Energy-Conscious, Deterministic I/O Device Scheduling in Hard Real-Time Systems

Vishnu Swaminathan and Krishnendu Chakrabarty, *Senior Member, IEEE*

***Abstract*—Energy consumption is an important design parameter for embedded and portable systems. Software-controlled (or dynamic) power management (DPM) has emerged as an attractive alternative to inflexible hardware solutions. However, DPM via I/O device scheduling for real-time systems has not been considered before. We present an online I/O device scheduler, which we call low-energy device scheduler (LEDES), for hard real-time systems that reduces the energy consumption of I/O devices. LEDES takes as inputs a predetermined task schedule and a device-usage list for each task and it generates a sequence of sleep/working states for each device such that the energy consumption of the device is minimized. It also guarantees that real-time constraints are not violated. We then present a more general I/O device scheduler, which we call multi-state constrained low-energy scheduler (MUSCLES), for handling I/O devices with multiple power states. MUSCLES generates a sequence of power states for each I/O device while guaranteeing that real-time constraints are not violated. We present several realistic case studies to show that LEDES and MUSCLES reduce energy consumption significantly for hard real-time systems.**

***Index Terms*—Deadlines, dynamic power management (DPM), embedded systems, low-energy, low-power, multiple power states, real-time operating systems.**

I. Introduction

COMPUTING systems can be broadly classified into two distinct categories: general purpose and embedded. Embedded systems are usually intended for a specific application. Such systems tend to be I/O intensive, and many of them require real-time guarantees during operation. Examples of embedded systems include remote sensors, digital cellular phones, audio and video disc players, sonar, radar, magnetic resonance imaging medical systems, video telephones, and missile systems.

Power consumption is an important design issue in embedded systems. The power consumption, and the corresponding energy consumption, of the various components of an embedded system directly influence battery lifetime and, hence, the lifetime of the system. Therefore, system lifetime can be extended by reducing energy consumption in an embedded system. Decreased power consumption also results in higher component reliability. Many embedded systems tend to be situated at remote locations; the cost of replacing battery packs is high when the batteries that power these systems fail.

Power-reduction techniques can be viewed as being static or dynamic. Power can be reduced statically by applying compile-time optimizations to generate restructured, power-conscious machine code. Static techniques can also be applied at design time to synthesize power-optimized hardware. Dynamic power-reduction techniques are used during run-time to take advantage of variations in workload.

It is easy to see that the use of static power-reduction techniques causes the system to be relatively inflexible to changes in the operating environment. Although static techniques result in significant energy savings, recent research has focused on dynamic power-management techniques. These techniques usually take advantage of the features provided by the underlying hardware to obtain greater energy savings.

Dynamic power management (DPM) refers to the methodology in which power-management decisions are made at run-time to take advantage of variations in system workload and resources. Modern hardware designs provide several features to support DPM. These features include multiple *power states* in I/O devices and variable-voltage processors. Given these features, a DPM scheme can make decisions on changing the operating voltage of a processor [also called dynamic voltage scaling (DVS)], and switching devices to low-power *sleep* states during periods of inactivity. DVS makes use of the fact that energy consumption of the CPU is quadratically proportional to its operating voltage. I/O-centric DPM techniques identify time intervals where I/O devices are not being used and switch these devices to low-power modes during these periods. Such techniques can be implemented at both the hardware and software levels.

More recently, DPM at the operating system (OS) level has gained importance due to its flexibility and ease of use. The OS, having a global view of system resources and workload, can make more intelligent power-management decisions and can, therefore, provide greater energy savings than hardware-based DPM alone. The Advanced Configuration and Power Interface (ACPI) standard, introduced in 1997, allows hardware-power states to be controlled by the OS through system calls, effectively transferring the power reduction responsibility from the hardware (BIOS) to the software (OS) [1].

A number of embedded systems are designed for real-time use. These systems must be designed to meet both functional *and* timing requirements [4]. Thus, the correct behavior of these systems depends not only on the accuracy of computations but also on their timeliness. Systems in which the violation of these

Manuscript received June 4, 2002. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Grant N66001-001-8946, in part by a graduate fellowship from the North Carolina Networking Initiative, and in part by DARPA and the U.S. Army Research Office under Award DAAD19-01-1-0504. A preliminary version of this paper appeared in *Proceedings of the International Symposium on Hardware/Software Codesign (CODES)*, Copenhagen, Denmark, 2001, pp. 235–241. This paper was recommended by Associate Editor R. Gupta.

The authors are with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: vishnus@ee.duke.edu).

Digital Object Identifier 10.1109/TCAD.2003.814245

timing requirements results in catastrophic consequences are termed *hard real-time systems*. Any real-time scheduling algorithm must guarantee timeliness and schedule tasks such that the deadline of every task is met. Energy minimization adds a new dimension to these design issues.

In modern computer systems, the CPU and the I/O subsystem are two significant consumers of power. While reducing CPU power results in significant energy savings, the I/O subsystem is also a potential candidate to target for power reduction in systems that are I/O intensive. However, switching between device-power states has associated time and power penalties, i.e., a device takes a certain amount of time and power to transition between its power states. In real-time systems where tasks have associated deadlines, this switching must be performed with great caution to avoid the consequences of tasks missing their deadlines. Current-day practice involves keeping devices in hard real-time systems powered up during the entirety of system operation; the critical nature of I/O devices operating in real-time prohibits the shutting down of devices during run-time in order to avoid the catastrophic consequences of missed task deadlines.

In this paper, we present two online device-scheduling algorithms for hard real-time systems to reduce the energy consumption of the I/O devices in the system. Our algorithms guarantee that no task deadline is missed. These algorithms can be easily implemented along with the task scheduler using a tick-driven scheduling implementation as described in [11]. We present case studies to show that energy savings of over 50% can be realized for real-life task sets. To the best of our knowledge, this is one of the first attempts at minimizing the energy consumption of I/O devices for real-time systems.

The rest of the paper is organized as follows. We first briefly review related prior work on DPM techniques for the CPU and I/O subsystem in Section II. In Section III, we motivate the need for deterministic device scheduling in real-time systems. We also explain in greater detail the problem we attempt to solve and we clarify our assumptions. We then explain the I/O device-scheduling algorithm, which we term LEDES. In Section IV, we present a more general I/O device-scheduling algorithm along with the underlying theory. In Section V, we present our experimental results. Finally, in Section VI, we summarize the paper and conclude with directions for future research.

