J. Low Power Electron. Appl. **2014**, *4*, 90-109; doi:10.3390/jlpea4020090

OPEN ACCESS

Journal of
Low Power Electronics and Applications
ISSN 2079-9268
www.mdpi.com/journal/jlpea

Article

A Practical Framework to Study Low-Power Scheduling Algorithms on Real-Time and Embedded Systems

Jian (Denny) Lin [1,*], Albert M. K. Cheng [2] and Wei Song [2]

[1] Department of Management Information Systems, University of Houston—Clear Lake, Houston, TX 77058, USA

[2] Department of Computer Science, University of Houston, Houston, TX 77204, USA; E-Mails: cheng@cs.uh.edu (A.M.K.C.); song.songwei@gmail.com (W.S.)

* Author to whom correspondence should be addressed; E-Mail: linjian@uhcl.edu; Tel.: +1-281-283-3187; Fax: +1-281-283-3951.

Received: 29 January 2014; in revised form: 18 March 2014 / Accepted: 10 April 2014 / Published: 7 May 2014

Abstract: With the advanced technology used to design VLSI (Very Large Scale Integration) circuits, low-power and energy-efficiency have played important roles for hardware and software implementation. Real-time scheduling is one of the fields that has attracted extensive attention to design low-power, embedded/real-time systems. The dynamic voltage scaling (DVS) and CPU shut-down are the two most popular techniques used to design the algorithms. In this paper, we firstly review the fundamental advances in the research of energy-efficient, real-time scheduling. Then, a unified framework with a real Intel PXA255 Xscale processor, namely real-energy, is designed, which can be used to measure the real performance of the algorithms. We conduct a case study to evaluate several classical algorithms by using the framework. The energy efficiency and the quantitative difference in their performance, as well as the practical issues found in the implementation of these algorithms are discussed. Our experiments show a gap between the theoretical and real results. Our framework not only gives researchers a tool to evaluate their system designs, but also helps them to bridge this gap in their future works.

Keywords: real-time/embedded systems; low-power; scheduling

---

1. Introduction

Recently, there has been an explosive growth in the popularity of portable and embedded devices. While power demands are increasing rapidly, battery capacity cannot keep up [1]. Power has been considered as the most important constraint in embedded systems, and how to manage it plays important roles. Managing energy consumption introduces a variety of benefits, such as the prolonged life of the battery in mobile devices, the reduced cost of energy and a low operating temperature. Today, low-power design is a widely discussed topic from the low circuit level to the high system synthesis level. This work is a study of low-power and energy-efficient design at the system level, more specifically embedded and real-time systems.

With the advanced technology of VLSI circuit designs, the dynamic voltage scaling (DVS), which adjusts the supply voltage and frequency dynamically, becomes an important power management technique in computer architecture. Modern well-known CPUs in embedded systems (Intel StrongARM [2], Xscale [3]) have adopted the DVS technique. The main sources of the power consumption of the CMOS circuits in a DVS processor are dynamic power, short circuit power and leakage power consumption. The dynamic power results from the charging and discharging of gate switching, and is dependent upon the processor's speed, s, which can be approximated by:

$$P_{dyn}(s) = C_{ef} V_{dd}^2 s \tag{1}$$

where $s = K\frac{(V_{dd}-V_t)^2}{V_{dd}}$. The C_{ef}, V_t, V_{dd} and K denote the effective switch capacitance, the threshold voltage, the supply voltage and a hardware-design-specific constant, respectively. The short-circuit power is proportional to the supply voltage. As a result, the dynamic power consumption of the DVS processor has a quadratic dependency on the supply voltage, which can be reduced by lowering the supply voltage and, thus, the speed. The leakage power is independent of the processor's speed, which is the power consumed by unintended leakage that does not contribute to the processor's function. When a processor is idle, it may be temporarily shut down to reduce the overall power consumption.

Many applications running on embedded systems are real-time tasks in which the task response time is an important requirement. A real-time task is expected to complete its execution before its deadline to maintain the system stability (e.g., the control tasks). If the timing resource is not 100% utilized on a system, there are some idle intervals in CPU cycles. These CPU idle cycles, termed as slack, can then be used to slow down the execution of a real-time task (or shut down the CPU), and its deadline can still be met. It is not hard to see that minimizing the energy consumption, as well as satisfying the multiple tasks' real-time requirements need a careful calculation to schedule the tasks. Figure 1 demonstrates a general idea in power-aware scheduling in real-time systems. In the figure, Schedule 1 is a schedule of the three tasks in Table 1 running at full speed (s = 1.0), and Schedule 2 is a schedule where the speed of the processor is slowed down by 50%. The idle intervals in Schedule 1 may be used to shut down the processor if they are long enough. Generally, a processor is temporarily shut down only if the idle interval is sufficiently long. This is because a processor's shut-down and wake-up will incur overheads.

Considering the potential ability in power saving, power-aware scheduling using DVS or/and CPU shut-down has received a lot of attention in the past few decades. A large number of algorithms have been proposed to complete the goal of power-saving using different models and considering different

scenarios. Although most algorithms are shown to be quite effective under their own experimental settings, almost all of their performance results are obtained solely via simulations. The major restriction of using simulations is the difficulty in determining the real performance, because many unpredictable and non-modeled factors may affect the results. The minor restriction could be the hardness in comparing different designs, making it difficult for developers to select an appropriate energy-efficient algorithm for a given application/system.

Figure 1. An example of power-aware scheduling using a slowed-down speed.

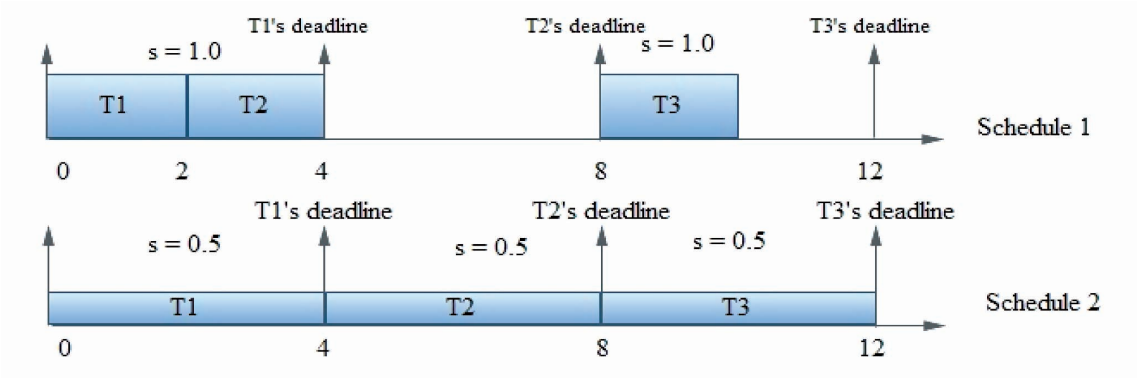


Table 1. Three real-time tasks.

Task no.	Ready time and deadline	Execution time
T1	(0, 4)	2
T2	(2, 8)	2
T3	(7, 12)	2

This paper extends and enhances our prior work in [4], which designs a complete framework, including hardware settings and software implementation, to evaluate power-aware scheduling algorithms on a real platform. Our work shows a gap between the theoretical results and the practical results. We not only evaluate low-power system designs, but also provide a way to bridge the gap in one's future work. The rest of the paper is organized as follows. In Section 2, we review the important advances in real-time, power-aware scheduling research works. Then, a novel, unified framework, called Real-Energy, is proposed in Section 3 to implement the power-aware algorithms and evaluate their performance. Section 4 presents a case study of the real performance about several classical algorithms using DVS or/and CPU shut-down. The energy efficiency and the quantitative difference in their performance, as well as the practical issues found in the implementation of these algorithms are discussed. The last section gives a summary.

