate determination

rate with FCFS scheduling policy. Even though the "WRR' scheduling policy yields better energy savings for nodes with smaller incoming connection set, its benefits will

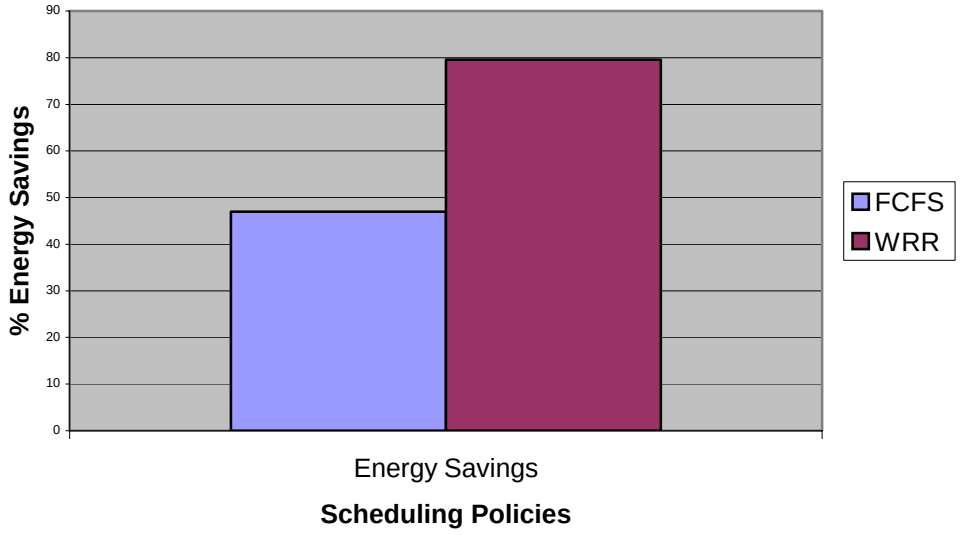

Figure 19: Energy savings with all nodes employing either fcfs or wrr scheduling policy

become very limited as the size of the incoming connection set grows. This is partially because of the "super linear" nature of overhead in determining the periodic service rate. Figure 20 presents the energy savings obtained by the proposed technique on various real-world and synthetic examples. These measurements are taken using Xscale embedded processor clock speed and voltage level specifications. As seen in the below figure, the proposed technique yields significant energy savings.

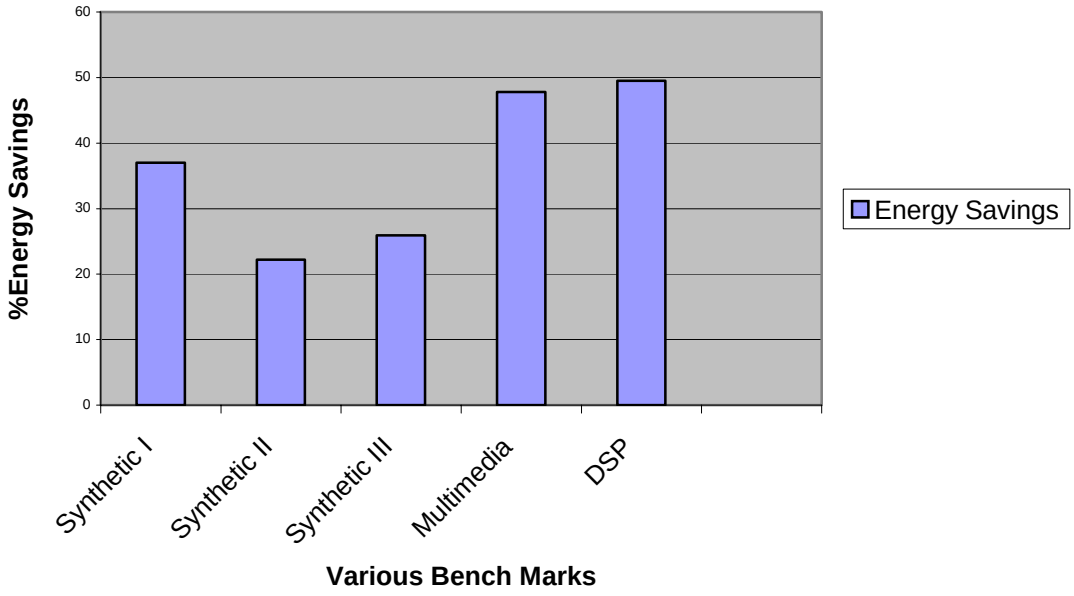

Figure 20: % Energy Savings with the proposed technique on various test cases

G. CONCLUSIONS

In this work, an energy efficient technique that address the issue of guaranteeing the end-to-end and local deadlines of the messages in real-time distributed embedded systems is presented. The proposed techniques use a comprehensive traffic description function that provides adequate information about the worst-case traffic behavior anywhere in the network. The service rate of a node is reduced to take advantage of the fact that the processing of the messages can be delayed or extended up to their worst-case delays. Further, the traffic pattern is periodically analyzed to determine the best possible service rate to take advantage of the run time variations in computation demands. Experimental results showed that the proposed techniques lead to considerable energy savings.