II. Related Prior Work

The past decade has seen a significant body of research on low-power design methodologies. This research has focused primarily on reducing the power consumption of the CPU and I/O devices. We first review DPM methods for the CPU. In [25], a minimum-energy, offline preemptive task scheduling algorithm is presented. This method identifies *critical intervals* (intervals in which groups of tasks must run at a constant maximum voltage in any optimal schedule) in an iterative fashion. These tasks are then scheduled using the earliest deadline first scheduling policy [14]. An online scheduling algorithm using a preemptive task model is presented in [8]. It also accepts aperiodic tasks whose deadlines are guaranteed to be met. The guarantee is provided by an acceptance test. In [21], an online DVS algorithm based on the well-known rate-monotonic algorithm is presented. This approach uses the fixed-priority implementation model described in [11]. In [19], a near-optimal offline fixed-priority scheduling scheme is presented. This work is extended in [20] to generate optimal solutions for the DVS problem.

Almost all prior work on DPM techniques for I/O devices has focused primarily on scheduling devices in a nonreal-time environment. I/O-centric DPM methods broadly fall into three categories: timeout-based, predictive, and stochastic. Timeout-based DPM schemes shut down I/O devices when they have been idle for a specified threshold interval [7]. The next request generated by a task for a device that has been shut down wakes it up. The device then proceeds to service the request. Predictive schemes are more readily adaptable to changing workloads than timeout-based schemes. Predictive schemes such as the one described in [9] attempt to predict the length of the next idle period based on the past observation of requests. In [18], a device-utilization matrix keeps track of device usage and a processor-utilization matrix keeps track of processor usage of a given task. When the utilization of a device falls below a threshold, the device is put into the sleep state. In [5], devices with multiple sleep states are considered. Here too, the authors use a predictive scheme based on adaptive learning trees to shut down devices. Stochastic methods usually involve modeling device requests through different probabilistic distributions and solving stochastic models (Markov chains and their variants) to obtain device switching times [3], [22]. In [10], a theoretical approach based on the notion of competitive ratio is developed to compare different DPM strategies. The authors also present a probabilistic DPM strategy where the length of the idle period is determined through a known probabilistic distribution.

An important observation to make here is that none of the above I/O DPM methods are viable candidates for use in real-time systems. Due to their inherently probabilistic nature, the applicability of the above methods to real-time systems falls short in one important aspect—real-time temporal guarantees cannot be provided. Shutting down a device at the wrong time can result in a task missing its deadline. The above methods have been shown to provide significant savings in energy for nonreal-time systems. They also differ from their real-time counterparts in that they attempt to minimize task response times rather than attempting to meet task deadlines. Such methods perform efficiently for interactive systems where user waiting time is an important design parameter. In real-time systems, minimizing mean response time of a task does not guarantee that its deadline will be met. It, thus, becomes apparent that new algorithms that operate in a deterministic manner are needed in order to guarantee real-time behavior.

A recent approach for I/O device scheduling for real-time systems relies on the notion of a mode dependency graph (MDG) for multiple processors and I/O devices [13]. An algorithm based on topological sorting is used to generate a set of valid mode combinations. A second algorithm then determines a sequence of modes for each resource such that all timing constraints are met and max-power requirements are satisfied for a given task set. In contrast to [13], we are interested here in minimizing energy instead of satisfying max-power constraints. A schedule generated in [13] is not necessarily an energy-optimal schedule for the task set. Furthermore, the

work in [13] does not distinguish between I/O devices and processors. On the other hand, the model we assume here is that of a set of periodic tasks executing on a single processor.

Minimizing communication power in multiprocessor systems has also been considered [16]. The authors assume a multiprocessor system, with each node having a voltage-scalable processor, and a communication channel. The network interface at each node can transmit and receive at different speeds, with corresponding power levels. The problem addressed in [16] is to identify a communication speed and a processor speed for each node such that the global energy consumption is minimized.

In the next section, we present our problem statement and describe the assumptions we make.

III. Low-Energy I/O Device Scheduling for Two-State I/O Devices

In this section, we present an I/O device-scheduling algorithm termed LEDES for two-state I/O devices. We assume that each device has two power states—a high-power working state and a low-power sleep state. In Section IV, we will extend the problem to that of scheduling I/O devices with more than two power states.

A. Problem Statement

We are given a task set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n tasks. Associated with each task $\tau_i \in \mathcal{T}$ are the following parameters:

- *release* (or arrival) time a_i;
- period p_i;
- deadline d_i;
- worst case execution time c_i;
- *device usage list* L_i, consisting of all the devices used by task τ_i.

The hyperperiod H of the task set is defined as the least common multiple of the periods of all tasks. We assume that the deadline of each task is equal to its period, i.e., $p_i = d_i$.

The system also uses a set $\mathcal{K} = \{k_1, k_2, \ldots, k_p\}$ of p I/O devices. Each device k_i has the following parameters:

- two power states—a low-power sleep state $ps_{l,i}$ and a high-power working state $ps_{h,i}$;
- a transition time from $ps_{l,i}$ to $ps_{h,i}$ represented by $t_{wu,i}$;
- a transition time from $ps_{h,i}$ to $ps_{l,i}$ represented by $t_{sd,i}$;
- power consumed during wake-up $P_{wu,i}$;
- power consumed during shutdown $P_{sd,i}$;
- power consumed in the working state $P_{w,i}$;
- power consumed in the sleep state $P_{s,i}$.

Requests can be processed by the devices only in the working state. All I/O devices used by a task must be powered-up before the task starts execution. Since an *online* device scheduler must be fast and efficient, we assume that device-scheduling decisions are made only at task starts and completions. The timer capabilities that are present in the operating system can potentially be used to switch device-power states at any time instant. However, interrupt processing incurs a service-time penalty which is architecture-dependent. This service-time penalty, and, therefore, its inclusion in our task model and device-scheduling algorithms, requires special handling; forcing all device switching to be performed only at task starts and completions is a simple approach with no significant impact on energy savings.

We assume that the power consumed in the sleep state is less than the power consumed in the working state, i.e., $P_{s,i} < P_{w,i}$. Without loss of generality, we assume that for a given device k_i, $t_{wu,i} = t_{sd,i} = t_{0,i}$ and $P_{wu,i} = P_{sd,i} = P_{0,i}$. The energy consumed by device k_i is $E_i = P_{w,i}t_{w,i} + P_{s,i}t_{s,i} + MP_{0,i}t_{0,i}$, where M is the total number of state transitions for k_i, $t_{w,i}$ is the total time spent by device k_i in the working state, and $t_{s,i}$ is the total time spent in the sleep state.

Incorrectly switching power states can cause increased, rather than decreased, energy consumption for an I/O device. This leads to the concept of *breakeven time*, which is the time interval for which a device in the powered-up state consumes an energy exactly equal to the energy consumed in shutting a device down, leaving it in the sleep state and then waking it up [9]. Fig. 1 illustrates this concept. If any idle time interval for a device is greater than the breakeven time t_{be}, energy is saved by shutting a device down. For idle time periods that are less than the breakeven time, energy is saved by keeping it in the powered-up state.