2. Advances in Power-Aware Scheduling

2.1. Fundamental Techniques for Power-Aware Scheduling

The first important work in power-aware scheduling can be traced back to 1995 in a paper done by Yao *et al.* [5]. In the work, the authors study a scheduling problem of minimizing the CPU's energy

consumption for a set of real-time, single-instance or non-repeating tasks, when the earliest deadline first (EDF) algorithm is used to schedule tasks. Yao's algorithm is a greedy, off-line algorithm, which recursively calculates the highest intensity in a set of time intervals where an interval $[t_1, t_2]$ defines the executions of tasks released not earlier than t_1 and have a deadline not later than t_2. After the heaviest interval is identified, the intensity ratio, workload/interval's length, is used as the speed of the processor in this interval to execute tasks, and this interval is removed from the set of intervals. This process repeats, until all tasks are assigned a processor speed to run. For example, in Figure 1, there are six intervals, (0, 4), (2, 8), (0, 8), (2, 12), (7, 12) and (0, 12). The highest intensity ratio is the one of (0, 12), which is 0.5. Since the interval includes all of the three tasks, a processor speed of 0.5 is used to run all of them.

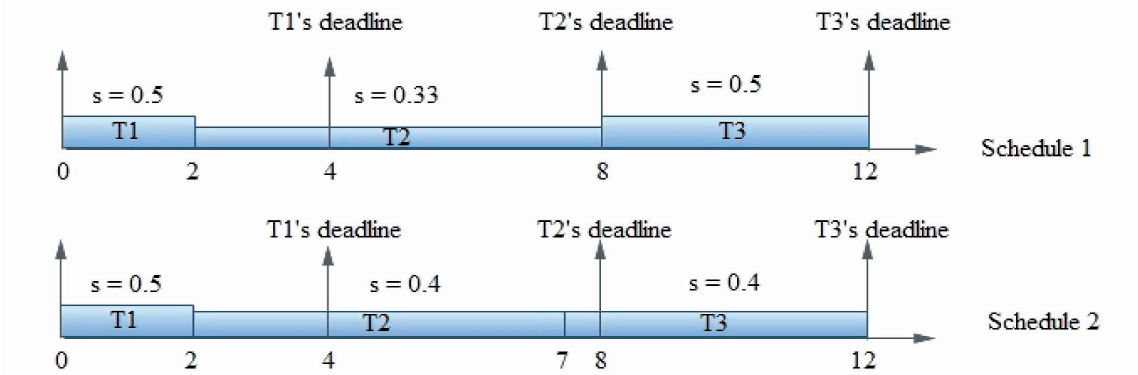
A periodic task is another important task model for real-time systems in which a series of job instances of a task arrive continuously at a certain rate. An important type of real-time task running on embedded systems, control tasks, can be modeled as periodic tasks. Three fundamental parameters generally define a periodic task, the maximum (or worst case) execution time, c; the period or rate, p; and utilization, u, where $u = c/p$. Aydin *et al.* prove that for all tasks scheduled by EDF, using a common speed, $s = U$, where $U = \sum u$, minimizes the CPU energy consumption of a set of periodic tasks if the power consumption function is convex [6].

EDF is a dynamic priority scheduling algorithm, such that the priorities of each task to be scheduled change during execution. There is another type of scheduling algorithm, called fixed-priority, such that each task's priority for scheduling does not change during the execution of the system. Rate-monotonic (RM) [7] is the optimal fixed-priority algorithm in which a task with a shorter period is assigned with a higher priority in scheduling. The common processor speed used for a set of periodic tasks scheduled by a fixed-priority, for example RM, is calculated in [8,9]. The calculation utilizes the schedulability analysis for fixed-priority scheduling based on the critical instant theorem, which says that if a task meets its deadline whenever the task is requested simultaneously with requests for all higher priority tasks, then the deadline will always be met for all task phasings [10,11]. A lowest speed ratio is calculated for every task using RM, such that the task and all of its higher priority tasks are schedulable to meet their deadlines using that speed. The highest ratio among the calculated speed ratio of each task is used as the common speed for all tasks.

The above techniques calculate a static speed off-line by assuming the worst case execution time (WCET) of real-time tasks. In real-time systems, the use of the WCET is to guarantee the deadline requirement, but it is conservative, because during the run-time, most job instances usually spend less time than their WCET in completing the execution. An idle interval that is generated by the early completion of a task, called on-line slack, can be used to further reduce the speed of the processor. A good review is done to survey the techniques to exploit on-line slack in [12]. The stretching-to-NTA (Next Task Arrival time) technique, used in the algorithms of *lppsEDF/RM* in [9], is based on a slack up to the arrival time of the next task. While in this technique, the slack can be exploited only if one task is in the ready queue, another one, The *ccRM* algorithm [8], theoretically improves the result by distributing the slack to multiple tasks. Utilization updating is used in *ccEDF* and *laEDF* [8], based on an observation that the actual processor utilization during runtime is usually lower than the worst case. The algorithm, *laEDF*, takes a more aggressive approach by estimating the amount of work

required to be completed before NTA. The *DRA* algorithm [13] exploits the priority-based slack stealing, such that when a higher-priority task completes its execution earlier than its WCET, the following lower-priority tasks can use the slack time from the completed higher-priority task. Figure 2 shows an example of how on-line slack can be used in a power-aware schedule. Assume that after the common speed, 0.5, is used, task T1's actual execution time is one unit under the full speed. T1 completes at Time 2, and the dynamic slack of two time units can be used to reduce the speed of T2 to 0.33, as in Schedule 1. If the on-line slack is distributed to be used by both T2 and T3, the two tasks both execute at a speed of 0.4. Considering that the power function is convex, Schedule 2 is theoretically better than Schedule 1 in energy minimization.

Figure 2. An example of power-aware scheduling using on-line slack.



There are some other, additional considerations when scheduling a set of real-time tasks. For examples, there exist jobs that are non-preemptive. In [14], the authors describe a scheduling technique that provides an optimal schedule in terms of power consumption on a uniprocessor platform running single-instance, non-preemptive jobs. The problem of scheduling periodic tasks is studied in [15] in which an algorithm that can produce lower speeds for a variety of non-preemptive task sets is proposed. Furthermore, computer jobs can be executed on multiprocessors in order to enhance the throughput. The low-power design problem has been extensively studied for homogeneous multiprocessor systems [16–18] and heterogeneous multiprocessor systems [19–21].

While the DVS algorithms focus on the reduction of dynamic power consumption, the static one, leakage power, is no longer negligible today. When a processor is idle for execution, it may be temporarily shut down to reduce the static power consumption and wake up later to execute other jobs. In real-time, power-aware scheduling, another fundamental technique is to delay future jobs' execution as much as possible in order to create a long idle interval to shut down the processor. This technique, called procrastination, can reduce the frequency of shut-down and wake-up and, hence, reduce the overhead, including energy consumption. Leakage-aware algorithms proposed in [22–25] address procrastination scheduling to safely delay the execution of tasks in order to increase the idle interval for CPU shut-down. The general idea in these works is first to calculate an allowed idle interval for each task, and then update the wake-up time according to the dynamic arrivals of tasks. Some other works [26,27] combine the DVS with the shut-down technique. They assume that there exists a critical speed, in which executing any task higher or lower than this speed, both consume more energy. If a CPU speed is calculated as lower than the critical speed, the critical speed is used as the CPU's speed. While there is no job ready,

the longest duration is calculated and used to shut down the CPU, so that the incoming jobs will not miss their deadlines.

