行政院國家科學委員會專題研究計畫 期中進度報告

數位家庭:網路、平台與應用--子計畫一:家庭網路與平台之省電設計(2/3)
期中進度報告(完整版)

計 畫 類 別 ： 整合型
計 畫 編 號 ： NSC 95-2219-E-002-014-
執 行 期 間 ： 95 年 08 月 01 日至 96 年 07 月 31 日
執 行 單 位 ： 國立臺灣大學資訊工程學系暨研究所

計 畫 主 持 人 ： 郭大維

報 告 附 件 ： 出席國際會議研究心得報告及發表論文

處 理 方 式 ： 期中報告不提供公開查詢

中 華 民 國 96 年 05 月 28 日

行政院國家科學委員會補助專題研究計畫 □ 成果報告 ■期中進度報告

數位家庭：網路、平台與應用-
子計畫一：家庭網路與平台之省電設計(II/3)

計畫類別：□ 個別型計畫　　■ 整合型計畫
計畫編號：NSC 95－2219－E－002－014－
執行期間：　95 年 08 月 01 日至 96 年 07 月 31 日

計畫主持人：郭大維教授
共同主持人：
計畫參與人員：陳雅淑、修丕承、朱原陞、林建宏、陳健偉、張素熒

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
■出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，■一年□二年後可公開查詢

執行單位：國立台灣大學資訊工程研究所

中　華　民　國　　96　年　　05　月　25　日

行政院國家科學委員會補助專題研究計畫期中報告

計畫名稱：數位家庭：網路、平台與應用-
子計畫一：家庭網路與平台之省電設計(II/3)

計畫編號：NSC95－2219－E－002－014－
執行期限：計畫自民國 95 年 08 月 01 日至民國 96 年 07 月 31 日止

主持人：郭大維 國立台灣大學資訊工程研究所、網路與多媒體研究所
計畫參與人員：陳雅淑、修丕承、朱原陞、林建宏、陳健偉、張素熒
國立台灣大學資訊工程研究所

中文摘要

過去十年來，消費者於各式個人電腦、行動裝置、與消費性電子產品上觀看、取得、與管理數位內容的需求不斷增加，雖然消費者可能對於整合各項產品的功能興趣不大，但如何讓這些產品更緊密的在一起提供服務卻是極為需要，隨著數位內容的提供與裝置連網的大幅成長，這樣的需要與日俱增。以上的觀察提供本計畫強烈的動機來探討與定位相關關鍵技術，以整合連網數位家庭中之個人電腦、行動裝置、與消費性電子產品，藉以提供較佳之數位內容與數位控制傳輸品質與服務。此計畫為三年計畫。第二年我們著重於討論如何設計省電即時核心以提供具有省電功能的工作同步協定，我們提出頻率鎖定(frequency locking)的觀念，並將優先級限高協議(Priority Ceiling Protocol)加入頻率鎖定概念使得頻率轉換的代價得以控制。我們並提出如何計算工作基本頻率使得所有工作都能符合其時間要求，且最優化系統能源消耗。我們也一併提出動態優先權的省電工作協定。經過一連串的實驗，證實我們的方法確實可以大大減低系統中存在工作協定之能源消耗。

Abstract

There has been a growing demand from consumers to acquire, view, and manage digital media on various mobile, PC, and consumer-electronics devices in the past decade. Consumers, in general, might not be interested in merging those devices together in functionality. Instead, consumers do want to have those devices to work much better together. The proliferation of digital media and networking further fuels up the consumer demand. Such an observation motivates the study and prototype implementation of this project. That is to identify and develop critical technology to seamlessly merge isolated islands in the PC Internet world, mobile device world, and consumer electronics world for better delivery of digital contents and control messages in the digital home. A tiny energy-efficient real-time kernel and some development environments would be built as a proof the concept. In the second year of this three-year project, we consider real-time task synchronization protocols with the minimization of energy consumption. We propose the concept of frequency locking and extend the Priority Ceiling Protocol by locking the processor frequency in a restricted way so that the cost in frequency switching is better managed. Algorithms are proposed to assign tasks base frequencies in the minimization of the energy consumption and with the consideration of schedulability tests. Dynamic priority assignment is also addressed. The capability of the proposed methodology is evaluated by a series of experiments, for which encouraging results were presented.

關鍵詞: DVFS, real-time synchronization protocol, energy-efficiency, real-time scheduling

1 Introduction

This subproject is a three-year project. In the second year, we focus on the study and designs of energy-efficient real-time task synchronization protocols for the real time kernel. This report summarizes the major results we have achieved.

With the advance of VLSI circuit designs, many modern processors, such as the Intel StrongARM SA1100 processor [8] and the Intel XScale [9], can now operate at different processor speeds. It provides an excellent means in energy-efficient designs, and a strong driving force in energy-efficient real-time task scheduling is thus generated by that strong demand. While energy-efficient real-time task scheduling is often modeled and resolved in terms of optimization problems, various excellent scheduling algorithms and approximate solutions were proposed in the past decade, e.g. [2, 4, 11, 7].

Beside excellent results reported in the energy-efficient optimization problems, researchers also explore overhead in speed switching, e.g., [14, 5]. It was shown in [8, 9, 22] that the worst-case overhead could be up to hundreds of microseconds in delay because of the changing of the supply voltage of the DC-DC converter or that of the system clock by the phase-locked loop. Sometime the synchronization of the processor and off-chip components, e.g., memory, could take up to one half of a millisecond. For the simplicity of the presentation, voltage or speed switching is also referred to as frequency switching of the processor in this report when there is no ambiguity (Please see Section 2 for the definition of the power consumption function).

While various optimization problems are explored in energy-efficient real-time task scheduling, there are relatively few results in providing real-time task synchronization protocols with energy efficiency considerations (close to the excellent work by Sha, et al [3, 18] in priority ceiling and inheritance). There are two essential technical issues here: (1) What frequency should be assigned to each task to satisfy its timing constraint? (2) When and how should the processor frequency be adjusted when blocking or resumption of a task occurs? The closest related work is an extension of the Priority Ceiling Protocol [18] in frequency inheritance, where a blocking task could inherit the processor frequency of the blocked task to speed up the execution of the blocking task [10]. Other related work is the concept of two alternative frequencies for task executions [21]: One is for the execution of any task when it is not blocked, and the other is adopted to execute the critical section of any task when the task is blocked.

Different from the past work, this research work is motivated by the significant overhead in frequency switching, as reported in [8, 9, 22], and the needs in task synchronization protocols. We propose the concept of frequency locking in task executions and extend the Priority Ceiling Protocol by locking the processor frequency in a restricted way so that the cost in frequency switching is better managed. The objective is to minimize the energy consumption of a given task set, provided that the schedulability of tasks is guaranteed. One particular characteristic of the proposed concept is that there is no frequency switching when a lock request to a semaphore is blocked and when a task resumes its execution from a blocked request. We shall show later that there is also no frequency switching when task preemption occurs. Algorithms are proposed in this report to assign tasks frequencies in the minimization of the energy consumption and with the consideration of schedulability tests. Dynamic priority assignment is then addressed in the report. The capability of the proposed methodology is evaluated by a series of experiments, for which encouraging results were presented.

The rest of this report is organized as follows: Section 2 presents the system model and the problem definition. Section 3 proposes the concept of frequency locking and extend the Priority Ceiling Protocol and the Priority Inheritance Protocol. Algorithms in frequency assignment and schedulability tests are presented. Section 4 reports the performance evaluation of the proposed approach. Section 5 is the conclusion.