We further assume that the start time for each *job* is known *a priori*. Several commercial real-time operating systems support tick-driven scheduling and embedded systems which are mission-critical require a scheduling mechanism that is inherently deterministic [15].

We now provide a simple problem statement.

- $\mathcal{P}_{io}$: Given the start times $\mathcal{S} = \{s_1, s_2 \ldots, s_n\}$ for a real-time task set $\mathcal{T}$ that uses a set $\mathcal{K}$ of I/O devices, determine a sequence of sleep/working states for each I/O device $k_i \in \mathcal{K}$ such that the total energy consumed $\sum_{i=1}^{p} E_i$ by the set $\mathcal{K}$ of I/O devices is minimized and all tasks meet their deadlines.

In the following sections, we describe the conditions under which device-state transitions can be made while guaranteeing the timely completion of tasks. These conditions are different for different scenarios, the scenarios being dependent on the execution times of the tasks that comprise the task set and the number of sleep states present in a device. We first start by assuming that all task execution times are greater than the transition time of all devices and all devices have a single sleep state. We then show that when devices have multiple power states, guaranteeing timeliness becomes more complex.

One notable advantage of online I/O device scheduling is that online DPM decision-making can further exploit underlying hardware features such as buffered reads and writes. A device schedule constructed offline and stored as a table in memory precludes the use of such features owing to its timer-driven approach.

In the next section, we motivate the need for deterministic I/O device-scheduling policies with an example. We then introduce some additional concepts that will be used later.

B. Motivation for Deterministic I/O Device Scheduling

Before we describe the LEDES algorithm, we first use a simple example to motivate the need for deterministic I/O device-scheduling methods.

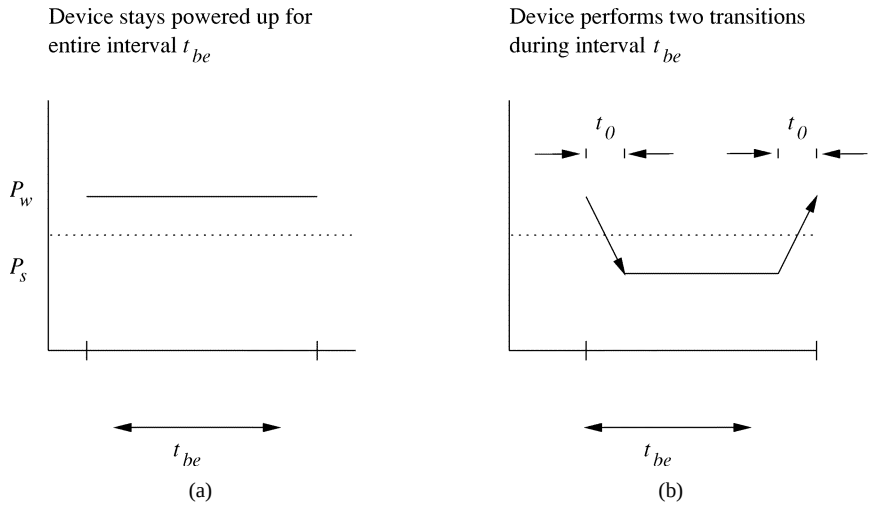

Fig. 1. Illustration of breakeven time. The time interval for which the energy consumptions are the same in (a) and (b) is called the breakeven time.

TABLE I
EXAMPLE TASK SET

Task	Execution time	Period (Deadline)	Device list
τ_1	10	50	k_1
τ_2	20	80	k_2
τ_3	40	100	k_3

Table I describes a simple three-task real-time task set. The task schedule using the rate-monotonic algorithm is shown in Fig. 2.

For the sake of simplicity, but without loss of generality, we have assumed that these tasks use independent devices. We have assumed, also for the sake of simplicity, that the device characteristics are the same for all devices. We assume a working power of 60 units, a transition power of 30 units and a sleep power of 10 units for both devices. Assuming that the transition time is 10 units, the breakeven time for the devices is calculated to be 10 units.

Let us focus on the first 100 time units. At $t = 0$, all devices are powered up. At this point, decisions must be made to power-down certain devices in order to reduce energy consumption. We can easily see that device k_1 cannot be shut down because it is used immediately by task τ_1. Assuming that we have no knowledge of future task requests, we cannot deterministically identify (at this point) whether any of the devices can be shut down and yet be powered up in time for future device requests. Thus, all devices stay powered up at $t = 0$. At $t = 10$, when task τ_1 completes its execution, τ_2 is scheduled to run. At this point, let us assume a timeout-based policy is used to shut down devices, and that the timeout value is 10 units.

Now, devices k_2 and k_3 have not been used for 10 units of time. Therefore, they are both candidates for shutdown. However, k_2 is used by task τ_2 and cannot be powered-down. Since k_3 is not used by τ_2, it is powered down and placed in the sleep state.

When τ_2 completes execution at $t = 30$, task τ_3 is ready to run. However, device k_3 has to be woken up before τ_3 can run. Since the transition time for k_3 is 10 units, τ_3 *waits* for 10 units until k_3 is powered up and then starts its execution, *effectively* at $t = 40$. It is now easy to see that task τ_3 will miss its deadline due to the latency involved in powering up device k_3.

Thus, it is not possible to guarantee timely completion of tasks without *a priori* knowledge of future device requests. A naive, probabilistic algorithm cannot be used for real-time task sets. This introduces the notion of *lookahead*, which is a bound on the number of tasks whose device-usage lists must be examined before making a state transition decision, in order to guarantee that no future task deadline will be missed.

The need for the assumption that scheduling points always lie at task start or completion times also follows from the above example. A direct effect of flexible device scheduling is that it introduces additional context switches for device scheduling. These additional context switches may push the limits of a tightly constrained task set. Rather than introducing this additional overhead, we choose instead to schedule devices only at task starts and completions, where context switches are inevitable. Performing device scheduling in this manner does not create any additional context-switching overhead.

Next, we present the LEDES for online scheduling of I/O devices with two power states.

C. LEDES Algorithm

The LEDES algorithm makes the assumption that the execution times of all tasks are greater than the transition times of the devices they use. Under this assumption, the amount of lookahead required while making wake-up decisions to guarantee timeliness is bounded. We derive this result by proving the following theorem.

Theorem 1: Given a task schedule for a set $\mathcal{T}$ of n tasks with completion times $c_1, c_2, \ldots, c_n$, the device utilization for each task, and an I/O device k_l, it is necessary and sufficient to look ahead as only m tasks to guarantee timeliness, where m is the smallest integer such that $\sum_{i=1}^{m} c_i \geq t_{0,l}$.