Recently, energy harvesting, a technology that allows one to capture otherwise unused ambient energy and convert it into electrical energy that can be used immediately or later, is considered in real-time power scheduling [28,29]. When using ambient energy to support a system's execution, a challenge is that the energy source may be unstable and change from time to time. The research presented in [30] develops an energy-aware DVS algorithm, the task execution of which is slowed down if the system does not have sufficient available energy; otherwise, the tasks are executed at full speed. This work is improved later in [31] in which task executions are slowed down whenever possible for energy saving and the on-line slack are exploited by all tasks in the ready task queue. So far, the power-aware scheduling with energy harvesting makes some assumptions, such as a predictable and well-defined power-gaining function, which are difficult to apply on a real platform.

2.2. Evaluations of the Performance of Power-Aware Algorithms

In order to apply the algorithms to real systems, we are curious about their true performance. There have been several existing works to evaluate the performance of power-aware algorithms. Kim *et al.* in [32] compare several DVS algorithms for hard real-time systems, but their work uses simulations to obtain the experimental results. For measuring real energy consumption, a project called PLEB 2 developed by David *et al.* is shown in [33]. The system is specialized for observing energy distribution on the board when running applications. Although PLEB 2 was used to study the DVS technique, the following reasons limit its usability in other settings. First, designing and producing such a system requires a significant modification to the integrated circuits on the board. This may be beyond the scope of many researchers' work in developing algorithms. Second, PLEB 2 does not have a software framework for implementing real-time scheduling algorithms, and thus, it cannot be used to run multiple real-time tasks. There are several other works on estimating the power consumption for the DVS technique. Miyoshi *et al.* [34] examine a set of micro-benchmarks on two different systems. They find that the lowest speed is not always the best for saving energy. In [35], on an Xscale-based computer, Weissel monitors the number of memory references and instructions executed at runtime in order to estimate the reduction in frequency. Due to a lack of a framework for programming, these works have a similar drawback of PLEB 2, that they are not suitable for evaluating real-time scheduling algorithms.

3. Real-Energy

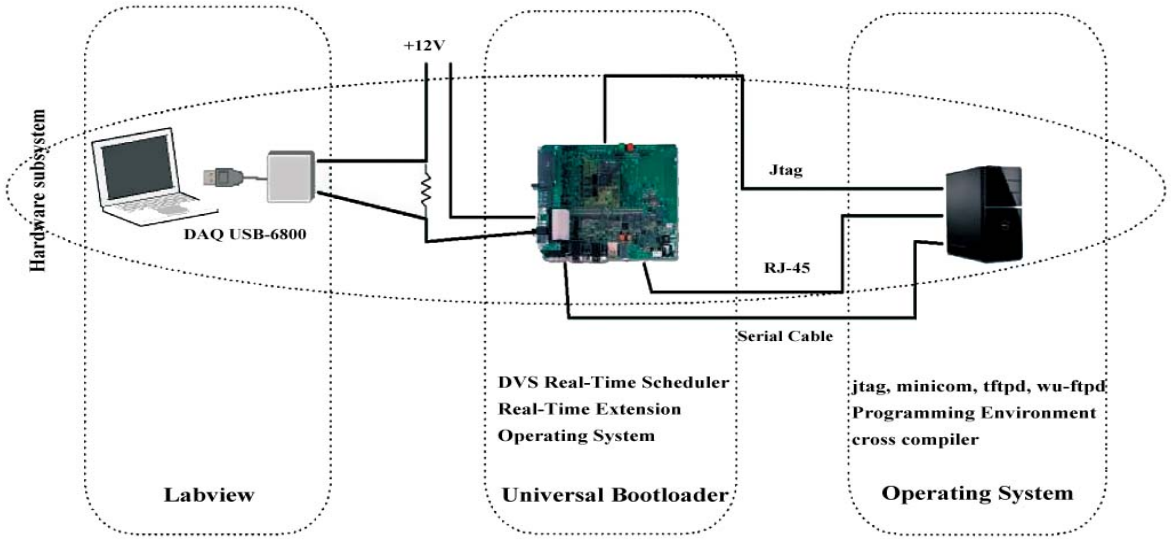
In this section, we describe the Real-Energy framework. Although the framework is based on an Intel PXA255 XScale processor, the methodology and programming design can be easily applied to other similar platforms.

3.1. Environment Setup

Figure 3 shows the environment's outline. Our hardware platform subsystem includes a PYTHECT phyCOREr PXA255 rapid development kit, a desktop and a data acquisition (DAQ) USB-6800 from National Instruments. The rapid development kit is the target platform in our embedded development

environment. The desktop is our host machine. The computer program is developed, compiled and linked on the host machine before it is downloaded to the target platform. DAQ USB-6800 is a data collection device used to measure the power consumption of the target platform. The data is stored on a computer and analyzed by the LabView software tool.

Figure 3. Real-Energy's environment setup. DAQ data acquisition; DVS, dynamic voltage scaling.



For the programming environment, we use Linux 2.14.19 as the operating system (OS). The patch RTAI-24.1.3 is used, where RTAI stands for Real-Time Application Interface, which is a real-time extension for the Linux kernel. We extend the RTAI scheduler to a power-aware enabled scheduler for accommodating various low-power scheduling algorithms. There are three connections from the host to the embedded board: A JTAG-to-parallel connection is used to download the boot loader (Universal Bootloader U-Boot-1.0.2); A serial cable connection is used to download the OS and file system image; An Ethernet connection is used to download the real-time applications.

The PYTHECT phyCOREr PXA255 rapid development kit is available on the market and designed for use in industrial applications that require high processing speed with low power. It includes a carrier board and a PXA255 computing module. All that is required for operation is a single 3.3 V supply voltage. A 128 MB SDRAM and a 64 MB flash memory are integrated onto the sub-miniature (70 mm $\times$ 57 mm) system on module. An SPI (Serial Peripheral Interface) port expander with 28 GPIOs (General-purpose input/output) for matrix keypads and a touch screen controller are also available on the PXA255 module. A 10/100 Mbps Ethernet and a CAN controller also populate the module, enabling the connection of the module to networks. To support the programming and analysis for applications running on the module, the computing module is designed to plug into a carrier board that provides I/O connectors, such as DB-9, RJ-45, USB and power jack, as well as any other interface circuitry not provided on the module itself. In order to focus on the interested portions, we have disabled the ports and controllers that we do not use in this work, such as the 10/100 Mbps Ethernet, CAN controller, USB, *etc.* The LCD touch screen is also shut off. Intel PXA255 runs at frequencies from 99.1 to 398.1 MHz. The

number of supported frequency settings is fixed. Table 2 shows four supported CPU frequency settings under the same flash memory and RAM frequency, which are used in our work.

Table 2. Four supported frequency settings of Intel PXA255.