2 System Model and Problem Definition

The dynamic power consumption $P(f)$, due to gate switching at processor frequency f, could be modelled as a convex function of the processor speed [15]: $P(f) = C_{ef}V_{dd}^2 f$, where $f = \kappa\frac{(V_{dd}-V_t)^2}{V_{dd}}$, and C_{ef}, V_t, V_{dd}, and κ denote the effective switch capacitance, the threshold voltage, the supply voltage, and a hardware-design-specific constant, respectively. In this report, we will focus our discussions on the cases in which the processor can operate at any frequency f in a given range $[f_{min}, f_{max}]$. Further extensions of this work for processors with discrete available speeds can be done by approaches similar to those reported in [11, 7].

Each periodic task τ_i in the system is associated with an initial arrival time a_i, a worst-case execution CPU cycles c_i, and a period p_i. The relative deadline d_i of τ_i is equal to its period p_i. A task is a template of its jobs, where a job is instantiated for each request of the task. When τ_i is an aperiodic task, p_i denotes its minimum separation time of any two consecutive requests, and d_i is defined explicitly by the system, e.g., p_i. Let the number of CPU cycles executed in a time interval be proportional of the processor frequency. In other words, the number of CPU cycles completed for a task running at a processor frequency f for t time units is $f * t$. The energy consumed for a processor in the execution of a task at the processor frequency f for t time units is $t * P(f)$. When there is no ambiguity in the report presentation, terminologies "task" and "job" are used interchangeably.

Suppose that each resource in the system is guarded by a unique semaphore. Before any task accesses any resource, the corresponding semaphore must be locked. In this report, we consider exclusive locks with an objective in the focusing of the concept for energy-efficient real-time task synchronization (such as that by the Priority Ceiling Protocol (PCP) [18]). Only binary semaphores are adopted for the simplicity of discussion (instead of counting semaphores in the Stack Resource Policy (SRP) [3]). When the lock request is blocked, the job is said *being blocked*. Depending on the adopted resource synchronization protocol and the run-time situation, each job might suffer from a different amount of *blocking time* from some lower-priority tasks, due to access conflict.

In this report, we shall propose the concept of frequency locking (FL) in the frequency adjustments of task executions so as to manage frequency switching overhead and provide real-time task synchronization with energy consumption considerations. We propose to take the priority inversion time of tasks into considerations and have frequency switching only when a task is dispatched and scheduled for the first time, or all of the executing tasks just finish their executions at this time point. The goal is to adjust the processor frequency in a proper pattern and limit the occasions for frequency switching (Please see the detailed policy in later sections). Note that, with schedulability analysis done before the run time, no frequency switching is considered in any blocking or resumption of tasks, due to semaphore locking or unlocking. The rationale behind the concept is on the overhead minimization in frequency switching, which is significant compared to task execution time [22], and the proper extensions of existing task synchronization protocols. In later sections, we shall show how to apply the FL concept to the well-known PCP and the Priority Inheritance Protocol (PIP) [18]. We shall show that their schedulability tests and properties could be extended in a straightforward way, and the worst-case energy consumption of a task or its task set could be derived. In this report, the energy consumption of a task set $\mathbf{T}$ is defined as the energy consumption of tasks in the *hyper-period* of $\mathbf{T}$, denoted by L, where L is the LCM of the task period in $\mathbf{T}$.

3 Frequency Locking for Energy-Efficient Real-Time Task Synchronization

3.1 Overview

The objective of this section is to propose the concept of frequency locking in energy-efficient real-time task scheduling. The idea is to do pre-analysis of the priority inversion time of tasks so that the system can be locked at a proper processor frequency when a new task is dispatched from the ready queue, or all of the executing tasks just finish their executions at this time point. The rationale behind the concept is to avoid any frequent adjustment of the processor frequency because of significant overhead and delay in the adjustment [8, 9, 22].

In this section, the well-known PCP and PIP [18] are extended with the frequency locking concept, referred to as FL-PCP and FL-PIP (Please see Section 3.2). With such an extension, no frequency adjustment is considered when a task is blocked or resumed from blocking to avoid significant overhead and delay, due to frequency adjustment. We shall show how to assign each task a proper processor frequency in the minimizing of the energy consumption without any violation of the timing constraints and with the consideration of the priority inversion time of tasks. The frequency assignment algorithm and properties

of FL-PCP and FL-PIP are then presented (Please see Section 3.3). Remarks on the frequency locking concept over dynamic priority assignment policies are then addressed (Please see Section 3.4). In Section 3.3, we extend the concept of frequency locking based on an observation that a lower priority task often need a higher frequency in execution because of more task preemption cost from higher priority tasks. It also aims at better estimation on the number of frequency switchings in task executions. In such a case, a higher priority task will run at a lower frequency to save energy if it is not blocked by any lower priority task. Moreover, a higher priority task would suffer from less blocking time from lower priority tasks because they run at higher frequencies. This approach is very different from previous results, e.g., [10, 16].

3.2 Frequency Locking: FL-PCP and FL-PIP

3.2.1 Protocol Definitions

In this section, PCP is extended with the concept of frequency locking, referred to as FL-PCP. The task model under discussions follows directly from that of PCP, where a set of fixed-priority tasks is considered in a uniprocessor environment, and binary semaphores are adopted for task synchronization. FL-PIP is then presented with a minor revision over FL-PCP.

As shown in Algorithm 1, the rules of task scheduling, resource allocation, and priority inheritance are exactly as the same as those of PCP, except the rule is on frequency switching. Different from the past work [10, 16], the time moment for frequency switching is determined when a task is dispatched and scheduled for the first time, or all of the executing tasks just finish their executions at this time point. The idea is to lock up the processor frequency for a longer time at a proper level. The first condition for frequency switching is when no task is dispatched and scheduled for the first time, and all of the executing tasks just finish their executions at this time point. The frequency is set as the one for the idle mode because there is no task in the ready queue[1]. The second condition is to run a task at its base frequency to save the energy (to be shown in the later section in the determination of the base frequency) because all of the executing tasks have finished their executions. The third condition requires the processor frequency to set in a non-decreasing order so as to satisfy the schedulability of executing tasks (Please see related theorems in the following section). The fourth condition is satisfied when no new task is dispatched, and some of the executing tasks have not finished their executions at this time point. In such a case, the processor frequency remains to avoid a potentially large number of frequency switchings and their overhead.

[1]This setting is, in fact, optional, and the setting depends on the overhead in switching back and forth between the idle mode and an operating mode of a target processor. For example, the switching overhead of StrongARM SA-1100 to the lower-power sleep mode is 160 ms [8]. Any idle setting action should be careful.

Algorithm 1 FL-PCP

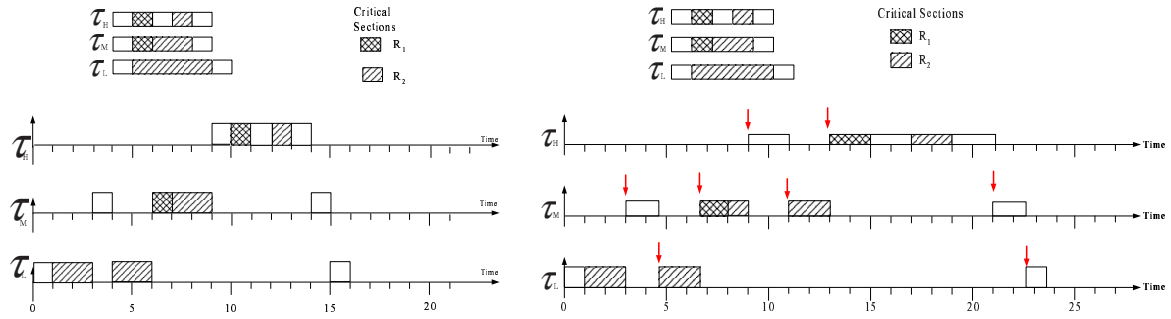
I Task Scheduling, Resource Allocation and Priority Inheritance: Rules are as the same as those of PCP.

