Leakage Aware Dynamic Voltage Scaling for Real Time Embedded Systems

Ravindra Jejurikar Cristiano Pereira Rajesh K. Gupta

Center for Embedded Computer Systems,
Department of Information and Computer Science,
University of California at Irvine,
Irvine, CA 92697
E-mail: jezz@ics.uci.edu, {cpereira,gupta}@cs.ucsd.edu

CECS Technical Report #03-35

November 30, 2003

Abstract

Traditionally, dynamic voltage scaling techniques have been designed to minimize the dynamic power consumption, which has been the dominant factor. As the technology scales, leakage current is contributing to significant energy consumption. In this paper, we propose task scheduling techniques that take leakage into account to minimize the total energy consumption. We compute an operating point called the critical speed *which minimizes the dynamic and leakage energy consumption per unit work. The leakage energy dominates when operating at speeds lower than the critical speed and it is energy efficient to execute faster and shutdown the system. Due to the time and energy cost associated with shutdown, longer shutdown intervals are better. To address this issue, we also present a scheduling procrastination scheme, which delays task execution to extend sleep intervals. Our simulation experiments show on an average 20% energy gains over a leakage oblivious dynamic voltage scaling and the procrastination scheme increases the gains to up to* 35%. *Our scheduling scheme extends the sleep intervals to up to* 5 *times while meeting all timing requirements.*

Contents

List of Figures

List of Tables

1 Introduction

Power management is of primary importance in the operation of embedded systems, which can be attributed to longer battery life, reliability and packaging costs. Power consumption of a device is broadly classified as dynamic power consumption which arises due to switching energy and static power consumption which is present even when no logic operations are performed. CMOS has emerged as a dominant technology because of its low static power consumption. CMOS device scaling trends, driven by the need for faster devices and higher transistor densities, show a 30% decrease in the device dimensions with each technology generation [5]. Constant electric field scaling allows a proportional reduction of supply voltage. As supply voltage is reduced, the threshold voltage (V_{th}) must be proportionately reduced to maintain the desired performance (delay) improvements. *This reduction of threshold voltage results in an exponential increase in the subthreshold leakage current [6], leading to larger standby current*.

Leakage current in CMOS circuits contribute to a significant portion of the total power consumption and is becoming an increasing concern. The subthreshold leakage current is $0.01\mu A/\mu m$ for the $130nm$ and is projected to be $3\mu A/\mu m$ for the $45nm$ technology [1]. A five fold increase in the leakage power is predicted with each technology generation [5]. The static power consumption is comparable to the dynamic power dissipation and projected to surpass it if measures are not taken to minimize leakage current [9]. Furthermore, leakage has an adverse effect with the increase in temperature [29].

To address this issue, efforts at process, circuit design and micro-architecture level are made to minimize leakage power. The exponentially dependence of subthreshold leakage current on the threshold voltage has led to threshold voltage scaling. Scaling the threshold voltage by controlling the body bias voltage [23, 21] has been proposed to minimize leakage. Multi threshold CMOS (MTCMOS) [7] is a popular technique to reduce standby current. Other techniques such as input vector control [14] and power supply gating [22] have been proposed. At higher levels of abstraction, recent works have focused on minimizing the leakage of components such as cache. Techniques like cache decay [11] and turning off cache lines [10] reduce have shown effective results in reducing cache leakage. Clock gating techniques are also used to control leakage in Systems on Chip (SoC). The IBM PowerPC 405LP [8] implements clock gating at the IP core and register level. The Intel PXA [12] family processors also support fine granularity clock gating to exploit the fact that not all system transistors are used at the same time. The chip aggressively shuts down elements of the processor which are idle by gating them off or disabling their input. Processors support various shutdown mode to save power. For examples, the Transmeta Crusoe [28] processor support various sleep modes (normal, autohalt, quick start, deep sleep, off) for various types of workload. These power states may be used to reduce the operating power of the processor during systems states that require little or no CPU activity.

Dynamic voltage scaling (DVS) [25, 24, 26, 3, 4, 15] based on performance requirements reduces the power consumption and can lead to significant energy gains Recently, techniques to optimize the total static and dynamic power consumption have been proposed in [16, 21]. Note that the energy savings based on DVS come at the cost of increased execution time. This implies that the devices will be on for a longer time duration and result in greater leakage energy consumption. With the steep increase in leakage current per generation, it is not obvious whether to perform DVS or to execute the system at maximum speed and shutdown. Note that the slowdown resulting from frequency scaling decreases the dynamic energy whereas increases leakage energy. It is important to select the correct operation point to minimize the total energy. Thus we have to judiciously balance the extent of slowdown and shutdown to minimize the total energy consumption. As shown later, operating at the maximum or minimum possible voltage (frequency) need not be the optimal point. Furthermore the additional time and energy cost of shutdown makes the problem harder.

Irani *et al.* [13] consider the combined problem of DVS and shutdown and propose a *3-competitive* off-line algorithm. They introduce the concept of critical speed which minimizes the energy per unit workload. They present a theoretical result, based on the assumption of a continuous voltage range, for any given convex power consumption function. Lee *et al.* propose an Leakage Control EDF (LC-EDF) [18] scheduling algorithm to minimize the leakage energy consumption in real time systems. They propose delaying task executions to extend idle intervals. Their algorithm is based on the assumption that the tasks are executed at the maximum speed and the processor is shutdown. However, it may not be energy efficient to execute at the maximum speed. We enhance this work by combining dynamic voltage scaling with shutdown to minimize the total energy consumption. Our contributions are as follows: Firstly, based on the leakage characteristics of the $0.07\mu m$ technology, we compute the *critical speed* for the system. Furthermore, if this assigned speed leaves idle intervals, we compute the time interval by which task executions can be delayed to extend the length of the idle periods to have more opportunity to shutdown the processor. Our work differs from that in [18], in that we compute delays given slowdown for the tasks. Our algorithm is simple to implement and has a very low run-time overhead compared to LC-EDF. Furthermore, the minimum idle period guaranteed by our algorithm is always greater than or equal to that by LC-EDF.

The rest of the paper is organized as follows: Section 2 and 3 discusses the leakage power model and the computation of the critical speed to minimize energy consumption. In Section 4, we present the procrastination algorithm for EDF scheduling. The experimental results are given in Section 5. Finally, Section 6 concludes the paper with future directions.