CPU frequency (MHz) at voltage (V)	Flash memory frequency (MHz)	SDRAM frequency (MHz)
99.1 MHz at 1.0 V	99.5 MHz	99.5 MHz
199.1 MHz at 1.0 V	99.5 MHz	99.5 MHz
298.6 MHz at 1.1 V	99.5 MHz	99.5 MHz
398.1 MHz at 1.3 V	99.5 MHz	99.5 MHz

3.2. Measuring Energy Consumption

The carrier board with the PXA255 computing module plugged in is connected to an external stabilized voltage supply. In order to measure the energy consumed by the computing module, we measure the energy used by the development kit (the carrier board and the computing module). To determine the energy consumed by the kit, we attach a small resistor (0.02) to the wire connecting to the external voltage supply. The voltage drop across this resistor is measured by data acquisition (DAQ). The DAQ reads and records the voltage drop 50 times per second. The instantaneous current through the system can be calculated by dividing the voltage by the resistance, and the instantaneous power consumption of the development kit can be calculated as:

$$P_{kit} \approx I \times V_{external} \tag{2}$$

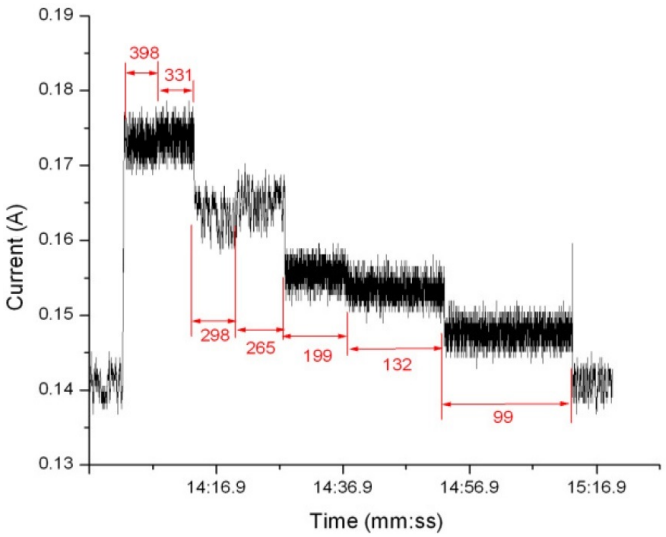
where I is the recorded current used by the kit and $V_{external}$ is the external supply's voltage to the kit. Figure 4 shows an example of the current measured for running a single instance task at different CPU frequencies. We observe that the instantaneous current, therefore the instantaneous power, at a low 331 (265) MHz is higher than that at a high 398 (298) MHz. This is because the memory frequency in the setting of 331 (265) MHz is higher than the one used in the setting of 398 (298) MHz [3]. This observation reminds us of a practical issue, that a CPU frequency setting may be related to a specific memory (flash memory) frequency on a real platform, and these combination settings must be considered in the design of low-power algorithms.

3.3. CPU Energy Consumption and System-Wide Energy Consumption

The original goal of DVS is to reduce the energy consumption of the processor. This section gives the methods for evaluating the DVS algorithms for both the CPU energy and system-wide energy (the energy used by the module) consumptions. Generally, the system-wide energy consumption gives a higher hierarchical understanding in view of saving energy, so that the reason for measuring it is obvious. However, we also expect to investigate the impact of a proposed scheduling algorithm and the application workload on different components of the hardware (e.g., the CPU). Then, we can fine-tune the algorithm and application to accomplish energy savings. Since the CPU is a major energy-consuming component,

evaluating the performance of scheduling algorithms and applications in reducing the CPU's energy consumption is a major step toward this goal.

Figure 4. An example of the current measured.



As stated earlier, the actual power we measure is the total power consumed by the development kit, which includes the computing module and the carrier board. Hence,

$$P_{kit} \approx P_{mod} + P_{board} \tag{3}$$

In Equation (3), P_{mod} is the power consumption of the computing module (considered as the target system) and P_{board} is the power consumption of the carrier board. Since most ports and controllers on the carrier board have been disabled, we can assume that the carrier board's instantaneous power consumption is stable and is a constant. P_{board} can be approximated to be the kit's energy consumption when the computing module is in the sleep mode. The sleep mode is the lowest power state with a very small energy usage, where only a real-time clock (RTC) and a power manager are active. Thus:

$$P_{board} \approx P_{kit}^{sleep} \tag{4}$$

and P_{kit}^{sleep} can be measured and known easily. For P_{mod}, the major sources of power consumption are: CPU, flash memory, SDRAM and leakage currents. Thus, P_{mod} can be defined as:

$$\begin{aligned} P_{mod} &\approx P_{CPU} + P_{flash} + P_{SD} + P_{leakage} \\ &\approx P_{kit} - P_{kit}^{sleep} \end{aligned} \tag{5}$$

In the system-wide evaluations, P_{mod} is what we want to know. However, for the algorithms targeting the CPU only, we need to find out P_{CPU}. Our solution is to run CPU-bound tasks on the system for measuring the kit's instantaneous power consumption. In a CPU-bound task, an empty loop is executed. When the task is running, there are very few memory accesses. The energy consumed by the SDRAM and flash memory is primarily from the clocks and the leakage currents. The following equations define the instantaneous power usage by the module and kit when running CPU-bound tasks.

$$P_{mod}^{CPU-Bound} \approx P_{CPU} + P_{flash}^{idle} + P_{SD}^{idle} + P_{leakage} \tag{6}$$

$$P_{kit}^{CPU-Bound} \approx P_{board} + P_{mod}^{CPU-Bound} \tag{7}$$

On the other hand, when the CPU is idle, the development kit and the module actually are in the idle mode, too. Thus,

$$P_{mod}^{idle} \approx P_{CPU}^{idle} + P_{flash}^{idle} + P_{SD}^{idle} + P_{leakage} \tag{8}$$

$$P_{kit}^{idle} \approx P_{board} + P_{mod}^{idle} \tag{9}$$

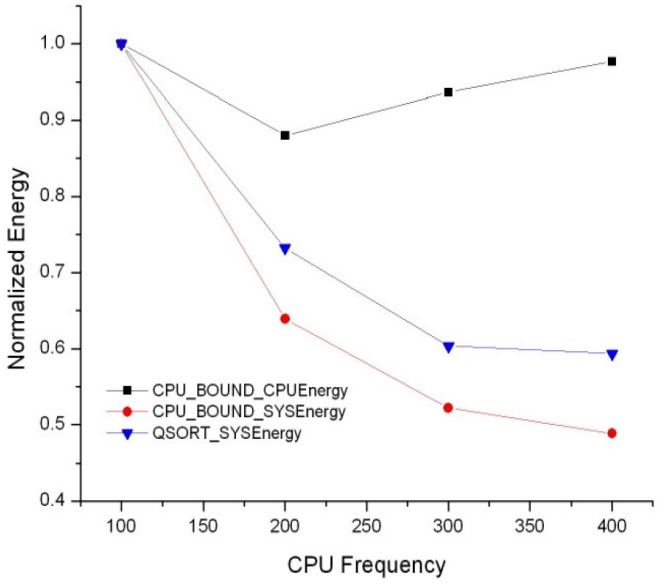
Now, subtracting Equation (9) from Equation (7), we have:

$$\begin{aligned} P_{kit}^{CPU-Bound} - P_{kit}^{idle} &\approx P_{mod}^{CPU-Bound} - P_{mod}^{idle} \\ &\approx P_{CPU} \end{aligned} \tag{10}$$

That is, the power consumption of a CPU can be obtained by measuring the power consumption of the development kit running a CPU-bound task and having the module in sleep mode. The small amount of the energy consumption of the CPU while it is idle is ignored. If the static power consumption needs to be considered, a system-wide energy measurement can be used.

In order to verify the effectiveness of our methods, a CPU-bound task and the QSort benchmark [33] are tested on our system for using DVS. The CPU's and the system-wide energy consumption are calculated according to the approach mentioned above. Figure 5 shows the normalized results for running these two tasks at the four supported frequency settings. It confirms that for Intel PXA255 using the lowest speed is not CPU energy efficient, and using a lower speed exhibits less efficiency in saving the system-wide energy.

Figure 5. Energy consumption using dynamic voltage scaling (DVS).