Proof: First, we prove necessity. Let us assume that at some scheduling instant s_i, task τ_i is scheduled for execution, and τ_i uses device k_p. Also, suppose task τ_j, $j \neq i$, uses device k_l, $p \neq l$, $s_j - s_i < t_{0,l}$ and $s_j + c_j = d_j$. Further, k_p is powered up and k_l is in the sleep state. We can easily see that

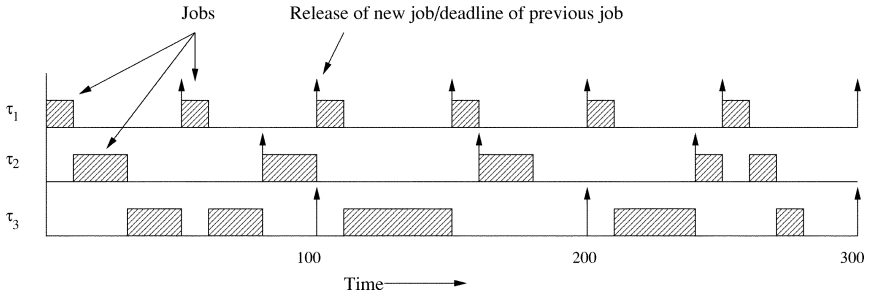

Fig. 2. Rate-monotonic task schedule for task set in Table I.

without lookahead, at s_j, k_l is not powered up and cannot serve requests for τ_j. This means that k_l must be powered up before τ_j can start. This results in $s_j + c_j + t_{0,l} > d_j$. Moreover, if $s_j - s_i < t_{0,l}$, the device will not have enough time to wake up before s_j. Hence, we need to look ahead m tasks such that $\sum_{i=1}^{m} c_i \geq t_{0,l}$.

We prove sufficiency as follows. As long as there is a lookahead of m, we can guarantee that there is at least one valid scheduling instant between the start times of two tasks that require different devices. This leads to the conclusion that with a lookahead of m, there is sufficient time to wake a device up before the start time of a task requesting it. This completes the proof of the theorem. □

In most practical cases, the completion times of all tasks in the taskset are greater than the transition times $t_{0,i}$ of device k_i. This leads to the following corollary to Theorem 1.

Corollary 1: Given a task schedule for a set $\mathcal{T}$ of tasks with completion times $c_1, c_2, \ldots, c_n$, the device utilization for each task, and an I/O device k_j, it is necessary and sufficient to look ahead one task to guarantee timeliness if the completion times of all tasks $\tau_i \in \mathcal{T}$ is greater than the transition time $t_{0,j}$ of device k_j.

The LEDES algorithm in pseudocode form is shown in Fig. 3.

The algorithm operates as follows. At the start time of task τ_i (Line 1), all devices not used by the next "immediate" tasks τ_i and τ_{i+1} are put in the sleep state (Lines 3 and 4). The difference between the start time of τ_{i+1} and the end of τ_i's execution (in the worst case) is checked (Line 7). If this time interval is sufficient to wake device k_j up at τ_i's finish time, then the device k_j is shutdown, since its wake-up can be guaranteed at τ_i's finish time. If device k_j is powered down, then a wake-up decision must be made (Line 11). A device must be woken up at s_i if its wake-up cannot be deferred to τ_i's finish time. This check is performed in Line 12 and the device is woken up if needed.

If the scheduling instant at which LEDES is invoked is the completion time of τ_i (Line 15) and k_j is powered up (Line 17), it can be shut down only if it can fully enter the powered down state before s_{i+1} (there may be a need for it to be woken up again). If k_j is in the sleep state (Line 21) and it is used by τ_{i+1}, it must be woken up to guarantee the timely start of τ_{i+1}. These checks are performed on a per device basis and the entire process continues in a periodic manner. Although there is no mention of the breakeven time in Fig. 3, an implicit check to ensure that the idle period for a given device is always greater than the breakeven time.

```
Procedure LEDES(k_j, τ_i, τ_{i+1})
curr: current scheduling instant;
1.  if scheduling instant is the start of τ_i then
2.    if k_j is powered-up then
3.      if k_j ∉ L_i and L_{i+1} then
4.        shutdown k_j
5.      end if
6.      if k_j ∈ L_{i+1} then
7.        if s_{i+1} − (s_i + c_i) ≥ t_{0,j} then
8.          shutdown k_j
9.        end if
10.     end if
11.   else
12.     if k_j ∈ L_{i+1} and s_{i+1} − (s_i + c_i) < t_{0,j} then
13.       wakeup k_j
14.     end if
15.   else end
16. if scheduling instant is completion of task τ_i then
17.   if k_j is powered-up then
18.     if k_j ∉ L_{i+1} and s_{i+1} − curr ≥ t_{0,j} then
19.     shutdown k_j
20.     end if
21.   else
22.     wakeup k_j
23.   else end.
```

Fig. 3. LEDES algorithm.

A simple extension to LEDES can effectively handle devices that possess both multiple speed states and the ability to switch from any low-power state directly to the working state. Although the transition times from the powered-up state to the sleep states may be different, LEDES can easily identify the correct sleep state to switch a device to by simply performing a series of transition time checks to verify that if the device were to be switched to the chosen sleep state, there is sufficient time to wake it up before the start of the task that uses it next. However, LEDES cannot make full use of the available sleep states for devices which possess multiple sleep states, but do *not* possess the ability to jump to any sleep state from the powered-up state. We next present a more general I/O-centric power management algorithm for hard real-time systems. We refer to the proposed algorithm as the MUSCLES. MUSCLES can also handle devices which do not have the ability to jump from the powered-up state to any sleep state. This is explained in further detail next.

For a given, predetermined task schedule, MUSCLES generates a sequence of device states for each device such that the energy consumed by the I/O subsystem is minimized. Like LEDES, MUSCLES guarantees that no task deadlines

are missed. Each device has $m \geq 1$ power states and the powered-up state. As in LEDES, we assume that device scheduling occurs only at the start and completion of a task.

Devices which possess multiple power states and the ability to jump directly from any low-power state to the powered-up state can be viewed as a device with only two power states. As stated earlier, LEDES can efficiently schedule these types of devices for low-power. Although not explicitly mentioned in the algorithm of Fig. 3, LEDES can identify the deepest possible state to switch a device to without violating any temporal constraints (deadlines) by checking the transition time to each state and ensuring that enough time exists to wake the device up before the start time of the next task that uses it. However, in the case of MUSCLES, we assume that at a device-scheduling instant, a device may be switched from one power state to the next higher- or lower-power state, i.e., only a single transition is possible at any scheduling instant.

In the next section, we describe the MUSCLES algorithm in greater detail.

IV. Low-Energy I/O Device Scheduling for Multistate I/O Devices

In this section, we describe the MUSCLES scheduling algorithm. MUSCLES is a more powerful scheduling algorithm than LEDES because it efficiently schedules devices with multiple power states. It can also handle devices which can switch from a given state to either the next higher or lower power state. In the next subsection, we introduce some additional terminology as well as the underlying theory.