II Frequency Switching:

- The processor frequency is switched to a proper one when a task is dispatched and scheduled for the first time, or all of the executing tasks just finish their executions at this time point.
- The new processor frequency is set based on the following four conditions:
  - If no task is dispatched and scheduled for the first time, and all of the executing tasks just finish their executions at this time point, then the frequency is set as the one for the idle mode.
  - If a task is dispatched and scheduled for the first time, and all of the executing tasks just finish their executions at this time point, then the processor frequency is set as the base frequency of the task.
  - If a task is dispatched and scheduled for the first time, and some of the executing tasks have not finished their executions at this time point, then the processor frequency is set as the maximum of its base frequency and the current processor frequency.
  - Otherwise, the processor frequency remains.

FL-PCP could be better illustrated by the following example: Consider three tasks τ_L, τ_M and τ_H, that are ready to execute at time 0, 3 and 9, respectively. Let τ_H and τ_L have the highest priority and the lowest priority in the task set, respectively. The priority ceilings of $R1$ and $R2$ are both the priority of τ_H, according to the definitions of PCP. The execution times of τ_H, τ_M and τ_L are 5, 5 and 6 time units, respectively, when the processor frequency is the maximum available one f_{max}. Suppose that the base frequencies of τ_H, τ_M and τ_L are set as $1/2 * f_{max}$, $2/3 * f_{max}$ and f_{max}, respectively.

As shown in Figure 1.(a), when τ_L arrives at time 0, the processor frequency is set as f_{max}. τ_L later successfully locks $R2$, and the processor frequency remains. At time 3, τ_M preempts τ_L, and the processor frequency is locked at f_{max} (Condition 3). The lock request of τ_M to $R1$ is blocked at time 4 because the



(a) FL-PCP (b) PCP without frequency locking

Figure 1. FL-PCP v.s PCP with DVS

priority of τ_M is no larger than the priority ceiling of $R2$. τ_L resumes its execution and inherits the priority of τ_M, and the processor frequency is locked at f_{max}. At time 6, τ_L unlocks $R2$, and τ_M resumes its execution. τ_M later locks $R1$ and $R2$ successfully. The processor frequency stays at f_{max}. At time 9, τ_H preempts τ_M, and the processor frequency is also locked at f_{max} (Condition 3). Note that the processor frequency stays at f_{max} until τ_H, τ_M and τ_L all finish their executions at time 16 (Condition 1). There are only two frequency switchings in this case (when τ_L starts and finishes its execution, respectively), and one priority inversion occurs.

Suppose that we do not do frequency locking for the same set of tasks. As shown in Figure 1.(b), when τ_M preempts τ_L at time 3, the processor frequency is set as $2/3 * f_{max}$. The lock request of τ_M to $R1$ is later blocked at time 4.5 because the priority of τ_M is no larger than the priority ceiling of $R2$. The processor frequency is set back to f_{max} to execute τ_L, and τ_L inherits the priority of τ_M. As soon as τ_L unlocks $R1$ at time 6.5, τ_M resumes its execution, and the processor frequency is set back to $2/3 * f_{max}$. In this way, there will be totally 10 frequency switchings when all of the tasks finish their executions. As astute readers might point out, the overhead in frequency switching might be overwhelming, and there should be a proper way to lock the processor frequency at certain time points. The observation motivates this research.

The definition of FL-PIP is almost as the same as that of FL-PCP, except that the rule in resource allocation is revised as that for PIP. That is, a resource request of a task τ_i is granted if the resource is available; otherwise, τ_i is blocked by the task that locked the resource when the lock is issued. The policy in frequency switching remains. In the following section, we shall show how to determine the base frequencies for tasks under FL-PCP or FL-PIP by considering task synchronization. We shall later show important properties of the frequency locking concept.

3.2.2 Frequency Assignment

In this section, we shall first present algorithms to assign tasks base frequencies based on the concept of frequency locking to minimize the energy consumption without violating their deadline constraints. We will then show how to resolve the frequency assignment in terms of non-linear programming.

Suppose that tasks of each periodic task set under considerations are sorted and indexed in a non-decreasing order of their periods. Before the base frequencies of tasks under FL-PCP or FL-PIP are determined, we shall first derive the base frequencies of independent periodic tasks under the rate monotonic scheduling (RMS) algorithm, where the priorities of tasks are inversely proportional to their periods. Note that FL-PCP becomes the RMS algorithm with frequency locking when tasks become independent, and the priorities of tasks are assigned based on the RMS algorithm. In this section, tasks with smaller indices are assigned higher priorities based on the RMS algorithm. Let $U(i) = i * (2^{\frac{1}{i}} - 1)$ denote the least upper bound of utilization factors of i periodic independent tasks [13].

Algorithm 2 assigns independent periodic tasks base frequencies based on the concept of frequency locking and an observation that a lower priority task usually need a higher frequency in execution because of more task preemption cost from higher priority tasks. Let α denote the worst-case CPU cycles for each frequency switching [19]. As shown in Algorithm 2, tasks are first assigned base frequencies as the maximum available frequency of the processor for schedulability test (Step 1-5). If the task set is schedulable in such a case, then the minimal frequency f_u is derived to meet all of the timing constraints of tasks (i.e., the utilization factor checkup $U(N)$) without the considerations of frequency-switching cost (Step 6). The base frequencies of tasks are then adjusted again to reduce the energy consumption with frequency-

Algorithm 2 FA-ITS

Input: Task set $\mathbf{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$
Output: Frequency Assignment of An Independent Fixed-Priority Task Set $\mathbf{T}$

1: **for** each $i \leftarrow 1$ to N **do**
2: **if** $U(i) \geq \sum_{1 \leq j < i} (\frac{c_j}{f_{max}} * \frac{1}{p_j}) + (\frac{1}{f_{max}} * \frac{c_i}{p_i})$ **then**
3: $f_i = f_{max}$
4: **else**
5: return with the base frequency assignments of schedulable tasks
6: Find the minimal frequency f_u such that

$$U(N) \geq \sum_{1 \leq j \leq N} (\frac{c_j}{f_u} * \frac{1}{p_j})$$

7: **for** each $i \leftarrow 1$ to N **do**
8: Find the minimal frequency f_i such that

$$U(i) \geq \sum_{1 \leq j < i} (\frac{c_j}{f_j} * \frac{1}{p_j}) + \sum_{1 \leq j \leq i, f_j \neq f_k, \exists k, 1 \leq k < j} (\frac{\alpha}{f_j} * \frac{1}{p_i}) + (\frac{1}{f_i} * \frac{c_i}{p_i} + \frac{2\beta}{p_i})$$