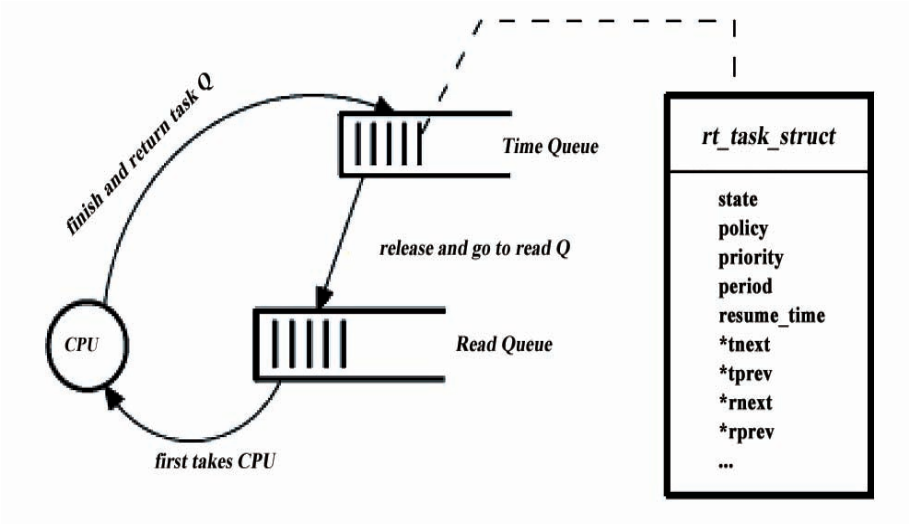
3.4. Programming Design

So far, the existing works to evaluate the real power consumption of a system do not have a programming design to evaluate real-time, power-aware scheduling algorithms. In this work, we extend

RTAI to work with DVS and CPU shut-down/wake-up. In what follows, we present several important aspects and techniques used in our framework.

Implementing a priority queue correctly is a critical part of designing a real-time scheduler. The work is easy for single-instance tasks, but for a periodic task, it requires additional efforts. In order to make a set of tasks periodically run, in our programming design, there are two task queues: time queue (TQ) and ready queue (RQ). After an initialization, a periodic task is pushed into TQ in which the tasks follow the nondecreasing order of the next job instance's arrival time. A time interrupt handler checks TQ at small, regular time intervals. If a task is ready to run, it is inserted into RQ and removed from TQ. Its position in RQ is decided by the scheduling algorithm the system uses (e.g., RM or EDF). The scheduler schedules tasks for execution based on the order in RQ. After a task is terminated, it is removed from RQ and pushed into TQ again. Figure 6 shows how these two queues work, as well as the task structure definition. The variable state in the task structure indicates the task status, such as READY, DELAYED and SUSPENDED. The policy is used to represent the energy-efficient scheduling policy. The period and resume time are release intervals of a periodic task and the release time of its next instance. Pointers *tnext, *tprev, *rnext and *rprev are pointed to the next or previous tasks in TQ and RQ, which form a double-linked list.

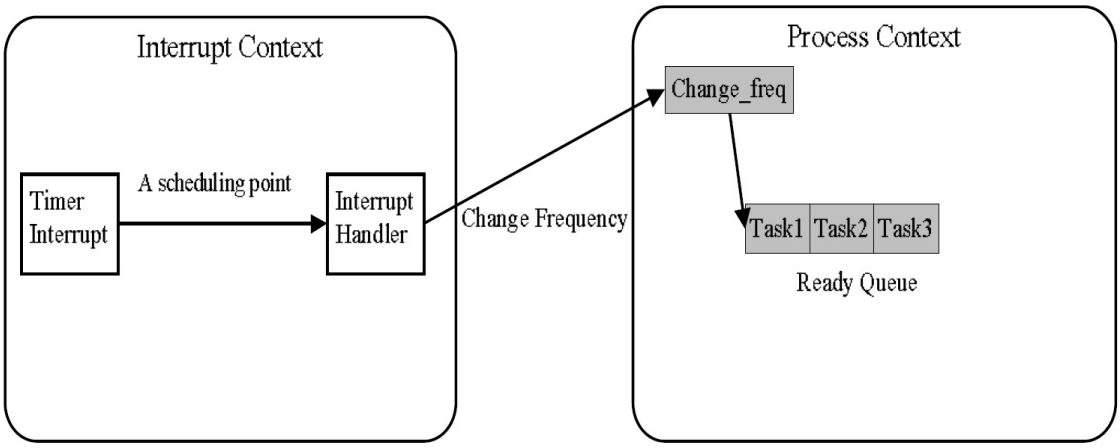
Figure 6. Time queue, ready queue and task structure.



When a real-time task arrives or completes, a real-time scheduler needs to make a scheduling decision. We call this point a scheduling point. In our work, the detection of a scheduling point is done by using a timer interrupt. An interrupt handler is called when a scheduling point is found. For an inter-task, power-aware algorithm, changing the CPU's frequency or putting it into sleep mode also happen at a scheduling point. An intuitive way is to implement the code as part of the interrupt handler. However, in Linux, a change of the CPU's frequency (including putting the CPU into sleep mode) is not allowed in the interrupt context (an interrupt handler runs in the interrupt context). To overcome this obstacle, we create a special task handling the job and then make this task run in the process context. The implementation of our technique is described as in Figure 7. A special task to change the CPU's frequency is set with the highest priority, and its default status is suspended. At a scheduling point, if a change of frequency is called, the interrupt handler activates the task and inserts it into the head of the RQ. In this way, the CPU

frequency can be immediately changed before the next real task is dispatched. The special task's status is set to be suspended after the change of frequency is done. This implementation is also applicable to intra-task DVS algorithms, where the change of the CPU's frequency may take place at any predefined point.

Figure 7. The process of changing the CPU's frequency.



In a CPU's sleep mode, all processor and peripheral clocks are disabled, except the RTC (real-time clock). The processor does not recognize interrupts or external pin transitions, except valid wake-up signals and reset signals, *etc.* In practice, a wake-up of a system is normally done by a hardware signal, e.g., touching the screen, moving the mouse, *etc.* Apparently, this hardware mechanism cannot be used with real-time tasks. Since only an RTC is active when the system sleeps, we utilize the RTC alarm to set up the wake-up time. While RTAI blocks the RTC alarm from running, we fortunately found that the blocking mechanism actually can be dynamically mounted, as well as unmounted.

We have integrated several key DVS and shut-down algorithms into the Real-Energy framework being used in our case study. Our programming design is module oriented and fully extensible to other scheduling algorithms. When one wants to evaluate a new algorithm, the new one can be written in a new module and added to the selections for the policy.

4. A Case Study

In this section, we perform a case study using our proposed framework, Real-Energy. In the study, we restrict our attention to study preemptive, periodic tasks scheduled by EDF and RM, because these two scheduling algorithms are the most used ones in the literature. The algorithms use either DVS or CPU shut-down to save energy. In the DVS algorithms, the main difference is in how they exploit the on-line slack times. The CPU's energy consumption is used to compare their effectiveness. For the algorithms using CPU shut-down, we mainly study the so-called procrastination algorithms, which delay task execution for putting the CPU into sleep mode. A practical issue is discussed in implementing the algorithms, and we bring a solution to it.

4.1. Experimental Setting

Considering the lowest frequency setting in Intel PXA255 is not energy-efficient, as shown in Figure 5, we use the other three frequencies in Table 2 to evaluate the DVS scheduling algorithms. The calculated speeds by the algorithms are used to determine the CPU's frequency. For example, if the calculated speed is 0.38, the frequency of 199 MHz is used to execute the tasks. In our experiments, ten tasks are created to run on the system. The periods of the tasks range from 50 to 120 units, where one unit is equal to 25 ticks period (1 tick period = 2 ms). The worst case utilization of these ten tasks are randomly generated, but the total utilization is assured to be less than 100%. For evaluating the algorithms under different workloads, we use loops in a task, and the number of loops is used to control the generation of slack time. The WCET is measured off-line, and we increase it by 5% to accommodate the run-time overhead.