2 Power Model

In this section, we describe the power model used to compute the static and dynamic components of power consumption of CMOS circuits. The dynamic power consumption (P_{AC}) of CMOS circuits is given by,

$$P_{AC} = C_{eff} V_{dd}^2 f \tag{1}$$

where V_{dd} is the supply voltage, f is the operating frequency and C_{eff} is the effective switching capacitance. Dynamic voltage scaling reduces the dynamic power consumption due to its quadratic dependence on voltage.

Different leakage sources [2] contribute to the total leakage in a device. The major contributors of leakage are the subthreshold leakage and the reverse bias junction current which can increase significantly with adaptive body biasing [21]. We use the power model and the technology parameters described by Martin *et al.* [21]. The threshold voltage V_{th}, subthreshold current I_{subn}, and cycle time t_{inv} as a function of the supply voltage V_{dd} and the body bias voltage V_{bs} are given below :

$$V_{th} = V_{th1} - K_1 \cdot V_{dd} - K_2 \cdot V_{bs} \tag{2}$$

where K_1, K_2 and V_{th1} are technology constants.

$$I_{subn} = K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} \tag{3}$$

Table 1. $0.07\mu m$ technology constants [21]

Const	Value	Const	Value	Const	Value
K_1	0.063	K_6	$5.26x10^{-12}$	V_{th1}	0.244
K_2	0.153	K_7	-0.144	I_j	$4.8x10^{-10}$
K_3	$5.38x10^{-7}$	V_{dd0}	1	C_{eff}	$0.43x10^{-9}$
K_4	1.83	V_{bs0}	0	L_d	37
K_5	4.19	α	1.5	L_g	$4x10^6$

where K_3, K_4 and K_5 are constant fitting parameters.

$$t_{inv} = \frac{L_d K_6}{(V_{dd} - V_{th})^{\alpha}} \tag{4}$$

The leakage power dissipation due to subthreshold leakage (I_{subn}) and reverse bias junction current (I_j) is given by,

$$P_{DC} = V_{dd} I_{subn} + |V_{bs}| I_j \tag{5}$$

This is the leakage per device and the total leakage power consumption is $L_g \cdot P_{DC}$, where L_g is the number of devices in the circuit.

The technology constants for the $0.07\mu m$ technology are presented in Tables 1 as given in [21]. The value for C_{eff} based on the Transmeta Crusoe processor, scaled to $0.07\mu m$ technology based on the technology scaling trends [5], is also given in the table. To reduce the leakage substantially, we use $V_{bs} = -0.7V$. The static and dynamic power consumption as the supply voltage is varied in the range of $0.5V$ and $1.0V$ is shown in Figure 1.

3 Critical Speed

There is an inherent cost in keeping the processor on, which must be taken into consideration in computing the optimal operating speed. In addition to the gate level leakage, there are certain processor components which consume power even when the processor is idle. Some of the major contributors are (1) the PLL circuitry, which drives up to $200mA$ current [12, 28]. (2) the I/O power supply V_{IO} has a higher voltage supply (2.5V to 3.3V) that the processor core with peak currents of $400mA$ during I/O. Though the current is comparatively lower when there is no I/O, the power consumption adds to a significant portion of the idle power consumption. This intrinsic power cost of keeping the system on is referred to as P_{on}. The power consumption of these components will scale with technology and architectural improvement and we assume a conservative value of $P_{on} = 0.1W$. The total power consumption by the processor, P, is :

$$P = P_{AC} + P_{DC} + P_{on} \tag{6}$$

where P_{AC} and P_{DC} are the dynamic and static power consumptions. The variation of the power consumption with supply voltage is shown in Figure 1. It can be seen that the total power consumption

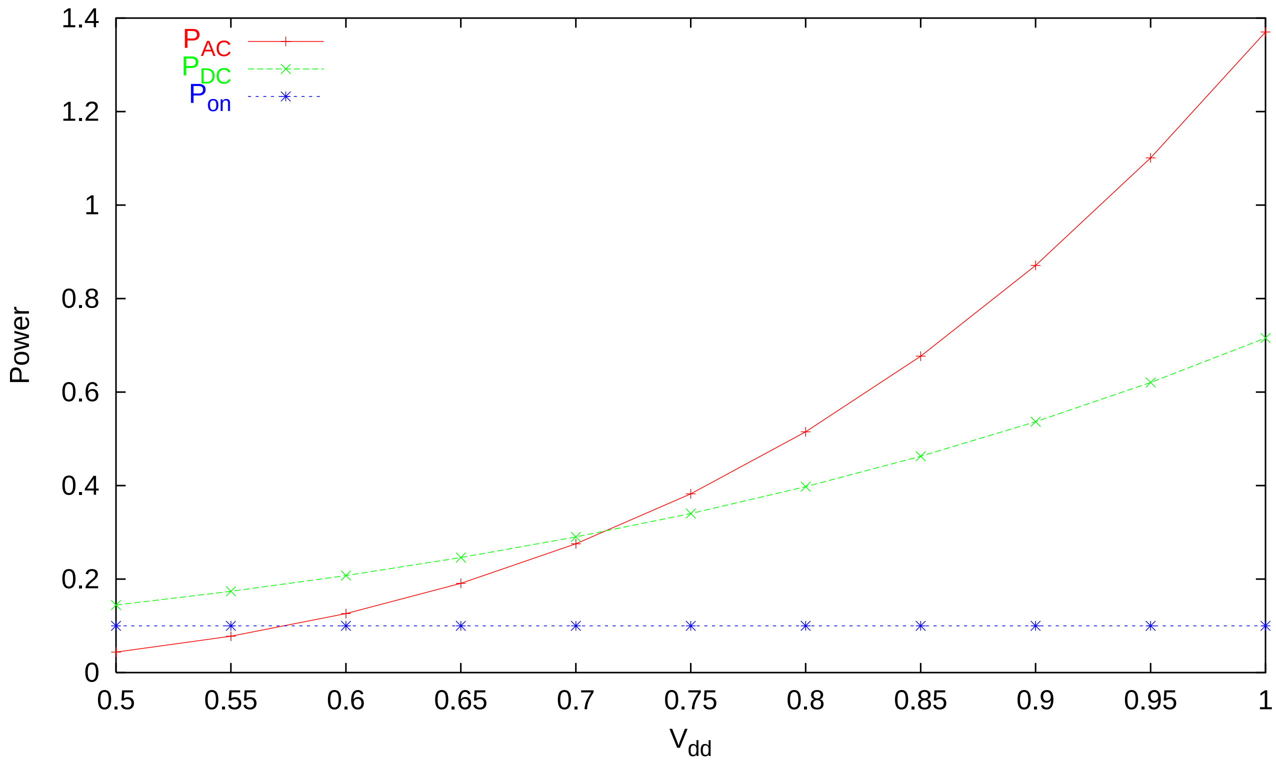

Figure 1. Power consumption of $0.07\mu m$ technology for Crusoe processor: P_{DC} is the leakage power, P_{AC} is the dynamic power and P_{on} is the intrinsic power consumption in on state

decreases as V_{dd} is scaled. The linear dependence of static power consumption on voltage and the quadratic dependence of dynamic power on voltage is seen in the figure.