9: **if** $f_i > f_{max}$ **then**
10: **for** each $j \leftarrow 1$ to N **do**
11: resetting $f_j = f_{max}$
12: return with the base frequency assignments of all tasks
13: **if** $f_i < f_u$ **then**
14: $f_i = f_u$
15: **for** each $i \leftarrow 1$ to $(N-1)$ **do**
16: **if** $f_i > f_{i+1}$ **then**
17: $f_{i+1} = f_i$
18: **for** each $i \leftarrow 1$ to $(N-1)$ **do**
19: **if** $\frac{P(f_N)c_i}{f_N} < \frac{P(f_i)(c_i+\alpha)}{f_i}$ **then**
20: $f_{i+1} = f_i = f_N$
21: **else**
22: **if** $f_i > f_{i+1}$ **then**
23: $f_{i+1} = f_i$
24: return with the base frequency assignments of all tasks

switching cost considerations: Tasks are reassigned base frequencies in a decreasing order of their priorities (Steps 7-14). The i-th task τ_i is assigned a base frequency f_i such that the utilization factor of the first i tasks is not more than $U(i)$ (Step 8), where the first, second, and third items in the right hand side of the inequality denote the preemption cost of higher priority tasks, the cost in frequency switching of higher priority tasks, and the utilization factor of the i-th task (plus the worst-case cost in switching between operating and idle modes). Let β denote the delay time to switch from the operating mode to the idle mode (and vice versa). In the worst case, there are 2β in the third item: One happens when τ_i arrives at the time moment when the processor is switched from the operating mode to the idle mode. The other happens when the processor is switched from the idle mode back to the operating mode to run a task with a priority no less than that of τ_i.

The second item comes from the worst-case estimated cost α/f_j to switch the processor frequency of a higher priority task to that of another higher priority task (including τ_i) because the former task finishes its execution, and the later then starts its execution at the same time. The second item is the ratio of the cost and the period of τ_i because the cost is charged to τ_i as a part of its utilization. When the base frequency of a task is larger than the maximum available frequency of the processor, then the algorithm should stop and return by assigning the base frequency of each task as the maximum frequency of the processor (Steps 9-12). If the derived base frequency of a task is lower than the minimal frequency f_u derived in Step 6, then the base frequency of the task is set as f_u (Steps 13-14). After the loop between Steps 7 and 14 terminates, the base frequencies of tasks are adjusted again such that lower priority tasks are assigned higher base frequencies (Steps 15-23). Note that the base frequency of any task is assigned as the maximum base frequency of tasks, i.e., f_N, if the energy consumption to run a task at a lower frequency plus the energy consumption in frequency switching is no less than the energy consumption to run the task at the maximum base frequency of tasks. In Section 3.3, we shall show that there is no frequency switching due to task preemption, because of the above idea and the concept of frequency locking (Please see Lemma 3.1).

Algorithm 3 assigns base frequencies to tasks under FL-PCP. Let B_i denote the worst-case blocking CPU cycles of a task τ_i under PCP (i.e., the corresponding worst-case blocking time of a task under PCP with CPU cycles as units). The derivation of B_i follows directly from the corresponding way in blocking time derivation under PCP [18]. Algorithm 3 is similar to Algorithm FA-ITS, except that the the blocking CPU cycles are considered. Algorithm FA-ITS is first invoked to derive the base frequencies of tasks when no resource sharing is considered (Step 1). In each iteration of the loop between Step 2 and Step 8, the blocking CPU cycles B_i of a task is considered. The lowest frequency is first determined as the worst case in the derivation of blocking time to τ_i (Step 3). The third item in the right hand side of the inequality of Step 8 of Algorithm FA-ITS is revised to check it up whether the resulted utilization factor, due to additional blocking CPU cycles B_i, will violate $U(i)$ (Step 5). If there is a violation, then a higher frequency is needed to satisfy the utilization bound. The task with f_L is increased until there is no such violation. If there is no such f_L,

Algorithm 3 Frequency Assignment of FL-PCP

Input: Task set $\mathbf{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$, Resource usage
Output: Frequency Assignment for Each Task τ_i

1: FA-ITS(T);
2: **for** each $i \leftarrow 1$ to N **do**
3: f_L= the lowest base frequency among tasks which might block τ_i
4: $U = \sum_{1 \le j < i} (\frac{c_j}{f_j} * \frac{1}{p_j}) + \sum_{1 \le j < i, f_j \ne f_k, \exists k, 1 \le k < j} (\frac{\alpha}{f_j} * \frac{1}{p_i}) + (\frac{1}{f_L} * \frac{B_i + c_i}{p_i})$
5: **if** $U > U(i)$ **then**
6: Increase the frequency of the task with f_L such that the value of the above U is no larger than $U(i)$
7: **if** $f_L > f_{max}$ **then**
8: leave the loop
9: **for** each $j \leftarrow 1$ to $(N-1)$ **do**
10: **if** $f_j > f_{j+1}$ **then**
11: $f_{j+1} = f_j$
12: **for** each $j \leftarrow 1$ to $(N-1)$ **do**
13: **if** $\frac{P(f_N)c_j}{f_N} < \frac{P(f_i j)(c_j+\alpha)}{f_j}$ **then**
14: $f_{j+1} = f_j = f_N$
15: **else**
16: **if** $f_j > f_{j+1}$ **then**
17: $f_{j+1} = f_j$
18: return the base frequency assignments of all tasks with schedulability guarantee of tasks $\tau_1, \cdots, \tau_{i-1}$

i.e., $f_L > f_{max}$, then the schedulability of task τ_i can not be guaranteed, and exit the loop (Steps 7 and 8). Note that i would be $(N+1)$ if every task is schedulable by considering the blocking CPU cycles. Furthermore, there is no cost in frequency switching for τ_i because the blocking will result in frequency locking (and no frequency switching occurs on τ_i in this case). (Please see Lemma 3.2 in Section 3.3.) Similar to Algorithm FA-ITS, the base frequencies of tasks are then revised to make sure that lower priority tasks are assigned higher base frequencies (Steps 9-17). The base frequencies of all tasks are then returned, and only tasks that pass the schedulability test (i.e., the formula in Step 4) are guaranteed being schedulable (Step 18). The complexity of Algorithm 3 is $O(n^3)$, when there are n tasks in the task set $\mathbf{T}$.

While Algorithm 3 (and Algorithm 2) provides a greedy and intuitive algorithm to determine the base frequencies of tasks under FL-PCP, the frequency assignment problem can be resolved in a more precise way by a non-linear programming solver [12] based on the Algorithms 2 and 3. The problem can be formulated as an optimization problem: The objective is to assign each task τ_i in a task set $T = \{\tau_1, \cdots, \tau_n\}$ a base frequency f_i so as to minimize the worst-case energy consumption of task executions in the hyperperiod, i.e., $P(f_n) * \sum_{1 \le i \le n}((\frac{c_i}{f_n} + 2\beta) * \frac{L}{p_i}) + P(f_n) * \frac{\alpha}{f_n} * \sum_{1 \le i \le (k-1)} \frac{L}{p_i} + P_{idle} * (L - (\sum_{1 \le i \le n}(\frac{c_i}{f_n} + 2\beta) * \frac{L}{p_i}))$, where P_{idle} is the energy consumption per time unit during the idle time, and k is the smallest task index such that $f_k = f_n$, subject to the following constraints (Please see Theorem 3.8 for the worst-case energy consumption estimation):

$$U(i) \ge \sum_{j=1}^{i-1} (\frac{c_j}{f_j} * \frac{1}{p_j}) + \sum_{j=1}^{i} (\frac{\alpha}{f_j} * \frac{1}{p_i}) + (\frac{1}{f_i} * \frac{c_i}{p_i} + \frac{2\beta}{p_i}) \quad i = 1, 2 \ldots, n$$

$$U(i) \ge \sum_{j=1}^{i-1} (\frac{c_j}{f_j} * \frac{1}{p_j}) + \sum_{j=1}^{i-1} (\frac{\alpha}{f_j} * \frac{1}{p_i}) + (\frac{1}{f_k} * \frac{B_i + c_i}{p_i}) \quad i = 1, 2 \ldots, n, \; k = i+1, i+2 \ldots, n$$