CHAPTER V

FUTURE WORK

A. SINGLE PROCESSOR

In this work, an energy efficient scheduling technique for single processor real-time embedded systems has been presented. This technique takes into account periodic task graphs and sporadic tasks with hard deadlines. However, the technique does not address on how to deal with soft aperiodic tasks. Soft aperiodic tasks do not have deadlines; however the response time for soft aperiodic tasks should be low for the system to be used. The proposed technique can be extended to soft aperiodic tasks.

B. MULTI PROCESSOR

The proposed technique for multi processor is only applicable to a canonical chain of tasks executing on different nodes. The approach does not address the task graphs containing tasks with more than one predecessor or one successor. It seems to be difficult to incorporate these kinds of task graphs in the existing model. The proposed system model can be extended to incorporate the above said task graphs.

CHAPTER VI

CONCLUSION

With the advent of portable systems, low power design has become a major consideration in the design of today's applications. As an increasing amount of system functionality tends to be realized through software, system-level power reduction techniques, especially at operating systems level, are becoming important. There have been extensive studies in the literature regarding low power system design. Still, for many years to come, the search for even better low power system-level techniques will continue. In this thesis, novel energy efficient techniques for the scheduling of real-time tasks in single and multi processor embedded systems are presented. The performance of these techniques has been evaluated on real-life benchmarks on an XScale embedded processor specifications. Experimental results showed that the proposed techniques yield significant energy savings.

REFERENCES

[1] K.Lahiri, A.Raghunathan, S. Dey and D. Panigraphi, "Battery-Driven System Design: A New Frontier in Low Power Design," *Proc. Int'l Conf. Very Large Scale Integration Design,* pp. 261-267, January 2002.

[2] Y.Shin and K.Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," *Proc. Int'l Design Automation Conference*, pp.134-139, June 1999.

[3] G.Fohler, "Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems," *Proc. Int'l Symp. Real-time Systems*, pp.152-161, December 1995.

[4] J.Luo and N.K.Jha, "Battery-Aware Static Scheduling for Distributed Real-Time Embedded Systems," *Proc. Int'l Design Automation Conference*, pp. 444-449, June 2001.

[5] G.Quan and X.Hu, "Energy Efficient Fixed Priority Scheduling for Real-Time Systems on Variable Voltage Processors," *Proc. Int'l Design Automation Conference,* pp.828 –833, June 2001.

[6] T.Okuma, T.Ishihara and H.Yasuura, "Real-Time Task Scheduling for a Variable Voltage Processor," *Proc. Int'l Symp. System Synthesis*, pp. 24-29, November 1999.

[7] J.Luo and N.K.Jha, "Power-Conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-Time Embedded Systems," *Proc. Int'l*

Conf. Computer-Aided Design, pp.357-364, November 2000.

[8] J.Pouwelse, K.Langendoen and H.sips, "Energy Priority Scheduling for Variable Voltage Processors," *Proc. Int'l. Symp. Low Power Electronics and Design*, pp. 28-33, August 2001.

[9] Y.Shin, K.Choi and T.Sakurai, "Power Optimization of Real-Time Embedded Systems on Variable Speed Processors," *Proc. IEEE/ACM Int'l Conf. Computer Aided Design*, pp. 365-368, November 2000.

[10] V.Swaminathan and K.Chakrabarty, "Real-Time Task Scheduling for Energy-Aware Embedded Systems," *Proc. IEEE Int'l Symp. Real-Time Systems (RTSS) – Work-in-Progress Session,* November 2000.

[11] I. Hong, M. Potkonjak and M. B. Srivastava, "On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor," *Proc. Int'l Conf. Computer-Aided Design (ICCAD)*, pp. 653-656, November 1998.

[12] L. Benini, G. Castelli, A. Macii, and R. Scarsi, "Battery-Driven Dynamic Power Management," *IEEE Design and Test of Computers*, vol. 18, no. 2, pp. 53-60, April 2001.

[13] A. Manzak and C. Chakrabarti, "Variable Voltage Task Scheduling Algorithms for Minimizing Energy," *Proc. Int'l Symp. Low Power Electronics and Design*, pp. 279-282, August 2001.

[14] C. Locke, D. Vogel, and T. Mesler, "Building a Predictable Avionics Platform in

Ada: A Casestudy," *Proc. IEEE Int'l Symp. Real-Time Systems*, pp. 181-189, December 1991.

[15] A. Burns, K. Tindell, and A. Wellings, "Effective Analysis for Engineering Real-Time Fixed Priority Schedulers," *IEEE Trans. Software Eng*., vol. 21, no. 5, pp. 475–480, May 1995.

[16] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin, "Visual Assessment of a Realtime System Design: A Case Study on a CNC Controller," *Proc. IEEE Int'l Symp. Real-Time Systems*, pp. 300-310, December 1996.