A. Terminology and Theoretical Background

The assumptions regarding the properties of a real-time periodic task remain unchanged from Section III-A. However, I/O device properties now include parameters describing the different power states. These device properties are restated as follows. Each I/O device $k_i \in \mathcal{K}$ is now characterized by:

- a set $\mathcal{PS}_i = \{ps_{i,1}, ps_{i,2}, \ldots, ps_{i,m}\}$ of m sleep states;
- a powered-up state $ps_{i,u}$;
- transition time from $ps_{i,j}$ to $ps_{i,j-1}$, denoted by $t_{wu}^{i,j}$;
- transition time from $ps_{i,j}$ to $ps_{i,j+1}$, denoted by $t_{sd}^{i,j}$;
- power consumed during switching up from state $ps_{i,j}$ to $ps_{i,j-1}$, denoted by $P_{wu}^{i,j}$;
- power consumed during switching down from state $ps_{i,j}$ to $ps_{i,j+1}$, denoted by $P_{sd}^{i,j}$;
- power consumed in the working state P_w^i;
- power consumed in sleep state $ps_{i,j}$, denoted by $P_s^{i,j}$.

We assume, without loss of generality, that for each device $k_i \in \mathcal{K}$, $t_{wu}^{i,j+1} = t_{sd}^{i,j} = t_{0,i}$ and $P_{wu}^{i,j+1} = P_{sd}^{i,j} = P_{0,i}$.

The total energy E_i consumed by device k_i over the entire hyperperiod is given by

$$E_i P_w^i t_w^i + \sum_{j=1}^{m} P_s^{i,j} t_s^{i,j} + M P_{0,i} t_{0,i}$$

where M is the number of state transitions, t_w^i is the total time spent by the device in the working state, and $t_s^{i,j}$ is the total time spent by the device in sleep state $ps_{i,j}$.

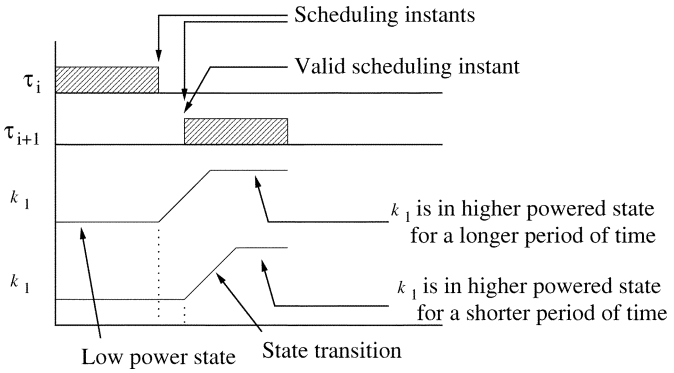

Fig. 4. Example of an invalid scheduling instant.

In order to provide conditions under which devices can be shut down and powered up, we first define a few important terms.

The *intertask time* IT_i for task τ_i is the time interval between the start of task τ_{i+1} and completion of task τ_i, i.e., $\mathrm{IT}_i = s_{i+1} - (s_i + c_i)$. There are two scheduling instants associated with a task τ_i. These correspond to the start and completion time of τ_i, respectively. For minimum-energy device scheduling under real-time constraints, it is not always possible to schedule devices at all scheduling instants. This is formalized using the notion of a valid scheduling instant.

The completion time of τ_i is defined to be a *valid scheduling instant* for device k_j if $s_{i+1} - (s_i + c_i) \geq t_{0,j}$. In other words, the completion time of τ_i is a valid scheduling instant if and only if $\mathrm{IT}_i \geq t_{0,j}$. The start time of τ_i is always a valid scheduling instant. Thus, a task τ_i can have either one or two scheduling instants, depending on the magnitude of IT_i relative to the transition time $t_{0,j}$ of a device k_j. Valid scheduling instants are important for energy minimization. Wake-ups can be scheduled at these points such that energy is minimized while at the same time guaranteeing that real-time requirements are met. Consider the example shown in Fig. 4. This figure shows two tasks τ_i and τ_{i+1} with the intertask time $\mathrm{IT}_i < t_{0,j}$. Assume that device k_1 (first used by task τ_{i+2}) is in state $ps_{1,1}$ at τ_i's completion time $(s_i + c_i)$. If a device were to be woken up at $s_i + c_i$, it would complete its transition to state $ps_{1,0}$ only in the middle of τ_{i+1}'s execution and would be in the higher powered state for the rest of τ_{i+1}'s execution, i.e., until the next scheduling instant. If the device were to be woken up at s_{i+1}, we can still guarantee that the device will be awake before task τ_{i+2} starts (with the assumption that $c_{i+1} > t_{0,1}$). However, the device will be in the lower powered state until s_i, resulting in greater energy savings (note that $s_i + c_i$ is not a valid scheduling instant). Hence, we see that wake-ups at valid scheduling instants always result in greater energy savings. It is always preferable to wake a device up as late as possible in order to further conserve energy.

In Section III-B, we have shown that a lookahead of one task is sufficient for the case where the completion time of every task is greater than the transition time $t_{0,i}$. However, this lookahead is insufficient when devices have multiple sleep states. This is clarified using the example shown in Fig. 5.

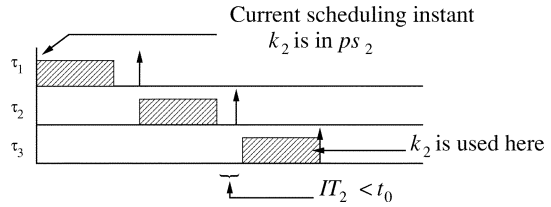

Fig. 5. Lookahead of one task is insufficient.

Fig. 5 shows the execution of three tasks τ_1, τ_2, and τ_3. The start time of τ_1 corresponds to the current scheduling instant. Tasks τ_1 and τ_2 do not use device k_2, which is in sleep state $ps_{2,2}$ at time s_1. An algorithm using a lookahead of one task, i.e., looking ahead only to task τ_2, would erroneously decide that there is no need to wake k_2 up at time s_1. The same situation arises at scheduling instant $s_1 + c_1$. At τ_2's start time (s_2), looking ahead to task τ_3, k_2 is switched to state $ps_{2,1}$. At τ_2's completion time, looking ahead one task to τ_3, k_2 is switched again to the powered-up state $ps_{2,u}$. However, if the intertask time IT_2 is less that $t_{0,2}$, k_2 will not have sufficient time to wake up. This results in τ_3 missing its deadline.