$$\frac{P(f_n)c_i}{f_n} \ge \frac{P(f_i)(c_i + \alpha)}{f_i} \quad i = 1, 2 \ldots, n$$

$$f_{min} \le f_i \le f_{max}, \quad i = 1, 2 \ldots, n$$

$$f_{i-1} \le f_i \quad i = 2 \ldots, n$$

Equations 1 and 2, that come from the Step 8 of Algorithm 2 and Step 4 of Algorithm 3, respectively, make sure that switching cost and worst-case blocking cost are both satisfied in schedulability tests. Note that the frequency switching cost in Equation 1 or 2 is easy to understand but more conservative, compared to the counterpart in Algorithms 2 or 3, and the cost can also be reformulated accordingly. Equation 3 is to make sure that the energy consumption to run a task at its base frequency plus the energy consumption in frequency switching is no larger than the energy consumption to run the task at the maximum base frequency of tasks. Equation 4 lets the base frequencies of tasks stay in a legal range $[f_{min}, f_{max}]$, and Equation 5 requires that lower priority tasks are assigned higher base frequencies[2]. *Note that Algorithm 3 and the above non-linear programming formulation could be used for FL-PIP directly with the worst-case blocking CPU cycles B_i of each task being derived based on PIP [18].*

3.3 Properties

The purpose of this section is to present the properties of FL-PCP and FL-PIP. For the rest of this report, we shall assume that the base frequencies of tasks under considerations are set by Algorithms 2 and 3. Tasks are also sorted and indexed in a non-decreasing order of their periods.

Lemma 3.1 *There is no frequency switching under FL-PCP when task preemption occurs.*

Proof. The correctness of this lemma follows directly from the third condition of FL-PCP in frequency switching and the fact that higher priority tasks are associated with lower base frequencies. □

[2]Note that the constraints based on Equations 1 and 2 could be further refined, but we choose to keep them in the current form for better understanding.

Lemma 3.2 *There is no frequency switching under FL-PCP when a task is blocked because of its lock request. Furthermore, any blocked task always executes at a frequency no less than the base frequency of the blocking task under FL-PCP until the task completes its execution.*

Proof. Suppose that a lower priority task τ_L blocks some higher priority task τ_H under FL-PCP. Based on the definition of FL-PCP, when the blocking of τ_H occurs, there is no frequency switching. Furthermore, the only possible situation for such a blocking to happen is when τ_L starts earlier than τ_H does, and τ_L locks a semaphore that later blocks τ_H. Let τ_H preempt some lower priority task (that might or might not be τ_L). Because of Lemma 3.1, there is no frequency switching when the task preemption occurs. Note that τ_L has started its execution when the task preemption occurs. Based on the rule of FL-PCP in frequency switching, τ_H always executes at a frequency no less than τ_L's base frequency (that is also no less than that of τ_H) under FL-PCP until τ_H completes its execution. □

Lemma 3.3 *No task executes at a frequency less than its base frequency under FL-PCP.*

Proof. The correctness of this lemma follows directly from Lemmas 3.1 and 3.2. □

Lemma 3.4 *There is at most one priority inversion for any task under FL-PCP. Furthermore, there is no transitive blocking and any deadlock under FL-PCP.*

Proof. The correctness of this lemma follows directly from the properties of PCP [18]. □

Theorem 3.5 *[18] Task τ_i is schedulable by PCP if $U(i) \geq \sum_{1 \leq j < i}(\frac{C_j}{p_j}) + (\frac{B_i + C_i}{p_i})$, where C_j, p_j, and B_j denote the worst-case CPU time, the period, and the worst-case blocking time of task τ_j, respectively.*

Lemma 3.6 *Let the processor never enter the idle mode under FL-PCP. A periodic task set is schedulable by FL-PCP if and only if it is schedulable by PCP.*

Proof. The correctness of the "only if" part follows directly from the fact that there is no cost in frequency switching under PCP, where all tasks run at the maximum available frequency f_{max}. Since β in the formula (of Step 8) of Algorithm 2 can be considered as zero (because the processor is assumed to stay at the operating mode), the satisfaction of both the frequency assignment formula in Step 8 of Algorithm 2 and the frequency assignment formula in Step 4 of Algorithm 3 would guarantee the schedulability of the task set based on Theorem 3.5.

The "if" part can be proved by considering two cases: (1)The processor frequency always stays at f_{max} under FL-PCP. (2)The processor frequency sometime drops below f_{max} under FL-PCP. The "if" part holds for Case 1 because FL-PCP becomes PCP when the processor frequency always stays at f_{max}. The major differences between Cases 1 and 2 are on the frequency switching cost under FL-PCP and the lower bound on the execution frequency of each task (because of the estimation of the worst-case execution time). Based on Lemma 3.1, there is no frequency switching due to task preemption. As a result, each task has at most one frequency switching before a lower priority task (e.g., τ_i) is scheduled. That happens when the task is scheduled at the time instant when another higher priority task finishes its execution. In other words, the frequency switching cost under FL-PCP is bounded by $\sum_{1 \leq j \leq i, f_j \neq f_k, \exists k, 1 \leq k < j}(\frac{\alpha}{f_j} * \frac{1}{p_i})$ (or $\sum_{1 \leq j < i, f_j \neq f_k, \exists k, 1 \leq k < j}(\frac{\alpha}{f_j} * \frac{1}{p_i})$ if τ_i is blocked by a lower-priority task) when the schedulability of task τ_i is considered. (Note that they are the second part in the frequency assignment formula in Step 8 of Algorithm 2 and that in Step 4 of Algorithm 3, respectively.) Furthermore, Lemma 3.3 shows that no task executes at a frequency less than its base frequency under FL-PCP. In other words, the worst-case CPU time of each task τ_i is bounded by $\frac{c_i}{f_i}$. (Note that it contributes the utilization factor of tasks in the first part in the frequency assignment formula in Step 8 of Algorithm 2 and that in Step 4 of Algorithm 3.) According to Lemma 3.2, any blocked task always executes at a frequency no less than the base frequency of the blocking task under FL-PCP until the task completes its execution, so the worst-case blocking time of τ_i is bounded by $\frac{B_i}{f_L}$. Since the worst-case blocking time of τ_i is taken into consideration by $\frac{B_i}{f_L}$, the schedulability of any task τ_i is guaranteed by Theorem 3.5 and the frequency assignment of tasks based on both the frequency assignment formula of Algorithms 2 and 3. □

When the overhead in frequency switches between the operating mode and the idle mode must be considered in schedulability analysis, the following theorem should be proved. Note that such overhead could be limited, e.g., no more than 1 μSec in many cases [20].