To evaluate the effectiveness of dynamic voltage scaling, we compute the energy consumption per cycle for different supply voltage values. Due to the decrease in the operating frequency with voltage, the leakage can adversely effect the total energy consumption with voltage scaling. We compute the energy per cycle to decide the aggressiveness of voltage scaling. The contribution of the dynamic energy per cycle is given by,

$$E_{AC} = C_{eff} V_{dd}^2 \tag{7}$$

The leakage power per device is given by Equation 5. Since the cycle time increases as voltage decreases, the leakage energy per cycle is given by,

$$E_{DC} = f^{-1} \cdot L_g \cdot (I_{subn} V_{dd} + |V_{bs}| I_j) \tag{8}$$

where f^{-1} is the delay per cycle. The energy to keep the system on increases with lower frequencies and is given by, $E_{on} = f^{-1} P_{on}$. The total energy consumption per cycle, E_{cycle}, with varying supply voltage levels is given below and shown in Figure 2 .

$$E_{cycle} = E_{AC} + E_{DC} + E_{on} \tag{9}$$

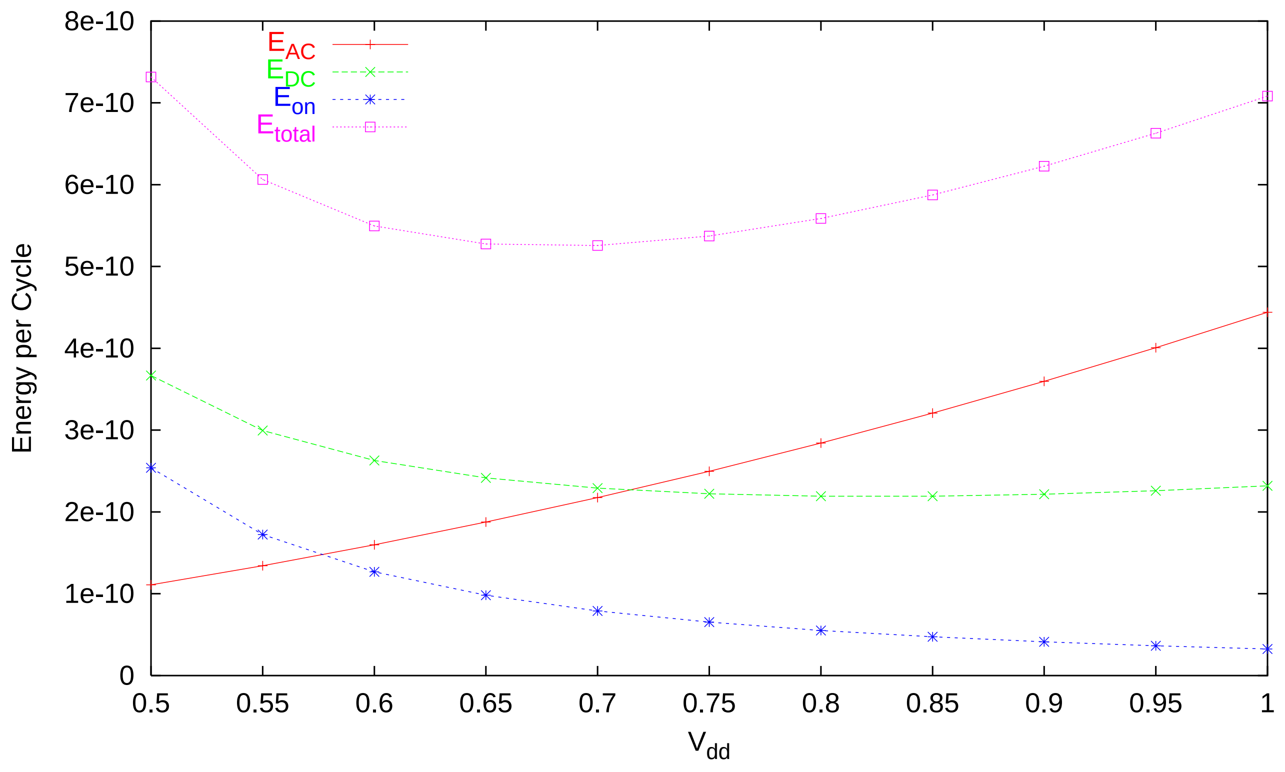

Figure 2. Energy per Cycle for $0.07\mu m$ technology for the Crusoe processor: E_{AC} is the switching energy, E_{DC} is the leakage energy and E_{on} is the intrinsic energy to keep the processor on.

We define the *critical speed* as the operating point that minimizes the energy consumption per cycle. We can compute the minimum of this energy function by evaluating the gradient of the energy function with respect to V_{dd}. Figure 2 shows the energy characteristics for the $0.07\mu m$ technology. From the figure, it is seen that the critical point is at $V_{dd} = 0.70V$. From the voltage frequency relation described in Equation 4, $V_{dd} = 0.7V$ corresponds to a frequency of 1.26 GHz. The maximum frequency at $V_{dd} = 1.0V$ is 3.1 GHz, resulting in a critical slowdown of $\eta_{crit} = 1.26/3.0 = 0.41$. It can be seen that it is not energy efficient to scale the voltage below $V_{dd} = 0.7V$. Executing at the critical speed and shutting down the system is more energy efficient than executing at voltages lower than $0.7V$.

4 Real Time Scheduling

In this section, we enhance real time scheduling techniques with the knowledge of critical speed η_{crit}, to minimize the total energy consumption of the system.

4.1 Background

In a classical real-time system model, tasks arrive periodically and have deadlines. A task set of n periodic real time tasks is represented as $\Gamma = \{\tau_1, \ldots, \tau_n\}$. A 3-tuple $\tau_i = \{T_i, D_i, C_i\}$ is used to represent each task τ_i, where T_i is the period of the task, D_i is the relative deadline with $D_i \leq T_i$, and C_i is the worst case execution time (WCET) for the task at maximum speed. These tasks are to be scheduled on a single processor system based on a preemptive scheduling policy. In this work, we assume task deadlines are equal to the period ($D_i = T_i$) and the tasks are scheduled by the Earliest Deadline First (EDF) [19] scheduling policy. All tasks are assumed to be independent and preemptive.