[17] C.L.Liu and Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *ACM Journal*, vol.20, no. 1, pp. 46-61, January 1973.

[18] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *IEEE Journal of Solid State Circuits,* vol. 31, no. 9, pp. 1277-1284, September 1996.

[19] T. Okuma, H. Yasuura, and T. Ishihara, "Software Energy Reduction Techniques for Variable Voltage Processors," *IEEE Design & Test of Computers,* vol. 18, no. 2, pp. 31-41, March 2001.

[20] W. Nebel, and J. Mermet, "*Low Power Design in Deep Submicron Electronics*," Boston, Massachusetts: Kluwer Academic Publishers, 1997.

[21] J. Luo, and N. Jha, "Static and Dynamic Variable Voltage Scheduling Algorithms

for Real-Time Heterogeneous Distributed Embedded Systems," *Proc. ASP-DAC*, pp. 719- 728, January 2002.

[22] D.Zhu, R. Melhem, and B. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi Processor Real-Time Systems," *IEEE Trans. On Parallel and Distributed Systems*, vol.14, no. 7, pp. 686-700, December 2001.

[23] M.T.Schmitz, and B.M. Al-Hashimi, "Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems," *Proc. Int'l Conf. Design, Automation and Test in Europe,* pp. 514 -521, March 2002.

[24] S. Hua, G. Qu, and S. Bhattacharyya, "Energy Reduction Techniques for Multimedia Applications with Tolerance to Deadline Misses," *Proc. Int'l Design Automation Conference*, pp. 131-136, June 2003.

[25] J. Luo, and N. Jha, "Power-Profile Driven Variable Voltage Scaling for Heterogeneous Distributed Real-Time Embedded Systems," *Proc. Int'l Conf. Very Large Scale Integration Design*, pp. 369-375, January 2003.

[26] R. Mishra, N. Rastogi, and D. Zhu, "Energy Aware Scheduling for Distributed Real-Time Systems," *Proc. Int'l Symp. Parallel and Distributed Processing*, pp. 21-29, April 2003.

[27] W.C. Kwon, and T. Kim, "Optimal Voltage Allocation Techniques for

Dynamically Variable Voltage Processors," *Proc. Design automation Conference*, pp. 125-130, June 2003.

[28] D. Rakhmatov, and S. Vrudhula, "Battery-Conscious Task Sequencing for Portable Devices Including Voltage/Clock scaling," *Proc. Design Automation Conference*, pp. 189-194, June 2002.

[29] C. Hughes, J. Srinivasan, and S. Adve, "Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications," *Proc. Int'l Symp. Micro Architecture,* pp. 250-261, December 2001.

[30] C. Hwang, and A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event Driven Computation," *Proc. Int'l Conf. Computer Aided Design*, pp. 28-32, November 1997.

[31] J. W. S. Liu, "*Real-Time Systems*," Upper Saddle River, New Jersey: Prentice Hall Publishers, 2000.

[32] A. Raha, N. Malcolm, and W. Zhao, "Guaranteeing End-To-End Deadlines in ATM Networks," *Proc. Int'l Conf. Distributed Computing Systems*, pp.60-68, May 1995.

APPENDIX A

INTEL STRONG ARM SA-1100 CLOCK SPEED SPECIFICATIONS

The core clock frequency is configured by software through the core clock configuration field (CCF<4:0>) in the power manager phase-locked loop (PLL) configuration register (PPCR). This field should be programmed during the boot sequence for the desired full-speed operation.

Table 10: Intel SA-1100 Core Clock Configuration

CCF<4:0>	3.864 MHz Oscillator	3.5795 MHz Oscillator
00000	59.0	57.3
00001	73.7	71.6
00010	88.5	85.9
00011	103.2	100.2
00100	118.0	114.5
00101	132.7	128.9
00110	147.5	143.2
00111	162.2	157.5
01000	176.9	171.8
01001	191.7	186.1
01010	206.4	200.5
01011	221.2	214.8
01100-11111	Not supported	--

APPENDIX B

INTEL PXA250 XSCALE CLOCK SPEED SPECIFICATIONS

Table 11: Intel PXA250 Core Clock Specification

Range	Frequency	Voltage
Low Voltage Range	132.7 MHz	0.935 V
Medium Voltage Range	199.1 MHz	1.1 V
High Voltage Range	298.7 MHz	1.21 V
Peak Voltage Range	398.2 MHz	1.43 V

VITA

Rajesh Babu Prathipati received his undergraduate degree in computer science and engineering in 2000 from Regional Engineering College, Warangal, India. He worked as a Software Engineer at Nortel Networks, where he developed software for second generation wireless networks. He began his study for his Master of Science degree in computer science at Texas A&M University in 2001. He received his Master of Science degree in May 2004.

Permanent Address

P.V.V. SubbaRayudu

5-3-71, Opp: T.V. Station

Amalapuram - 533201

INDIA

The typist for this thesis was Rajesh Prathipati.