Theorem 3.7 *Suppose that the processor might enter the idle mode under FL-PCP. A periodic task set is schedulable by FL-PCP if and only if it is schedulable by PCP.*

Proof. The correctness of this theorem directly follows from that of Lemma 3.6 by considering the total time in frequency switches between the operating mode and the idle mode under FL-PCP (in the "if" part proof). There are two cases to consider in the schedulability of any task τ_i: (1) If τ_i is blocked, then there is no time spent in frequency switches between the operating mode and the idle mode based on Lemma 3.2 (when the schedulability of τ_i is considered). (2) If τ_i is never blocked, then there is only one chance for frequency switches between the operating mode and the idle mode (when the schedulability of τ_i is considered). In such a case, there would be no possibility for any lower-priority task to block τ_i because it would has no chance to lock anything to block τ_i. As a result, the frequency assignment formula in Step 8 of Algorithm 2 and Theorem 3.5 would guarantee the schedulability of τ_i. □

Theorem 3.8 *Given a set of n periodic schedulable tasks, the total energy consumption under FL-PCP in a hyper-period is no more than $P(f_n) * \sum_{1 \le i \le n}((\frac{c_i}{f_n} + 2\beta) * \frac{L}{p_i}) + P(f_n) * \frac{\alpha}{f_n} * \sum_{1 \le i \le (k-1)} \frac{L}{p_i} + P_{idle} * (L - (\sum_{1 \le i \le n}(\frac{c_i}{f_n} + 2\beta) * \frac{L}{p_i}))$, where P_{idle} is the energy consumption per time unit during the idle time, and k is the smallest task index such that $f_k = f_n$.*

Proof. The correctness of the energy consumption bound follows directly from the following three facts: (1) The first part serves as an upper bound on the energy consumption of task executions and the frequency switching to their corresponding base frequency, i.e., $\frac{\alpha}{f_i}$, in a hyper-period because $P(f_n) * \frac{c_i}{f_n} \ge P(f_i) * \frac{c_i + \alpha}{f_i}$ for each task τ_i (where $f_i \ne f_n$), and the worst-case delay time in switching between operating and idle modes per task execution is at most 2β. (2) The second part is the energy consumption in the processor frequency switching to execute tasks with the base frequency as f_n. Since a frequency switching for the execution of a task with the base frequency as f_n occurs only when the current processor frequency is not f_n, the number of frequency switchings to execute all of tasks with the base frequency as f_n is bounded by $\sum_{1 \le i \le k-1} \frac{L}{p_i}$. Note that when a task with the base frequency as f_n is dispatched, the processor frequency will be locked at f_n until the task completes according to Lemmas 3.1 and 3.2. (3) The third part is an upper bound on the energy consumption of task executions at the idle mode, where $(\sum_{1 \le i \le n}(\frac{c_i}{f_n} + 2\beta) * \frac{L}{p_i})$ is the minimum time in executing tasks in a hyper-period, including the worst-case delay time in switching between operating and idle modes. □

Note that the theorems proved in this section for FL-PCP remain correct for FL-PIP when the worst-case blocking CPU cycles B_i of each task is derived based on PIP [18].

3.4 Dynamic Priority Assignment: EDF and SRP

When EDF is adopted for independent task scheduling, it is shown that the frequencies of all task executions should be the same in the minimization of the energy consumption [2]. As a result, we should find one frequency for all task executions that results in 100% utilization factor, and Algorithm 2 can be revised accordingly.

When SRP [3] adopts the RMS priority assignment, the frequency switching rule of FL-PCP can be applied directly. Algorithms 2 and 3 remain correct in the base frequency assignment, except that the blocking CPU cycles of tasks should be derived based on SRP. The resulted SRP is referred to as FL-SRP(RMS). When SRP adopts the EDF priority assignment, the base frequency assignment could be done as follows: One initial base frequency of all tasks are first determined as discussed in the above paragraph (i.e., one single frequency for all tasks [2]). The basic idea is to repeatedly invoke Algorithms 2 and 3 until the base frequencies of tasks satisfy the Step 8 of Algorithm 2 and the Step 4 of Algorithm 3. In each iteration, we run Algorithm 3 by setting $U(i) = 1$ for all i to adjust the base frequency of each task by considering the blocking CPU cycles of tasks. Algorithm 2 is then invoked to further adjust the base frequency of each task based on the cost in frequency switching, where $U(i) = 1$ for all i. Note that we assume that tasks are sorted in a non-decreasing order of their relative deadlines (and preemption levels). The resulted SRP is referred to as FL-SRP(EDF).

4 Performance Evaluation

4.1 Data Sets and Performance Metrics

The purpose of this section is to have performance evaluation of FL-PCP in terms of the energy consumption in task executions, compared to PCP without voltage scaling (i.e., PCP) [18], PCP with a minimal constant frequency (PCPC), the Uniform Slowdown with Frequency Inheritance algorithm (USFI) [10], and the Dual Speed algorithm (DS) [21]. Under PCPC, all of the tasks executed at the maximum of the minimal required frequency of each task based on the schedulability test of PCP. When DS was adopted, no pre-

emption was allowed for any critical section. The high speed H and the low speed L under DS were derived based on the following inequalities, where the rate monotonic priority assignment (RM) was adopted: $\frac{\sum_{j=1}^{i}\lceil\frac{p_i}{p_j}\rceil*c_j+B_j}{p_i} \leq H$ and $\frac{\sum_{j=1}^{i}\lceil\frac{p_i}{p_j}\rceil*c_j}{p_i} \leq L, \forall 1 \leq i \leq n$. Note that RM assigned a higher priority to a task with a smaller period.

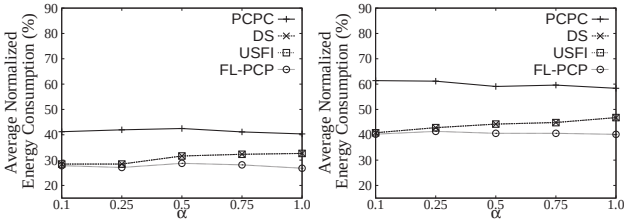
The specifications of Intel XScale was adopted in the experiments, where the power consumption function was approximated by $P(f) = (0.08 + 1.52f^3)$ Watt [17]. The maximum and minimum available frequencies were normalized to 1 and 0.1, respectively. The workloads in the experiments were generated in a randomized way: The number of tasks per task set ranged from 10 to 20. Tasks were all periodic, and their periods were picked up in the range [20ms, 8000ms]. The CPU utilization of each task ranged from 5% to 10%, and the total CPU utilization of each task set ranged from 40% to 70% based on the maximum processor frequency. The initial arrival time of each task was generated by a random distribution function. The number of semaphores shared by tasks ranged from 5 to 10, and which semaphore might be locked by which task was determined by a random distribution function. As a result, the probability of blocking for each task ranged from 25% to 50%. Furthermore, a task with a higher CPU utilization tended to lock more semaphores.

The *blocking factor* of a task was defined as the maximum percentage of time in the execution time of a task in blocking, and it ranged from 10% to 30%. In other words, the maximum blocking duration of a task τ_i was $c_i * blocking\_factor_i$. The frequency switching overhead ranged from $0.1ms$ to $1ms$ at the maximum processor frequency [8, 9, 22][3]. The delay time in switching from the operating mode to the idle mode (and vice versa) was defined as 1 μs [20]. The power consumption in the idle mode was set as 0.01 times of that in the operating mode [6]. In the experiments, 1000 task sets were generated for each system configuration, and their results were averaged for comparison. Each simulation run was conducted for the duration of the hyperperiod of the task set.

The primary metric in performance evaluation was the *normalized energy consumption*. Let X and Y denote the energy consumption amount of the algorithm under performance evaluation and that of PCP, respectively. The normalized energy consumption of the algorithm was defined as X/Y.