Dynamic voltage scaling (DVS) have been proposed to minimize the dynamic power consumption in real time systems. A task slowdown factor is the extent of slowdown that can be applied while meeting specified performance requirements. A task *slowdown factor* (η_i) can be viewed as the normalized operating frequency and lies in the range [0,1]. Under EDF scheduling, a slowdown equal to the task utilization is the optimal slowdown to minimize the dynamic energy consumption [4].

4.2 DVS and Critical Speed

The task slowdown can be computed with any known dynamic voltage scaling algorithm. We assume that the system utilization is assigned as the slowdown for each task. Since executing below the critical speed consumes more time and energy, we set the minimum value for the slowdown factor as the critical speed (η_{crit}). We update a task slowdown factor to the critical speed if it is smaller than η_c. The algorithm is as follows:

$$\overset{\forall i}{i = 1, \ldots, n} \quad if(\eta_i < \eta_{crit})\ \eta_i \leftarrow \eta_{crit} \tag{10}$$

Since we are only increasing slowdown factors of a given feasible task set, the feasibility of the task set is maintained.

4.3 Shutdown Overhead

In previous work, the overhead of processor shutdown/wakeup has been neglected or considered only as the actual time and energy consumption incurred within the processor. However, a processor shutdown and wakeup has a higher overhead. The Intel PXA processor family [12], when switched to the deepest sleep mode, loses its registers and cache contents. Thus, all the registers have to be saved in main memory and all the dirty data cache lines have to be flushed to main memory before shutdown. This memory access results in an additional energy overhead. Waking up from the sleep state also incurs a considerable overhead. First there is an inherent energy delay cost of wakeup as specified in the datasheets. In addition, components such as data and instruction caches, data and instruction translation look aside buffers (TLBs) and branch target buffers (BTBs) have to be initialized, resulting in cold start misses in case of the caches and TLBs, and branch mispredictions in case of the BTBs. These result in an additional energy cost. This cost will vary depending on the nature of the application and the processor architecture.

4.4 Procrastination Algorithm

Due to the cost of shutdown, we have to make a decision whether to shutdown or not. An unforeseen shutdown can result in extra energy and/or missing task deadlines. Based on the idle power consumption, we can compute the minimum idle period, referred to as the *idle threshold* interval $t_{threshold}$, to break even with the wakeup energy overhead. Let P_{idle} be the power consumption in the idle state in addition to the power consumption in the shutdown state. If $t_{shutdown}$ and $E_{shutdown}$ are the time and energy overhead incurred due to shutdown (the overhead of shutdown as well as wakeup), then $t_{threshold}$ is given by:

$$P_{idle} \cdot t_{threshold} = E_{shutdown}$$

Thus, it is energy efficient to shutdown if the idle interval is greater than $t_{threshold}$. Thus longer idle interval increase the changes of shutdown, thereby saving energy. We present a procrastination scheme to achieve this goal.

4.4.1 Computing Procrastination Interval

First, we give the EDF feasibility condition with task slowdown. Given n independent periodic tasks, the task set is feasible at a slowdown factor of η_i for task τ_i if,

$$\sum_{i=1}^{n} \frac{1}{\eta_i} \frac{C_i}{T_i} \leq 1 \tag{11}$$

We compute a procrastination interval for each task while maintaining feasibility. The procrastination interval, Z_i, for task τ_i is the time interval by which the τ_i can be delayed while guaranteeing all task deadlines. The computation of Z_i is given by Lemma 1. The detailed explanation of all results are given in [27].

Lemma 1 *[27] Given tasks are ordered in non-decreasing order of their period, a procrastination of task τ_i by Z_i time units guarantees the deadlines of all tasks if,*

$$\overset{\forall i}{i = 1, \ldots, n} \quad \frac{Z_i}{T_i} + \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{T_k} \leq 1 \tag{12}$$

$$\forall_{k<i} \;\; Z_k \leq Z_i \tag{13}$$

4.4.2 Algorithm

It is assumed that the *power manager* which handles task procrastination is implemented in a FPGA. When the processor enters sleep state, it handles over the control to the power manager (FPGA controller), which handles all the interrupts and task arrivals while the processor is in sleep state. The controller has a timer to keep track of time and wake the processor after a specified time period. The procrastination algorithm is shown in Figure 3. When the processor is sleep state and the first task τ_i arrives, the timer is set to Z_i. The timer counts down every clock cycle. If another task arrives before the counter expires, the counter is adjusted based on the new task arrival. If another task τ_j arrives, then the timer is updated to the minimum of the current timer value and Z_j. This ensures that no task τ_k in the system lets the processor be in sleep state for more than Z_k time units after its arrival. When the counter counts down to zero (expires), the processor is woken up and the scheduler schedules the highest priority task in the system. All tasks are scheduled at their assigned slowdown factor.

```
    On arrival of a new job $J_i$ :
(1)  if (processor is in sleep state)
(2)      if (Timer is not active)
(3)          $timer \leftarrow Z_i$; //(Initialize timer)
(4)      else
(5)          $timer \leftarrow min(timer, Z_i)$;
(6)      endif
(7) endif
    On expiration of Timer $(timer = 0)$:
(1) Wakeup Processor;
(2) Scheduler schedules highest priority task;
(3) Deactivate timer;
    Timer Operation :
(1) timer--;
        // Counts down every clock cycle;
```

Figure 3. Procrastination Algorithm

Theorem 2 *[27] Given tasks are ordered in non-decreasing order of their period, the procrastination algorithm guarantees all task deadlines if the procrastination interval Z_i of each task τ_i satisfies:*

$$\overset{\forall i}{i = 1, \ldots, n} \quad \frac{Z_i}{T_i} + \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{T_k} \leq 1 \tag{14}$$

$$\forall_{k<i} \;\; Z_k \leq Z_i \tag{15}$$

We also compute the minimum idle period guaranteed by the procrastination algorithm. This idle period helps make better shutdown decisions.

Lemma 3 *The minimum idle period guaranteed by the procrastination algorithm is is given as,*

$$Z_{min} = min_{1 \leq i \leq n} \quad \left\{ Z_i = (1 - \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{T_k}) T_i \right\} \tag{16}$$

We also compare our algorithm to LC-EDF [18]. Since the LC-EDF algorithm assumes all tasks execute at maximum speed, we prove that our proposed algorithm guarantees more procrastination than LC-EDF if tasks are executed at maximum speed.