4.2. Experimental Results for DVS Algorithms

Six key DVS algorithms (as introduced in the previous section) listed in Table 3 are implemented and compared. These algorithms are classical in that a lot of the latter works are extensions of the former ones. The ideas of exploiting the on-line slack are still used in recent research works.

Table 3. DVS algorithms. EDF, earliest deadline first; RM, rate-monotonic.

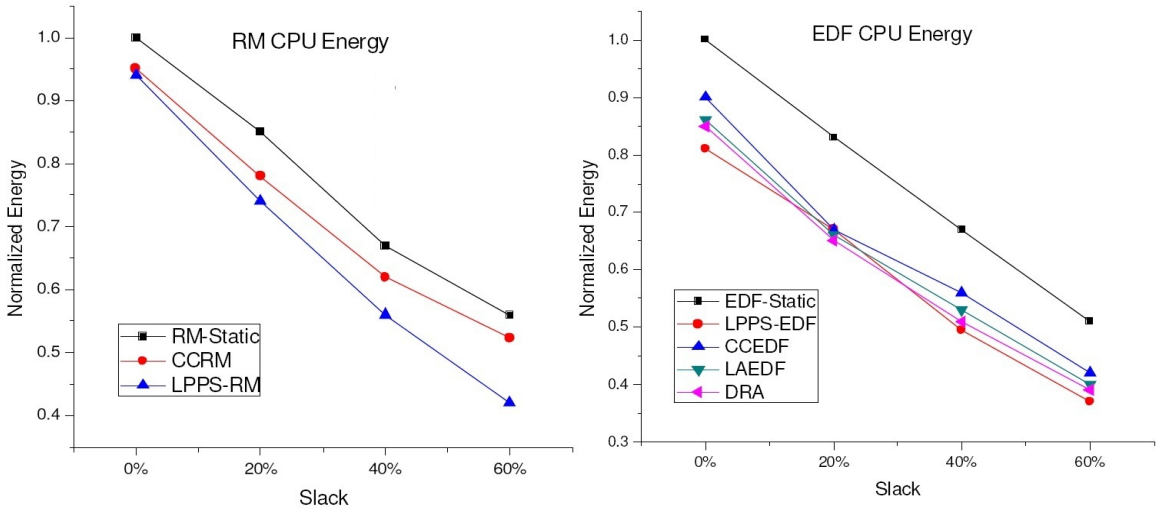
EDF	*lppsEDF* [9]; *ccEDF* [8]; *laEDF* [8]; *DRA* [13].
RM	*lppsRM*[9]; *ccRM*[8]

The experimental results for DVS algorithms are presented in two categories, RM or EDF, as shown in Figure 8. In RM, the results of using *lppsRM* and *ccRm* are normalized and compared to the result of using a fixed static speed without utilizing an on-line slack. These results are compared at four workload conditions, where the percentage of slack means the average ratio between the actual execution time (by adjusting the number of loops) and the WCET. It is easy to see that *lppsRM* and *ccRM* both effectively reduce the CPU's energy usage. According to what is reported in [32], *ccRM* is better than *lppsRM* with the belief that distributing the on-line slack to multiple tasks enhances the power-saving performance. However, in our experiment, *lppsRM* obviously outperforms *ccRM*, which challenges the theoretical result. We believe that the disagreement is caused by the higher overhead in *ccRM*, which is not factored into the simulations in [32]. In these two algorithms, *ccRM* uses a more complicated mechanism to calculate the slack times at each scheduling point. These calculations consume energy, as well. When the system is heavily loaded, although the complicated method may be better for exploiting slack times, the slack is scarce, which could weaken this advantage. When the system is lightly loaded, since the slack times are easy to obtain, paying a high overhead may not be worthwhile. Anyway, the results of the experiment tell us that the overhead from the algorithm itself is not negligible, and a simple design sometimes beats an advanced design.

In the EDF's results, the four EDF algorithms are more aggressive than the RM algorithms in reducing the CPU's energy consumption. This is because the current technique used to guarantee tasks' deadline

in RM is more conservative than EDF, so that less slack can be used to lower the speed in RM. Similar to the results of RM, the simple algorithm, *lppsEDF*, has the best performance.

Figure 8. CPU energy consumption for the RM and EDF algorithms.



We are also interested in how much of the total energy is consumed by the system when only the CPU is working. This information is useful to analyze the energy usage contributed by the idle memory, leakage currents, *etc.* Figure 9 shows the measured system-wide energy consumption for using the RM and EDF algorithms when running CPU-bound tasks. It can be seen that the performance gaps between the algorithms and their norm (the curve of using static speed) are smaller than those in Figure 8. This is because using the DVS technique cannot reduce the energy consumption from other units. Figure 10 shows the average energy usage distribution (at 40% slack) for using the EDF and RM algorithms, where the usage for other parts is calculated by subtracting the CPU's energy usage from the total system's usage. It is apparent that the energy used by other parts on the system on chip can no longer be ignored.

Figure 9. System-wide energy consumption for the RM and EDF algorithms.

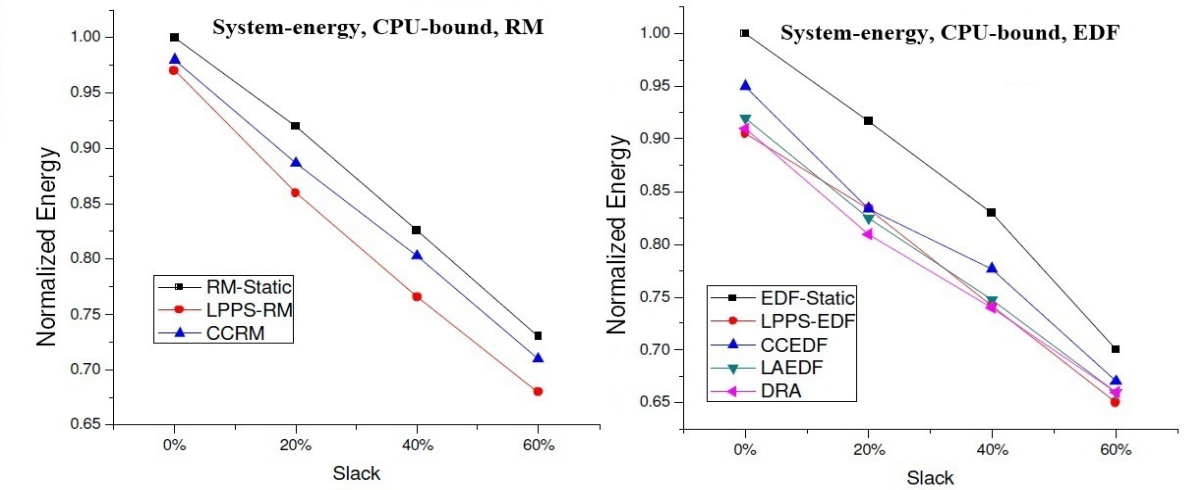
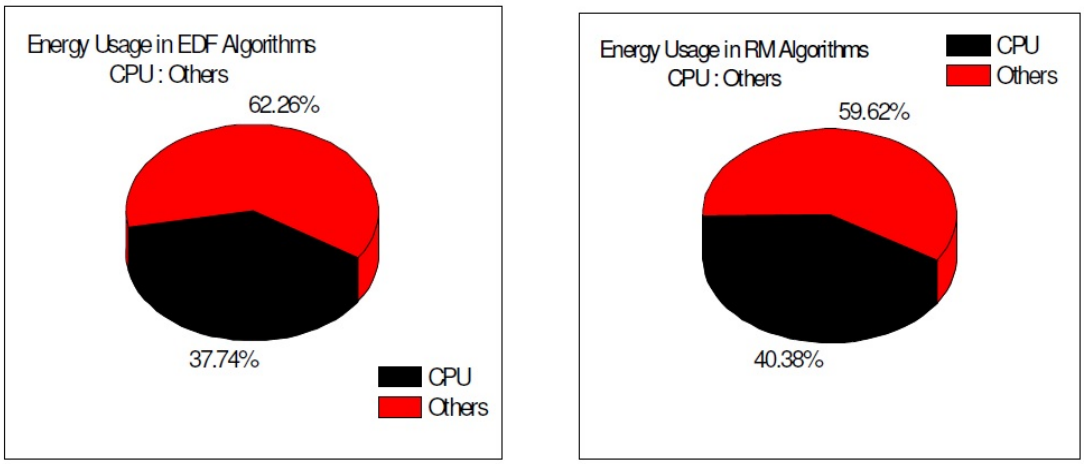


Figure 10. Energy distribution for the EDF and RM algorithms.