From the above example, we note that lookahead represented as the number of future tasks to be considered is not adequate for the multiple state low-energy scheduling problem. For the case where devices have multiple states, lookahead must be represented as the number of *valid scheduling instants* between tasks. In fact, the notion of lookahead changes slightly when considering multiple-state I/O devices. Lookahead now represents the *number of future tasks that do not use a given device when making scheduling decisions for that device*. Scheduling complexity, thus, increases with increasing lookahead. Hence, minimizing lookahead makes the scheduler more efficient.

We now provide an upper bound on the lookahead necessary to guarantee timeliness while making shut down decisions for a device.

Theorem 2: Consider an ordered set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n tasks that have been scheduled *a priori*. Let $\mathcal{K} = \{k_1, k_2, \ldots, k_p\}$ be the set of p I/O devices used by the tasks in $\mathcal{T}$. In order to decide whether to switch a device $k_i \in \mathcal{K}$ from state $ps_{i,j}$ to $ps_{i,j+1}$ at task τ_c's start or completion time, it is necessary and sufficient to look ahead L tasks, where L is the smallest integer such that the total number of valid scheduling instants associated with the sequence of tasks $\tau_c, \tau_{c+1}, \ldots, \tau_{c+L-1}$ excluding the current scheduling instant is at least equal to $j+1$. The device k_i can be switched down from $ps_{i,j}$ to $ps_{i,j+1}$ if no task τ_t, $c \leq t \leq c+L-1$, uses device k_i.

Proof: We first prove necessity. Suppose that at each valid scheduling instant, device k_i can transition from a low-power state to the next higher powered state. At the $(j+1)$th valid scheduling instant, k_i can move from state $ps_{i,1}$ to $ps_{i,u}$. Let us assume that device k_i is in the $(j+1)$th power state and there exist only q, $q < j+1$, valid scheduling instants before k_i is used again. In this situation, at the qth valid scheduling instant, k_i will switch from state $ps_{i,j+1-q}$ to $ps_{i,j-q}$, which is not the powered-up state. Hence, we see that it is necessary to lookahead $j+1$ scheduling instants from the current scheduling instant to guarantee real-time behavior.

Procedure MUSCLES($\mathcal{S}, \mathcal{PS}, k_i$)
curr: current scheduling instant;
At s_m:
1. Find first task τ_L that uses device k_i;
2. Compute number of valid scheduling instants X between s_m and τ_L;
3. **if** $X \geq j+1$ switch down k_i from $ps_{i,j}$ to $ps_{i,j+1}$;
4. **else if** $X = j$ wake k_i up from $ps_{i,j}$ to $ps_{i,j-1}$;

At $s_m + c_m$:
5. Find first task τ_L that uses device k_i;
6. Compute number of valid scheduling instants X between s_m and τ_L;
7. **if** $X \geq j+1$ switch down k_i from $ps_{i,j}$ to $ps_{i,j+1}$;
8. **else if** $X = j$ and curr is a valid scheduling instant
9. Wake k_i up from $ps_{i,j}$ to $ps_{i,j-1}$;
10. **else** leave k_i in $ps_{i,j}$.

Fig. 6. Pseudocode description of MUSCLES.

TABLE II
EVALUATION TASKSETS FOR LEDES AND MUSCLES

Taskset	Approximate number of jobs	Hyperperiod
T_1	303	1700
T_2	68951	567800
T_3	36591	341700

We prove sufficiency as follows. Suppose we use the amount of lookahead specified by Theorem 2. As long as there is a lookahead of $j+1$ valid scheduling instants, we can guarantee that a device in the $(j+1)$th state has sufficient time to be woken up before the next task that it uses requests the device. This completes the proof of the theorem. □

If the intertask times of all tasks are less than the transition time $t_{0,j}$ for device k_j, Theorem 2 yields the following corollary.

Corollary 2: Suppose the intertask time $\mathrm{IT}_i <$ transition time $t_{0,i}$ for every task $\tau_c \in \mathcal{T}$. In order for a device $k_i \in \mathcal{K}$ to be switched down from state $ps_{i,j}$ to $ps_{i,j+1}$ at the start or completion time of task τ_c, it is necessary and sufficient to look ahead $j+1$ tasks to guarantee timeliness. Moreover, no task τ_t, $i \leq t \leq j$, must use device k_i.

On the other hand, if the intertask times for all tasks is greater than or equal to the transition time $t_{0,j}$, Theorem 2 leads to the following corollary.

Corollary 3: Suppose the intertask time $IT_i \geq$ transition time $t_{0,j}$ for every task $\tau_c \in \mathcal{T}$. In order for a device $k_j \in \mathcal{K}$ to be switched down from state $ps_{i,j}$ to $ps_{i,j+1}$ at the start or completion time of task τ_c, it is necessary and sufficient to lookahead $\lceil (j+1)/2 \rceil$ tasks to guarantee timeliness. Moreover, device k_j must not be used by any task τ_t, $i \leq t \leq j$.

An interesting point to note here is that the amount of lookahead increases as the depth of the sleep-state increases.

We next provide an upper bound on lookahead for making wake-up decisions. The proof is omitted since it is similar to that for Theorem 2.

Theorem 3: Consider an ordered set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n tasks and a set $\mathcal{K} = \{k_1, k_2, \ldots, k_p\}$ of p devices used by the tasks in $\mathcal{T}$. Suppose the first task after τ_c that uses device k_i is τ_{c+L}. The device $k_i \in \mathcal{K}$ must be switched up from state $ps_{i,j+1}$ to $ps_{i,j}$ at the start or completion time of task τ_c if and only if the total number of valid scheduling instants including the current scheduling instant associated with the tasks

TABLE III
DEVICE PARAMETERS USED IN EVALUATING LEDES AND MUSCLES

Device k_i	Device type	P_w	$P_{sd}^{i,0} = P_{wu}^{i,1}$	$P_{sd}^{i,1} = P_{wu}^{i,2}$	$P_{sd}^{i,2} = P_{wu}^{i,3}$	t_0	$P_s^{i,1}$	$P_s^{i,2}$	$P_s^{i,3}$
k_1	HDD [6]	2.3W	1.5W	0.6W	0.3W	0.6s	1.0W	0.5W	0.2W
k_2	NIC [2]	0.3W	0.2W	0.05W	—	0.5s	0.1W	3mW	—
k_3	DSP [24]	0.63W	0.4W	0.1W	—	0.5s	0.25W	0.05W	—