Lemma 4 *[27] Given, tasks are executed at maximum speed, the minimum delay interval guaranteed by the procrastination algorithm is greater than or equal to that guaranteed by LC-EDF.*

5 Experiments

We implemented the different scheduling techniques using a discrete event simulator. To evaluate the effectiveness of our scheduling techniques, we consider several task sets, each containing up to 20 randomly generated tasks. We note that such randomly generated tasks is a common validation methodology in previous works [4, 18, 26]. Based on real life task sets [20], tasks were assigned a random period and WCET in the range [10 ms,125 ms] and [0.5 ms, 10 ms] respectively. All tasks are assumed to execute up to their WCET. We use the processor power model described in Section 2. The critical speed for this processor is $\eta_{crit} = 0.41$. We compared the energy consumption for the following techniques presented in the paper:

- No DVS (no-DVS): where all tasks are executed at maximum processor speed.
- Traditional Dynamic Voltage Scaling (DVS) : where tasks are assigned the minimum possible slowdown factor.
- Critical Speed DVS (CS-DVS): where all tasks are assigned a slowdown greater than or equal to the processor critical speed.
- Critical Speed DVS with Procrastination (CS-DVS-P): This is CS-DVS with the procrastination scheduling scheme.

We assume that the processor supports discrete voltage levels in steps of $0.05V$ in the range $0.5V$ to $1.0V$. These voltage levels correspond to discrete slowdown factors and each computed slowdown factor is mapped to the smallest discrete level greater than or equal to it. In all the scheduling schemes except CS-DVS-P, the processor wakes up on the arrival of a task in the system. The idle interval in these techniques is assumed to be the time period before the next task arrival in the system. CS-DVS-P adds the minimum guaranteed procrastination interval to estimate the minimum idle interval. The processor is shutdown if the idle period greater than $t_{threshold}$, the minimum idle period to result in energy gains.

5.1 Shutdown Overhead

An embedded processor like the PXA-250 dissipates almost negligible power in the sleep mode ($180\mu W$), whereas in idle mode it dissipates $555mW$ of power. As discussed in Section 4.3, the cache results in additional energy overhead. The Intel PXA family processors have typically cache sizes of 32KB. We assume 20% lines of the data cache to be dirty before shutdown which results in 6554 memory writes. With an energy cost of 13nJ [17] per memory write, the cost of flushing the data cache is computed as $85\mu J$. We assume the energy and latency of saving the registers to be negligible. On wakeup, there is an additional cost due to cache miss. Note that a context switch occurs when a task resumes execution which has its own cache miss penalty. However, shutdown has its own additional cost than a regular context switch due to the fact that these structures are empty. We assume 10% additional misses rate in both the instruction and data cache. For the TLBs and BTBs, we consider the overhead to be negligible. Therefore, the total overhead of bringing the processor to active mode is 6554 cache misses. A cost of 15nJ [17] per memory access, results in $98\mu J$ overhead. Adding the cache energy overhead to the actual charging of circuit logic, which we assume to be $300\mu J$, the total cost is $85 + 98 + 300 = 483\mu J$. Since the idle power consumption is $240mW$, the threshold idle interval, $t_{threshold}$ is $2ms$. We assume a sleep state power of $50\mu W$, which can account for the power consumption in the sleep state and that of the FPGA controller.

5.2 Energy Consumption

We compare the energy consumption of the techniques discussed in this section and the results are shown in Figure 4. no-DVS consumes the maximum energy and the energy consumption of other techniques are normalized to no-DVS. It is seen that all the techniques perform almost identical up to the critical speed. When the task slowdown factors fall below the critical speed, DVS technique starts consuming more energy due to the dominance of leakage. At lower speeds the energy consumed by DVS approaches close to that of no-DVS. The CS-DVS technique executes at the critical speed and shuts down the system to minimize energy. However if the idle intervals are not sufficient to shutdown, it can consume more energy that the DVS technique, as seen at utilization of 20% and 30%. CS-DVS leads to as much as 20% energy gains over no-DVS and 5% gains over DVS. The CS-DVS-P minimizes idle energy by stretching sleep intervals as much as possible to minimize the shutdown overhead. It is seen that the CS-DVS-P results in an additional 18% gains over CS-DVS.

Figure 5 and 6 compares CS-DVS-P to CS-DVS. Figure 5 shows the number of wakeups and the idle energy comparison of CS-DVS-P normalized to CS-DVS. Note that, since the slowdown factors are mapped to discrete voltage/frequency levels, there are idle intervals at higher utilization as well. These idle period can be used in dynamic reclamation [4] for more energy gains. However, we use these idle intervals to shutdown the processor to compare the benefits of our procrastination scheme. At higher