4.3. Experimental Results of Procrastination Algorithms

Procrastination algorithms [22–25] are used to temporarily shut down the processor in order to reduce the overall energy consumption, including the static one. During our study of applying the algorithms to a real hardware platform, we found that a practical issue existed in these procrastination algorithms: they all somehow rely on a calculation by a sleeping CPU. For example, in [22], the accumulated utilization needs to update during the sleeping interval (algorithm *LCEDF*) to determine a delay, Y_i, for a task, T_i; in [23–25], a function is performed to update the wake-up timer when the CPU is in sleep mode. Obviously, these computations require a CPU to be involved unless an additional computing unit undertakes the jobs, and this unit, if available, consumes energy, as well. Hence, either the additional energy usage should be studied or the algorithms should be revised to apply them on current commercial processors, where a CPU is not clocked during the sleep mode.

The revised algorithm, practical-leakage-current (PLC), of the algorithm, LC, in [22] is shown in Algorithm 1. In PLC, no computation is performed during the CPU's sleep state. Upon the completion of a task, the system keeps the CPU busy if any remaining task is in the ready queue. Otherwise, the CPU enters the sleep state, and the wake-up time is set to $NTA + Y$, where Y is the minimum value among the idle times calculated off-line for each task. Without updates during sleep mode, using the minimum value, Y, guarantees that it does not cause any task to miss its deadline.

Even with the revised algorithm, PLC, the procrastination technique can only be used for those real-time tasks for which the parameters are known in advance. For real-time tasks having dynamic arrivals, since the system cannot accept any task during the sleep mode, they may be lost due to the CPU's shut-down. Currently, in most existing works, this practical issue is not paid enough attention.

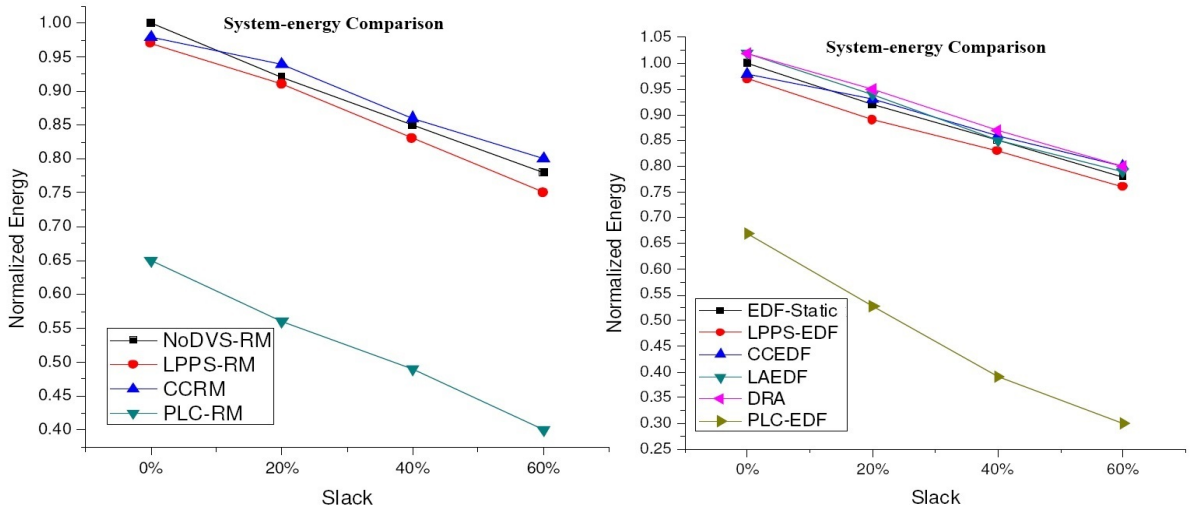
Considering the re-booting overhead, a CPU enters into the sleep state only if it is profitable to do so. In our experiment, we set this overhead to 2 ms. We compare our PLC-RM and PLC-EDF algorithms with their corresponding DVS algorithms, as shown in Figure 11. In these experiments, we use non-CPU-bound tasks in which the accessing of memory is frequent. As seen from the results, all of the DVS algorithms do not succeed in reducing the system-wide energy consumption. In some cases, the slow-down of the processor speed even increases the energy consumption of the system. This is because

when the CPU is active with a longer period, the other components have to be actively working, as well. While today, the CPU's energy consumption no longer dominates in a system, using the DVS alone to slow down the execution of computer tasks may not be a good solution for saving energy. Nevertheless, the procrastination algorithms using the CPU shut-down technique are very effective at achieving this goal.

Algorithm 1: Revised procrastination algorithms.

On a task's arrival:
  If (CPU is active)
    Insert the task into the ready queue
  Else
    Do nothing
  End
On a completion of a task:
  If(the ready queue is not empty)
    Execute the highest priority task in the queue
  Else
    The CPU enters into the sleep state
    The wake-up time is set at $NTA + Y$
  End
On a CPU's wake-up:
  Begin executing the tasks in the ready

Figure 11. Comparisons of the RM and EDF procrastination algorithms.



5. Conclusion

In designing power-aware algorithms, it is always important to know the real performance, because so many factors can affect the results. This motivates us to design a unified framework for the evaluations in spite of the complication. Our work does a comprehensive job of completing such a tool in hardware

and software. We not only use the tool for evaluation, we also can use it to test the implementation of algorithms in order to find out practical issues. In our case study, while the effectiveness of the DVS technique for reducing the CPU's energy consumption is shown, it also confirms that it does not effectively reduce the system's energy consumption. The CPU shut-down is a simple and effective one to save the system's energy consumption, but so far, it is only applicable for running tasks with a predefined release time. Otherwise, the CPU does not know when to wake up to continue the execution. We do not implement or evaluate the algorithms using a critical speed. The use of the critical speed requires a continuous range of the CPU speed, which is unpractical today on any platform. Therefore, there has been a focus on thermal-aware scheduling recently. Our framework can be used for the topic by finding a way to measure a CPU's temperature.

The Real-Energy framework is not specifically hardware and software dependent. The OS, Linux, with programming in C is widely used in embedded systems. The development kit and data collection device and their replacements are easily found on the market. It does not need any hardware modification. When a newer generation of hardware components appears, they can be easily used in the framework, as long as they have a similar architecture and functionalities.

At the point of writing this paper, Real-Energy is being used for a single-CPU system. Because of the lack of hardware support on the market, we have not found a suitable multiprocessor system to be used in our framework. If the support in the hardware is available, we will be interested in investigating the real performance of the algorithms working with a multiprocessor/multi-core system, such as those in [36,37].