[3]When the delay overhead in frequency switching was proportional to the difference between the original frequency and the current frequency, the delay time in each frequency switching of all algorithms was revised from $(\frac{\alpha}{f_i})$ to $\alpha * |f_i - f_{min}|$ [1]

4.2 Experiment Results



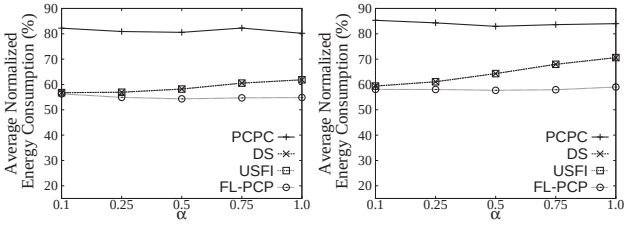
(a) U=0.5 (b) U=0.6

Figure 2. Normalized energy consumption with different values of the frequency switching overhead

Figure 2.(a) and (b) showed the normalized energy consumption of FL-PCP, PCP, PCPC, USFI, and DS, when there were 10 tasks with the total CPU utilization $U = 0.5$ and $U = 0.6$ at the maximum frequency, respectively, and the available frequencies could be any number between 0.1 and 1.0. The horizontal axis is the overhead in frequency switching at the maximum processor frequency, and the vertical axis is the normalized energy consumption. Because the experiments of other workloads had similar results, they were not included in the report. It was shown that FL-PCP always achieved the lowest normalized energy consumption, compared to other algorithms. FL-PCP also tended to do even better than other algorithms when the overhead in frequency switching increased. In other words, FL-PCP was also good in terms of energy efficiency with larger overhead in frequency switching.

The energy consumption under FL-PCP was much less than that of PCP (with only 30%-40% of that of PCP, where that of PCP was 100% in comparison). FL-PCP greatly outperformed PCPC, especially when the total CPU utilization of a task set increased (by up to 20%). This was because the increasing in the task utilization under PCPC might result in a high constant frequency such that the energy consumption increased much more quickly. As astute readers might notice, the energy consumption of task executions under PCPC did not increase when the overhead in frequency switching increased. It was mainly because all tasks executed at the same frequency under PCPC such that no frequency switching occurred during the run time. DS and USFI had similar performance in the experiments, as shown in Figure 2. In the experimental results, it was observed that the frequencies of tasks derived by DS and USFI were the same in most task sets. We also found out that a lower priority task usually needed a higher frequency in executions because of more task preemption cost from higher pri-

ority tasks in the experiments. The high speed H and the low speed L were the same under DS in most task sets. Under USFI, a task τ_i was assigned a frequency no lower than that of a task with a priority lower than that of τ_i. As a result, the frequencies of tasks were identical under USFI in most task sets.



(a) 10 tasks (b) 20 tasks

Figure 3. Normalized energy consumption with different values of the frequency switching overhead

Figure 3.(a) and (b) showed the normalized energy consumption of FL-PCP, PCP, PCPC, USFI, and DS, when there were 10 and 20 tasks, respectively, with the total CPU utilization $U = 0.7$ at the maximum processor frequency, and the available frequencies could be any number between 0.1 and 1.0. Compared with the results shown in Figure 3.(a) and (b), the energy consumption of task executions increased when the number of tasks increased. Furthermore, USFI and DS had similar performance in energy efficiency. The performance gap between FL-PCP and USFI/DS increased when the number of tasks increased. It was because more frequency switchings occurred under USFI/DS when there were more tasks. Similar to the experimental results in the above figure, since all tasks executed at the same frequency under PCPC, the performance gap between FL-PCP and PCPC remained similar when the number of tasks increased.

Experiments were also conducted to evaluate the energy consumption of FL-PCP with different blocking factors and different numbers of processor frequencies. The results show that the energy consumption increased, in general, when the blocking factor increased and the overhead in frequency switching was less than $0.5ms$ at the maximum processor frequency. Moreover, the energy consumption of task executions, when there were 5 available processor frequencies, was 10% more than that when there was an infinite number of processor frequencies. The corresponding figures are not included because of space limitation. Note that when the number of available processor frequencies was finite, the available frequencies were derived by picking up real numbers between 0.1 to 1 with an equal consecutive distance. For example, 0.1, 0.55, and 1 were the available frequencies of a processor with three available frequencies. The needed number of available processor frequencies increased, when the total CPU utilization increased, or when the total number of tasks increased. Furthermore, the results also showed that the overhead in frequency switching had little impacts on the number of needed processor frequencies.

5 Conclusion

This report explores real-time task synchronization with the minimization of energy consumption. We propose the concept of frequency locking in task executions and extend the Priority Ceiling Protocol by locking the processor frequency in a restricted way so that the cost in frequency switching is better managed. The objective is to minimize the energy consumption of a given task set, provided that the schedulability of tasks is guaranteed. One particular characteristic of the proposed concept is that no frequency switching occurs when a lock request to a semaphore is blocked, or when a task resumes its execution from a blocked request. Tasks with lower priorities are assigned higher processor frequencies because they have higher preemption cost and could result in lower blocking time to higher priority tasks. Algorithms which are proposed in this report assign tasks frequencies in the minimization of the energy consumption and with the consideration of schedulability tests. Dynamic priority assignment, e.g., the Earliest Deadline First algorithm [13] with the Stack Resource Policy [3], is then addressed. Performance evaluations show that our proposed algorithms can significantly reduce the energy consumption and outperformed previous results.

For future research, we shall further explore the minimization issues of energy consumption with more complicated system architectures and resources, such as memory, co-processors, and devices. More research and implementation experiences in these directions might be proved very rewarding for many mobile system designs.

References

[1] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi. Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. In *DATE*, 2004.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *ECRTS*, pages 225–232, 2001.

[3] T. P. Baker. A stack-based resource allocation policy for real-time process. In *RTSS*, 1990.
[4] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proceedings of the 2004 Symposium on Foundations of Computer Science*, pages 520–529, 2004.
[5] E. Bini, G. Buttazzo, and G. Lipari. Speed modulation in energy-aware real-time systems. In *ECRTS*, pages 309–318, 2005.
[6] I. T. R. I. S. T. Center. Parallel architecture core digital signal processor core. Technical report, Industrial Technology Research Instittute, `http://int.stc.itri.org.tw/eng/research/multimedia-soc.jsp?tree_idx=0200`, 2006.
[7] J.-J. Chen, T.-W. Kuo, and C.-S. Shih. $1+\epsilon$ approximation clock rate assignment for periodic real-time tasks on a voltage-scaling processor. In *the 2nd ACM Conference on Embedded Software (EMSOFT)*, pages 247–250, 2005.
[8] Intel. Intel StrongARM* SA-1110 microprocessor developer's manual. Technical report, Intel, `http://developer.intel.com/design/strong/manuals/278240.htm`, 2001.
[9] Intel. Intel PXA26x processor family developer's manual. Technical report, Intel, `http://www.intel.com/design/pca/applicationsprocessors/manuals/278638.htm`, 2004.
[10] R. Jejuilar and R. Gutpa. Energy aware task scheudling with task synchoronization for embedded real-time system. *IEEE Transcation on Computer Aided Design of Integrated Circuits and Systems*, 2006.
[11] W.-C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *DAC*, pages 125–130, 2003.
[12] Lindo Systems INC. LINGO 1.0.
[13] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
[14] A. Manzak and C. Chakrabarti. Variable voltage task scheduling algorithms for minimizing energy. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pages 279–282. ACM Press, 2001.
[15] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2nd edition, 2002.
[16] S. Saewong and R. Rajkumar. Practical voltage-scaling for fixed-priority rt-systems. In *RTAS*, pages 106–115, 2003.
[17] INTEL-XSCALE, 2003. http://developer.intel.com/design/xscale/.
[18] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 1990.
[19] D. Shin, J. Kim, and S. Lee. Intra-task voltage scheduling for low-energy hard real-time applications. *IEEE Design and Test of Computers*, 18(2):20–30, 2001.
[20] K. A. Vardhan and Y. N. Srikant. Transition aware scheduling: Increasing continuous idle-periods in resource units. In *Proceedings of ACM Computing Frontiers*, 2005.
[21] F. Zhang and S. T. Chanson. Processor voltage scheduling for real-time tasks with non-preemptible sections. In *RTSS*, 2002.
[22] Y. Zhu and F. Mueller. Feedback edf scheduling exploiting hardware-assisted asynchronous dynamic voltage scaling. *LCTES*, 2005.