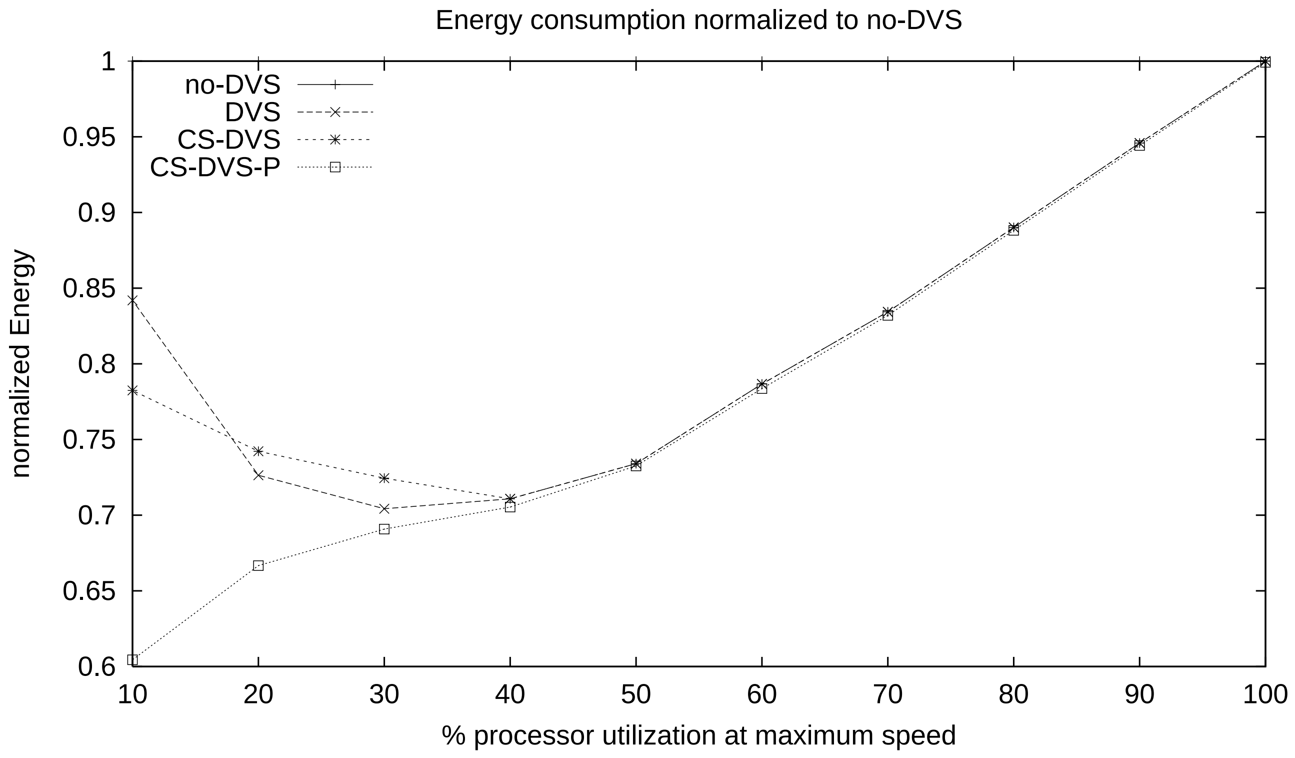

Figure 4. Energy consumption normalized to no-DVS

utilization, the inherent idle intervals are short and there is less chance for a shutdown. However due to procrastination, the idle periods can be extended resulting in energy savings through shutdown. As seen in the Figure 5, the number of shutdowns are higher at higher utilizations which results in reduced idle energy consumption. At lower utilization, the relative number of wake-ups compared to CS-DVS decrease considerably. Note that the idle energy consumption of CS-DVS-P is always lower than that of CS-DVS. The number of wakeup are reduced to as much as 25% percent thereby reducing the shutdown overhead. It is seen from the figure that the idle energy consumption also reduces proportionately.

Figure 6 compares the relative increase of the sleep time intervals of CS-DVS-P over CS-DVS. It is seen that on an average the sleep interval is increased by 4 to 5 times. This extended sleep interval is beneficial as it allows for a shutdown of other peripheral devices that are idle. I/O devices such as memory have a time overhead of 10*ms* to wake up from deep sleep states. This increases the opportunity to shutdown more devices to minimize the total system energy. The figure also compares the average idle interval (intervals when no task is executing i.e. an idle or sleep state). It is seen that the average idle interval increases up to 7 times. This suggests that CS-DVS has relatively more idle intervals where it does not shutdown the processor resulting in leakage energy consumption. CS-DVS-P clusters the task executions thereby increasing the opportunity to shutdown and thereby reduce leakage. This also means the power manager also has to make fewer decisions whether to shutdown.

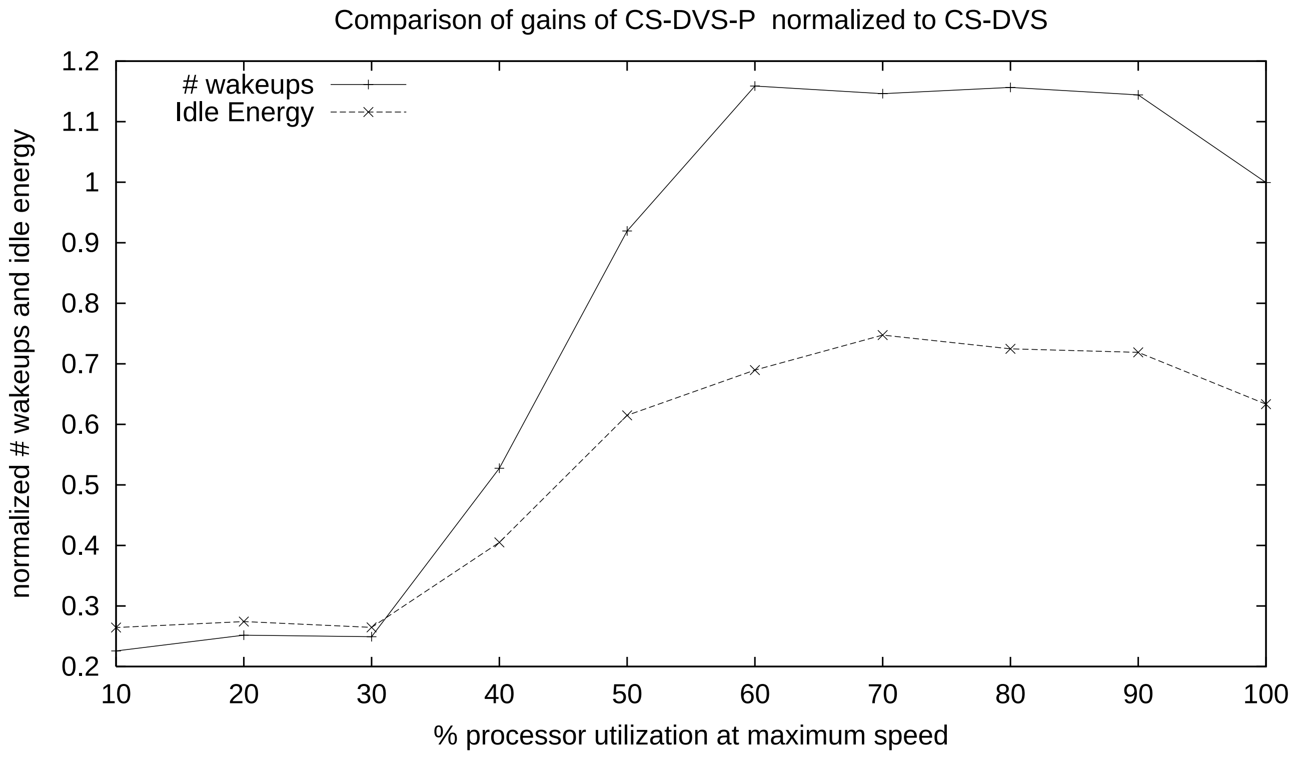

Figure 5. Comparison of # wakeups and idle energy of CS-DVS-P normalized to CS-DVS

6 Conclusions and Future Work

In this paper, we have presented scheduling techniques that consider leakage energy contribution to minimize the total energy consumption in a system. We show that executing at the maximum or minimum processor speed need not be the optimal operating point. While operating at the minimum speed increases the leakage energy contribution, we show that executing at the critical speed and shutting down the processor is more energy efficient. This results in up to 20% energy gains. Furthermore, extending the sleep intervals by our procrastination scheme increases the gains to 35% energy gains. Compared to a naive wakeup scheme, procrastination reduces the number of wakeups to one fourth while stretching the sleep intervals to up to 5 times. The extended idle periods results in an energy efficient operation of the system while meeting all timing requirements. The techniques are simple, energy efficient and can be easily implemented. We plan to extend these techniques to scheduling system wide resources for minimizing the total energy consumption.