Acknowledgments

This work was supported in part by the National Science Foundation under Award No. 0720856.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Ditzel, M.; Otten, R.; Serdijn, W. *Power-Aware Architecting for Data-Dominated Dpplications*. Springer: Berlin, Germany, 2007.
2. *Intel StrongARM SA-1110 Microprocessor Developers Manual*. Intel Corporation: Santa Clara, CA, USA, 2000.
3. *Intel PXA255 Processor Developers Manual*. Intel Corporation: Santa Clara, CA, USA, 2004.
4. Lin, J.; Song, W.; Cheng. A. Real energy: A new framework and a case study to evaluate power-aware real-time scheduling algorithms. In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), Austin, TX, USA, 18–20 August 2010.
5. Yao, F.; Demers, A.; Shenker. S. A scheduling model for reduced CPU energy. In Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Milwaukee, WI, USA, 23–25 October 1995.

6. Aydin, H.; Melhem, R.; Mosse, D.; Meja-Alvarez, P. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In Proceedings of the 22nd IEEE Real-Time Systems Symposium, London, UK, 2–6 December 2001.
7. Liu, J.W. *Real-Time Systems*; Prentice Hall: Englewood, Cliffs, NJ, USA, 2000.
8. Pillai, P.; Shin, K.G. Real-time dynamic voltage scaling for low power embedded operating systems. In Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, Banff, AB, Canada, 21–24 October 2001.
9. Shin, Y.; Choi, K.; Sakurai, K. Power optimization of real-time embedded systems on variable speed processors. In Proceedings of the 2000 IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, USA, 5–9 November 2000.
10. Liu, C.L.; Layland, J.W. Scheduling algorithms for multi programming in a hard real time environment. *J. ACM* **1973**, *20*, 46–61.
11. Lehoczky, J.; Sha, L.; Ding, Y. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In Proceedings of the IEEE Real-Time Systems Symposium, Santa Monica, CA, USA, 5–7 December 1989.
12. Lee, I.; Leung, J.; Son, S. *Handbook of Real-Time and Embedded Systems*; Chapman and Hall/CRC Press: Boca Raton, FL, USA, 2007.
13. Aydin, H.; Melhem, R.; Mosse, D.; Mejia-Alvarez, P. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In Proceedings of the 22nd IEEE Real-Time Systems Symposium, London, UK, 2–6 December 2001.
14. Andrei, S.; Cheng, A.; Radulescu, V.; McNicholl, T. Toward an optimal power-aware scheduling technique. In Proceedings of the 14th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, 26–29 September 2012.
15. Li, J.; Shu, L.; Chen, J.; Li, G. Energy-efficient scheduling in non-preemptive systems with real-time constraints. *IEEE Trans. Syst. Man Cybern. Syst.* **2013**, *43*, 332–344.
16. Chen, J.; Hsu, H.; Chuang, K.; Yang, C.; Pang, A.; Kuo, T. Multiprocessor energy-efficient scheduling with task migration considerations. In Proceedings of the IEEE EuroMicro Conference on Real-Time Systems, Catania, Italy, 30 June–2 July 2004.
17. Aydin, H.; Yang, Q. Energy-aware partitioning for multiprocessor real-time systems. In Proceedings of the IEEE 17th International Parallel and Distributed Processing Symposium, Nice, France, 22–26 April 2003.
18. Lin, J.; Cheng, A.M.K. Real-time task assignment in rechargeable multiprocessor systems. In Proceedings of the IEEE 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Kaohisung, Taiwan, 25–27 August 2008.
19. Schmitz, M.; Al-Hashimi, B.; Eles, P. Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems. In Proceedings of the IEEE Conference on Design, Automation and Test in Europe, Paris, France, 4–8 March 2002.
20. Lin, J.; Cheng, A.M.; Kumar, R. Real-time task assignment in heterogeneous distributed systems with rechargeable batteries. In Proceedings of the IEEE International Conference on Advanced Information Networking, Bradford, UK, 26–29 May 2009.

21. Luo, J.; Jha, N. Static and dynamic variable voltage dcheduling algorithms for realtime heterogeneous distributed embedded systems. In Proceedings of the 15th International Conference on VLSI Design, Bangalore, India, 7–11 January 2002.
22. Lee, Y.; Reddy, K.P.; Krishna, C.M. Scheduling techniques for reducing leakage power in hard real-time systems. In Proceedings of the EcuroMicro Conference on Real Time Systems (ECRTS), Porto, Portugal, 2–4 July 2003.
23. Jejruikar, R.; Gupta, P.K. Procrastination scheduling in fixed priority real-time systems. In Proceedings of the 2004 ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems, Washington, DC, USA, 11–13 June 2004.
24. Jekruikar, R.; Gupta, P.K. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In Proceedings of the ACM IEEE Design Automation Conference, San Diego, CA, USA, 13–17 June 2005.
25. Jekruikar, R.; Pereira, C.; Gupta, P.K. Leakage aware dynamic voltage scaling for real-time embedded systems. In Proceedings of the ACM IEEE Design Automation Conference, San Diego, CA, USA, 7–11 June 2004.
26. Irani, S.; Shukla, S.; Gupta, R. Algorithms for power savings. In Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, USA, 12–14 January 2003.
27. Lin, J.; Cheng, A. Energy reduction for scheduling a set of multiple feasible interval jobs. *J. Syst. Archit. Embed. Softw. Des.* **2011**, *57*, 663–673.
28. Chetto, M. A note on EDF scheduling for real-time energy harvesting systems. *IEEE Trans.Comput.* **2014**, *63*, 1037–1040
29. Chetto, M. Optimal scheduling for real-time jobs in energy harvesting computing systems. *IEEE Trans. Emerg. Top. Comput.* **2014**, doi:10.1109/TETC.2013.2296537.
30. Liu, S.; Qiu, Q.; Wu., Q. Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting. In Proceedings of the Design, Automation, and Test in Europe, Munich, Germany, 10–14 March 2008.
31. Liu, S.; Lu, J.; Wu, Q.; Qiu, Q. Harvesting-aware power management for real-time systems with renewable energy. *IEEE Trans. Large Scale Integr. (VLSI) Syst.* **2011**, *20* ,1473–1486 .
32. Kim, W.; Shin, D.; Yun, H.-S.; Kim, J.; Min, S.L. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In Proceedings of the Real-Time and Embedded Technology and Applications Symposium, San Jose, CA, USA, 24–27 September 2002.
33. Snowdon, D.; Ruocco, S.; Heiser, G. Power management and dynamic voltage scaling: Myths and facts. In Proceedings of the 2005 Workshop on Power Aware Real-time Computing, Jersey City, NJ, USA, 18–22 September 2005.
34. Miyoshi, A.; Lefurgy, C.; Hensbergen, E.V.; Rajamony, R.; Rajkumar, R. Critical power slope: Understanding the runtime effects of frequency scaling. In Proceedings of the 16th Annual ACM International Conference on Supercomputing, New York, NY, USA, 22–26 June 2002.
35. Weissel, A.; Bellosa, F. Process cruise control: Event-driven clock scaling for dynamic power management. In Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASE 2002), Greenoble, France, 8–11 October 2002.

36. Yang, C.; Chen, J.; Kuo, T. Preemption control for energy efficient task scheduling in systems with a DVS processor and Non-DVS Devices. In Proceedings of the 13th IEEE International Conferences on Embedded and Real-Time Computing Systems and Applications, Daegu, Korea, 21–24 August 2007.
37. AlEnawy, T.A.; Aydin, H. Energy-aware task allocation for rate monotonic scheduling. In Proceedings of the 11th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'05), San Francisco, CA, USA, 7–10 March 2005.