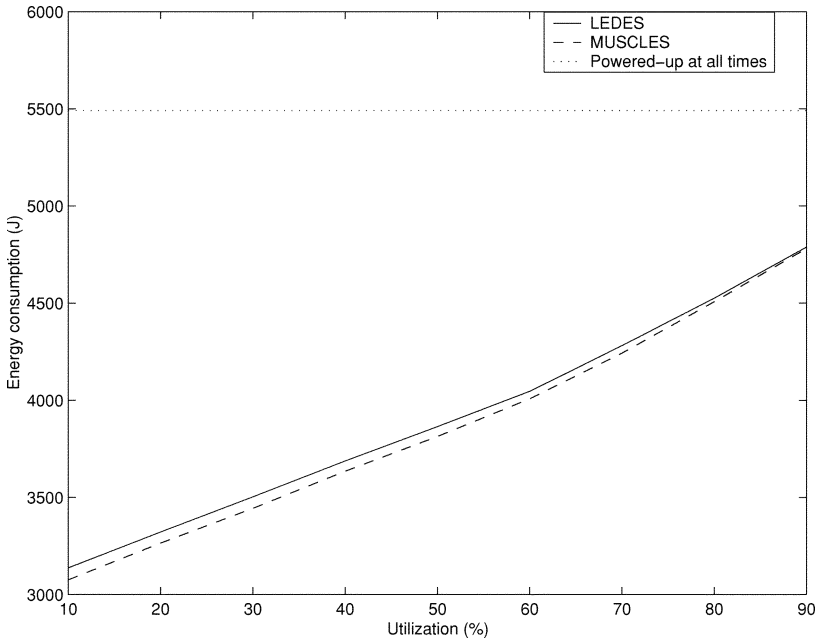

Fig. 7. Comparison of LEDES and MUSCLES for taskset T_1.

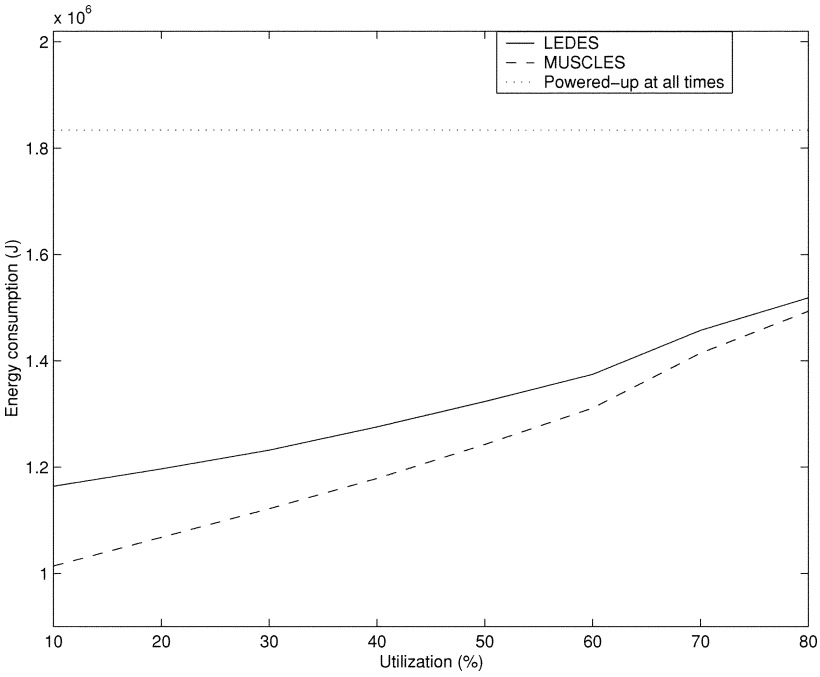

Fig. 8. Comparison of LEDES and MUSCLES for taskset T_2.

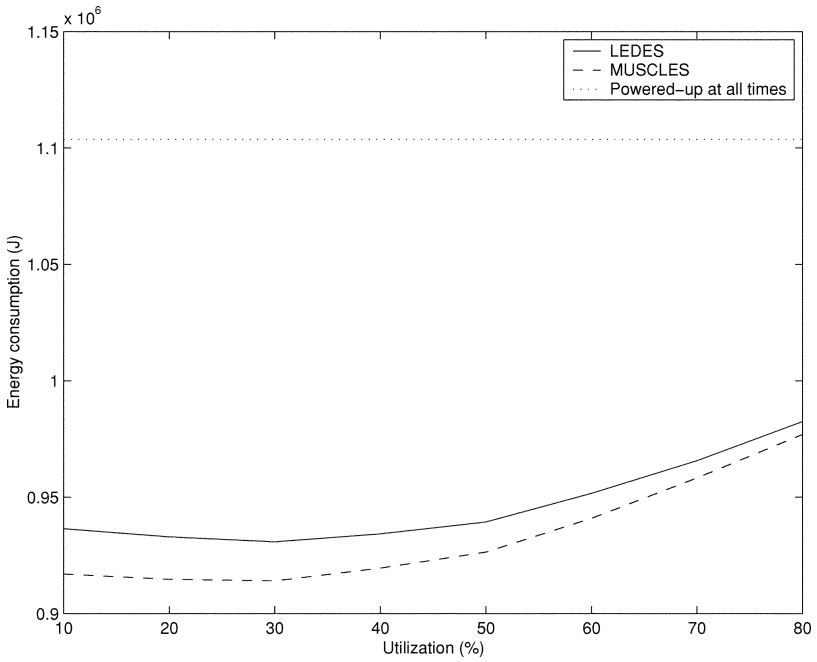

Fig. 9. Comparison of LEDES and MUSCLES for taskset T_3.

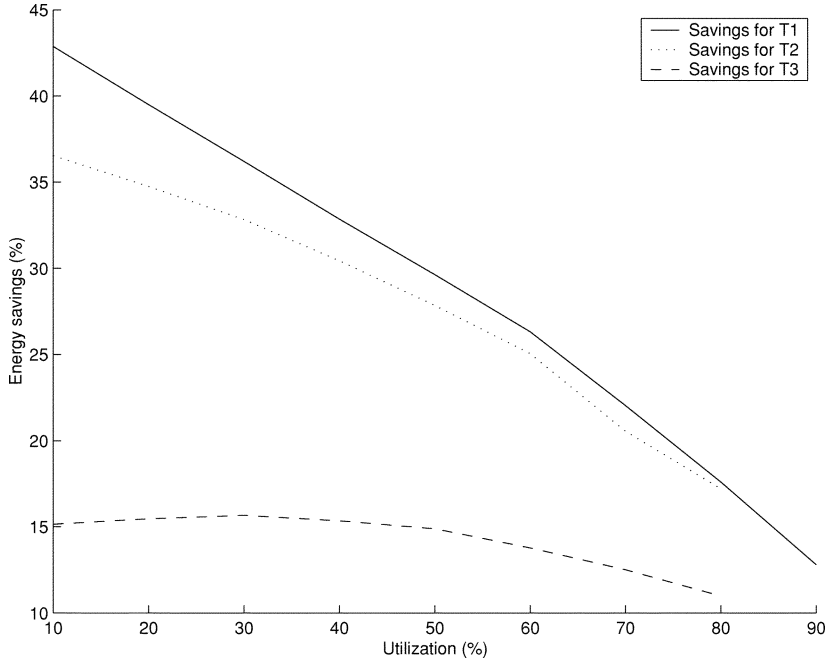

Fig. 10. Energy savings using LEDES.