References

[1] International Technology Roadmap for Semiconductors 2002 http://public.itrs.net.

[2] Berkeley Predictive Technology Models and BSIM4 http://www-device.eeecs.berkeley.edu/research.html.

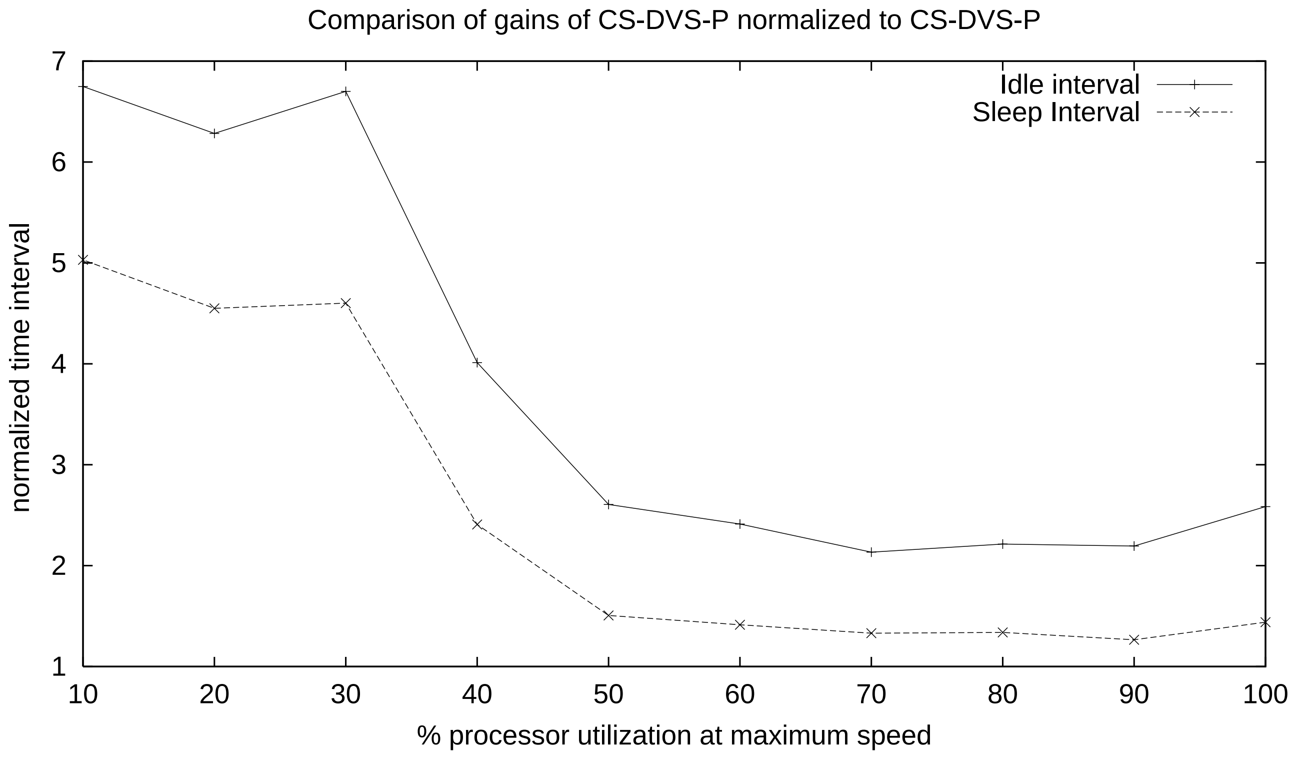

Figure 6. Comparison of average sleep and idle interval of CS-DVS-P normalized to CS-DVS

[3] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *EuroMicro Conference on Real-Time Systems*, 2001.

[4] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Real-Time Systems Symposium*, December 2001.

[5] S. Borkar. Design challenges of technology scaling. In *IEEE Micro*, pages 23–29, Aug 1999.

[6] J. A. Butts and G. S. Sohi. A static power model for architects. In *Intl. Symposium on Microarchitecture*, pages 191–201, 2000.

[7] B. H. Calhoun, F. A. Honore, and A. Chandrakasan. Design methodology for fine-grained leakage control in mtcmos. In *International Symposium on Low Power Electronics and Design*, pages 104–109, 2003.

[8] G. Carpenter. Low power soc for ibm's powerpc information appliance platform. In *http://www.research.ibm.com/arl*.

[9] D. Duarte, N. Vijaykrishnan, M. J. Irwin, and Y.-F. Tsai. Impact of technology scaling and packaging on dynamic voltage scaling techniques. In *15th Annual IEEE International ASIC/SOC Conference*, September 2002.

[10] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *ISCA*,, June 2002.

[11] Z. Hu, S. Kaxiras, and M. Martonosi. Let caches decay: Reducing leakage energy via exploitation of cache generational behavior. In *ACM Transactions on Computer Systems*, May 2002.

[12] Intel PXA250/PXA210 Processor. Intel inc. *(www.intel.com)*.

[13] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proceedings of the 14th Symposium on Discrete Algorithms*, 2003.

[14] M. Johnson, D. Somasekhar, and K. Roy. Models and algorithms for bounds on leakage in cmos circuits. In *IEEE Transactions on CAD*, pages 714–725, 1999.

[15] C. M. Krishna and Y. H. Lee. Voltage clock scaling adaptive scheduling techniques for low power in hard real-time systems. In *Proceedings of the 6th IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, Washington D.C, May 2000.

[16] N. K. J. L. Yan, J. Luo. Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems. In *International Conference on Computer Aided Design*, Nov. 2003.

[17] H. G. Lee and N. Chang. Energy-aware memory allocation in heterogeneous non-volatile memory systems. In *ISLPED*, pages 420–423, 2003.

[18] Y. Lee, K. P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *EcuroMicro Conf. on Real Time Systems*, 2003.

[19] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[20] C. Locke, D. Vogel, and T. Mesler. Building a predictable avionics platform in ada: a case study. In *Proceedings IEEE Real-Time Systems Symposium*, 1991.

[21] S. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *International Conference on Computer Aided Design*, 2002.