可供推廣之研發成果資料表

□ 可申請專利　■可技術移轉　日期：__年__月__日

國科會補助計畫	計畫名稱：數位家庭：網路、平台與應用- 計畫主持人：郭大維 計畫編號：NSC95－2219－E－002－014－ 學門領域：資訊工程
技術/創作名稱	Frequency Locking for Energy-Efficient Real-Time Task Synchronization
發明人/創作人	郭大維 陳雅淑
技術說明	中文： 頻率鎖定(frequency locking)概念是預先分析所有工作的優先權倒置(priority inversion)的情形，使得系統能夠被鎖定在一個適當的頻率狀態，只有當新工作被分配到系統或是所有工作都完成的時候才會作頻率轉換的動作。由於頻率轉換會造成多餘的電量消耗以及時間延遲，所以此觀念藉由避免無謂的頻率轉換，有效的降低系統能源消耗。
	英文： Frequency locking is to pre-analysis of the priority inversion time of tasks so that the system can be locked at a proper processor frequency when a new task is dispatched from the ready queue, or all of the executing tasks just finish their executions at this time point. It can avoid any frequent adjustment of the processor frequency to minimize system energy consumption because of significant overhead and delay in the adjustment.
可利用之產業 及 可開發之產品	此觀念可以實作到目前的各式即時系統核心（或作業系統）中所使用的工作同步協定，將會有效協助開發省電的系統晶片之各式相關產品。
技術特點	考慮到即時系統核心的客置化會造成系統發展的差異，頻率鎖定為一個簡單的觀念可以輕易的應用至各式的工作同步協定，保證系統中的每份工作的最晚反應時間以及系統最大需要消耗的能源。 為應用在實際系統當中，本技術考慮頻率轉換的能量消耗與時間延遲，能夠有效避免系統中頻繁的頻率轉換。此觀念並提供排程與能源分析的公式，能協助系統開發者有效且準確的分析系統可承載的工作量以及能源消耗。
推廣及運用的價值	此觀念可運用至系統晶片中的省電即時核心之設計理念， 並進一步推廣至設計省電系統晶片的系統層級設計工具，藉由省電即時核心能有效且準確的分析系統效能與耗電量，協助使用者設計具有省電功能的單晶片系統。

※ 1.每項研發成果請填寫一式二份，一份隨成果報告送繳本會，一份送　貴單位研發成果推廣單位（如技術移轉中心）。

※ 2.本項研發成果若尚未申請專利，請勿揭露可申請專利之主要內容。

※ 3.本表若不敷使用，請自行影印使用。

『十二屆亞洲與南太平洋設計自動化研討會』
(12th Asia and South Pacific Design Automation Conference (ASP-DAC 2007)) 暨參訪

心得報告

郭大維

(Tei-Wei Kuo)

國立台灣大學 資訊工程學系

1. Summary in Conference Events

12th Asia and South Pacific Design Automation Conference (ASP-DAC) was held between January 23 and 26 in Yokohama, Japan in 2007. ASP-DAC 2007 was hosted by Prof. Hidetoshi Onodera (Kyoto University), as the General Chair, and Prof. Yusuke Matsunaga (Kyushu University), as the Program Chair. ASP-DAC is the best conference in the Asian and Pacific Region for computer-aided design and automation. Every year, there are hundreds of professors, researchers, engineers, and graduate students from around the world to attend the great event! In 2007, there are totally 11 tracks, and 36 paper sessions. Paper presentations are arranged mainly in four parallel tracks. There are also ASP-DAC-associated tutorials running on January 23. In ASP-DAC 2007, I served as a technical program committee member and had one invited paper and one regular paper for oral presentation.

I left for Yokohama, Japan, on January 23 and arrived on the same day. On January 24 afternoon. I chaired a special session on embedded software for multiprocessor systems-on-chip. On January 25 morning, I chaired anther session on system-level modeling. In ASP-DAC 2007, we have two papers being orally presented. The first paper was an invited paper, which was entitled as “Energy-Efficient Real-Time Task Scheduling in Multiprocessor DVS Systems.” In the paper, we consider energy-efficient scheduling for real-time tasks in multiprocessor DVS systems. Dynamic voltage scaling (DVS) circuits have now been widely adopted in many computing systems to provide tradeoff between performance and power consumption. The effective use of energy could not only extend operation duration for hand-held devices but also cut

down power bills of server systems. Moreover, while many chip makers are releasing multi-core chips and multiprocessor system-on-a-chips (SoCs), multiprocessor platforms for different applications become even more popular. Multiprocessor platforms could improve the system performance and accommodate the growing demand of computing power and the variety of application functionality. This paper summarizes our work on several important issues in energy-efficient scheduling for real-time tasks in multiprocessor DVS systems. Distinct from most previous work based on heuristics, we aim at the provision of approximated solutions with worst-case guarantees. The proposed algorithms are evaluated by a series of experiments to provide insights in system designs.

Note that energy-efficiency has become an important design issue in a wide range of computer systems in the past decade, such as servers, PDA's, and phones. The pursuing of energy-efficient designs could not only extend the operating time of a mobile device but also save energy bills of computer systems. Dynamic voltage scaling (DVS) has now been widely adopted in many computing systems, such as those based on Transmeta Crusoe, Intel Xeon and Mobile AMD Duron. Technologies, such as LongRun, Intel SpeedStep, and AMD PowerNow!, are also introduced to provide dynamic voltage scaling for laptops to prolong the battery lifetime.

Our second presentation is to minimize the average flow time of a set of jobs under a given energy constraint, where the flow time of a job is defined as the interval length between the arrival and the completion of the job. Note that power-aware and energy-efficient designs play

important roles for modern hardware and software designs, especially for embedded systems. This paper targets a scheduling problem on a processor with the capability of dynamic voltage scaling (DVS), which could reduce the power consumption by slowing down the processor speed. The objective of the targeting problem is to minimize the average flow time of a set of jobs under a given energy constraint. We consider two types of processors, which have a continuous spectrum of the available speeds or have only a finite number of discrete speeds. Two algorithms are given: (1) An algorithm is proposed to derive optimal solutions for processors with a continuous spectrum of the available speeds. (2) A greedy algorithm is designed for the derivation of optimal solutions for processors with a finite number of discrete speeds. The proposed algorithms are extended to cope with jobs with different weights for the minimization of the average weighted flow time. The proposed algorithms are also evaluated with comparisons to schedules which execute jobs at a common effective speed.

ASP-DAC is the most important and major forum in the hosting of researchers and engineers in computer-aided design and automation in the Asian and Pacific Region. It also attracts a lot of embedded systems researchers in recent years. It is an excellent platform in setting up forums for people with multiple disciplines in the academics and the industry. During the conference and after that, I did have a lot of chances to discuss with professors from Waseda University for future collaboration. The opportunity in attending ASP-DAC provides me a great chance to see more research in CAD/IP designs and embedded system designs. The experience will help me in identifying future research trends in the field of embedded systems.

2. Conference Proceeding and Related Documents

The conference proceeding and the CD ROM of ASP-DAC 2007 was brought back for future reference.