[22] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada. 1-v power supply highspeed digital circuit technology with multithreshold- voltage cmos. In *IEEE Journal of Solid-State Circuits*, pages 847– 854, 1995.

[23] C. Neau and K. Roy. Optimal body bias selection for leakage improvement and process compensation over different technology generations. In *International Symposium on Low Power Electronics and Design*, pages 116–121, 2003.

[24] J. Pouwelse, K. Langendoen, and H. Sips. Energy priority scheduling for variable voltage processors. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pages 28–33, 2001.

[25] G. Quan and X. Hu. Minimum energy fixed-priority scheduling for variable voltage processors. In *Design Automation and Test in Europe*, pages 782–787, March 2002.

[26] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *International Conference on Computer Aided Design*, pages 365–368, 2000.

[27] Skipped for Blind Review. Nov. 2003.

[28] Transmeta Crusoe Processor. Transmeta inc. *(http://www.transmeta.com/technology)*.

[29] Y. Zhang, D. Parikh, M. Stan, K. Sankaranarayanan, and K. Skadron. Hotleakage: A temperature-aware, model of subthreshold and gate leakage for architects. In *Tech Report CS-2003-05, Univ. of Virginia Dept. of Computer Science*, Mar 2003.

A Proofs

We say a task τ_i is procrastinated if the system remains idle despite the presence of a task τ_i in the ready queue. The total time units by which an task instance is procrastinated is referred as its *procrastination* interval.

Theorem 1: Given tasks are ordered in non-decreasing order of their period, the procrastination algorithm guarantees all task deadlines if the procrastination interval Z_i of each task τ_i satisfies:

$$\overset{\forall i}{i = 1, \dots, n} \quad \frac{Z_i}{T_i} + \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{T_k} \leq 1 \tag{17}$$

$$\forall_{k<i} \; Z_k \leq Z_i \tag{18}$$

Proof: Suppose the claim is false and a task instance misses its deadline. Let t be the first time that a job misses its deadline. Let t' be the the latest time before t such that there are no pending jobs with arrival times before t' and deadlines less than or equal to t. Since no requests can arrive before system start time $(time = 0)$, t' is well defined. Let $A \subseteq \{\tau_1, \dots \tau_i\}$ be the set of jobs that arrive no earlier than t' and have deadlines at or before t. Since there are pending requests of jobs in A at all times during the interval $[t', t]$, only tasks in A execute during the interval $[t', t]$. Let $X = t - t'$, then the number of instances of tasks τ_k in the interval X is bounded by $\lfloor \frac{X}{T_k} \rfloor$. Let τ_l be the job that arrives at time t'. If the system is executing a lower priority job at time t', preemption occurs and jobs in A are executed. The job executions are procrastinated only if the processor is in the sleep state. By the procrastination algorithm, if at time t' the processor is in the sleep state with the timer not active or the timer value greater than Z_l, then the timer value is set to Z_l. Thus the wakeup timer value after the arrival of τ_l is at most Z_l. Since the timer value can only be decreased by the arrival of another task, the procrastination interval of every task in A is bounded by Z_l. From Equation 18, it follows that $\forall_{k \leq i} Z_k \leq Z_i$ and the maximum procrastination interval for any job in A is bounded by Z_i time units. Since a task misses its deadline at time t, the sum of the total execution time of jobs in A and the procrastination interval Z_i exceeds the length of the interval. Therefore,

$$Z_i + \sum_{k=1}^{i} \lfloor \frac{X}{T_k} \rfloor \frac{C_k}{\eta_k} > X$$

Since $\frac{X}{T_i} \geq \lfloor \frac{X}{T_i} \rfloor$, we have

$$\frac{Z_i}{X} + \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{T_k} > 1$$

Since all jobs in A have their arrival time and deadline in the interval $[t', t]$, we have $T_i \leq X$, and

$$\frac{Z_i}{T_i} + \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{T_k} > 1$$

which contradicts with Equation 17. Thus all tasks meet the deadline by the procrastination algorithm. ■

Lemma 3: The minimum idle period guaranteed by the procrastination algorithm is given as,

$$Z_{min} = min_{1 \leq i \leq n} \quad \left\{ Z_i = (1 - \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{T_k}) T_i \right\} \tag{19}$$

Proof: Note that the procrastination interval of Z_{min} for each task satisfies the constraints given in Lemma 1 and guarantees meeting all task deadlines. Thus for each task τ_i, there exists procrastination intervals $Z_i \geq Z_{min}$ that satisfy Lemma 1. According to the procrastination algorithm, if the processor is in the sleep state with the timer inactive, the arrival of a task sets the timer value to the procrastination interval of the arrived task. The timer counts down every clock cycle and the processor is woken up when the timer expires. If a task arrives with a smaller procrastination interval Z_l than the current timer value, then the timer is set to Z_l. Consider the last time that the timer was set before the expiration of the timer and let task τ_i be the task whose arrival set the timer. Thus the timer expires after a procrastination interval of at least Z_i. Since all tasks have a procrastination interval greater than or equal to Z_{min}, we have $Z_i \geq Z_{min}$. Thus the minimum procrastination interval is Z_{min}. ■

Lemma 4: Given, tasks are executed at maximum speed, the minimum delay interval guaranteed by the procrastination algorithm is greater than or equal to that guaranteed by LC-EDF.

Proof: The minimum idle period given by the LC-EDF algorithm is given as,

$$l_{min} = min_{1 \leq i \leq n} \quad \left\{ l_i = (1 - \sum_{k=1}^{n} \frac{C_k}{T_k}) T_i \right\}$$

and the minimum idle period given by the procrastination algorithm at maximum speed is given by

$$Z_{min} = min_{1 \leq i \leq n} \quad \left\{ Z_i = (1 - \sum_{k=1}^{i} \frac{C_k}{T_k}) T_i \right\}$$

We show that $Z_i \geq l_i$ and hence it follows that $Z_{min} \geq l_{min}$.

$$Z_i = (1 - \sum_{k=1}^{i} \frac{C_k}{T_k}) T_i$$

$$= (1 - \sum_{k=1}^{n} \frac{C_k}{T_k} + \sum_{l=i+1}^{n} \frac{C_l}{T_l}) T_i$$

$$= l_i + (\sum_{j=i+1}^{n} \frac{C_j}{T_j}) T_i$$

$$\geq l_i$$

Since $Z_i \geq l_i$, it follows that the minimum over all Z_i is greater than or equal to the minimum over all l_i. Thus $Z_{min} \geq l_{min}$. ■