$\tau_c, \tau_{c+1}, \ldots, \tau_{c+L-1}$ is exactly equal to $j+1$, where L is the lookahead from the current scheduling instant.

We next present and describe the MUSCLES scheduling algorithm.

B. MUSCLES Algorithm

MUSCLES takes as inputs a precomputed task schedule and a per task device-usage list and generates a sequence of power states for every device such that energy is minimized. It also guarantees that the deadlines for every task are met.

Fig. 6 describes the algorithm in pseudocode form. It operates as follows. Let device k_i be in state $ps_{i,j}$ at scheduling instant s_m. MUSCLES first finds the next task τ_L that uses the device (Line 1). Then, a check is performed to determine whether the device can be switched down. This is done by ensuring that there are at least $j+1$ valid scheduling instants between the current scheduling instant (excluding it) and τ_L's start time. If there exist $j+1$ valid scheduling instants, the algorithm switches device k_i from state $ps_{i,j}$ to $ps_{i,j+1}$ (Line 3). If $j+1$ valid scheduling instants do not exist, the algorithm then checks to see if the device must be switched up. If there exist exactly j instants, then the device must be switched up in order to guarantee timeliness (Line 4). At the completion of a task τ_m, the same process is repeated. However, an additional

check is performed to see if the current scheduling instant is a valid scheduling instant. This is done to minimize energy consumption. If the current scheduling instant is not a valid scheduling instant, the device is left in the same state until a valid scheduling instant (Line 10). MUSCLES guarantees that no task ever misses its deadline.

MUSCLES has a worst case complexity of $O(pn)$, where p is the number of I/O devices used in the system and n is the number of tasks that execute. On the other hand, LEDES has a worst case complexity of $O(p)$. This is due to the fact that in LEDES, the lookahead is constant. For devices with multiple power states, the amount of lookahead depends on the current power state. The complexity increases because the lookahead for each device has to be computed before the state of the device is changed. Nevertheless, the relatively low complexity of MUSCLES makes online scheduling for low energy and real-time execution feasible.

In the next section, we show that using LEDES and MUSCLES results in energy savings of over 50% for several test cases.

V. Experimental Results

We first evaluate LEDES and MUSCLES with several randomly generated task sets with varying utilizations. The task sets consist of six tasks with varying hyperperiods and randomly generated device-usage lists. These task sets are shown in Table II. Since jobs may be preempted, we consider each preempted slice of a job as multiple jobs with identical device usage lists. As a result, the number of jobs in each taskset in Table II is referred to as approximate. Each task in the taskset uses one or more out of three I/O devices. The power values for each of these devices, shown in Table III, were obtained from real devices that are currently deployed in embedded systems. Each of these tasksets is scheduled using the rate-monotonic algorithm. The utilization each of these tasksets is varied from 10% to 90%, in an effort to characterize the impact of the slack time present in the hyperperiod on the energy consumption of the I/O devices.

While evaluating LEDES, we assume that the low-power sleep state for a device corresponds to the highest-powered sleep state of the device. The energy consumptions at different utilizations for each of the three tasksets at varying utilizations are shown in Figs. 7–9. Fig. 10 illustrates the savings in energy (represented as a percentage) for each of the tasksets obtained from the LEDES algorithm.

We observe from Figs. 7–9 that the energy consumption using LEDES and MUSCLES increases with increasing utilization. This is because of the fact that devices are kept powered up for longer periods of time within the hyperperiod. The resulting decrease in sleep time causes this increased energy consumption. However, we see that energy savings of over 40% can be obtained for tasksets with low utilization and over 35% for tasksets with high utilization. No job deadlines are missed in a taskset at any utilization value.

Another important observation that can be made from the graphs is that the savings in energy obtained from MUSCLES *over* LEDES decreases with increasing utilization. This is because the number of valid scheduling instants decreases with increasing utilization and, therefore, MUSCLES is unable to place devices in deep sleep states as often for high-utilization tasksets as it can with low-utilization tasksets.

TABLE IV
CNC Taskset

Task	Execution time	Period	Deadline
τ_{smpl}	35	2400	2400
τ_{calv}	40	2400	2400
τ_{dist}	180	4800	4800
τ_{stts}	720	4800	4800
τ_{xref}	165	2400	2400
τ_{yref}	165	2400	2400
τ_{xctrl}	570	9600	4000
τ_{yctrl}	570	7800	4000

TABLE V
INS Taskset

Task	Execution time	Period	Deadline
1	1180	2500	2500
2	4280	40000	40000
3	10280	625000	625000
4	20280	1000000	1000000
5	100280	1000000	1000000
6	25000	1250000	1250000

TABLE VI
GAP Taskset

Task	Execution time	Period	Deadline
1	3000	200000	5000
2	2000	25000	25000
3	5000	25000	25000
4	1000	40000	40000
5	3000	50000	50000
6	5000	50000	50000
7	8000	59000	59000
8	9000	80000	80000
9	2000	80000	80000
10	5000	100000	100000
11	1000	200000	200000
12	3000	200000	200000
13	1000	200000	200000
14	1000	200000	200000
15	3000	200000	200000
16	1000	1000000	1000000
17	1000	1000000	1000000

We also evaluated LEDES and MUSCLES on three real-life tasksets. These tasksets are a computer numerical control (CNC) taskset [12], an instrument navigation system (INS) taskset [11], and a generic aviation platform (GAP) taskset [17], respectively. These tasksets are reproduced in Tables IV–VI. We assume that these tasksets pseudorandomly use some of the devices described in Table III. For example, task 2 in the GAP taskset is a communication task that uses the network interface controller. Task 7 is a status update task that performs occasional reads and writes and, therefore, uses a hard disk. The device usages for the tasksets have been inferred from their functionality. Table VII presents the energy consumptions for these tasksets using LEDES and MUSCLES. The energy values here are expressed in units of joules, and they correspond to the energy consumption of the I/O devices over the duration of a

TABLE VII
ENERGY SAVINGS FOR REAL-LIFE TASKSETS

Taskset	Energy (J) All powered up	Energy (J) LEDES	Energy (J) MUSCLES	Percentage savings (LEDES)	Percentage savings (MUSCLES)
CNC	403104	197140	117604	51%	70%
INS	16.5×10^6	7.7×10^6	3×10^6	51%	81%
GAP	381×10^6	210×10^6	153×10^6	45%	60%

TABLE VIII
COMPARISON WITH OPTIMAL AND TIMEOUT-BASED SCHEDULING POLICIES

LEDES	MUSCLES	OPTIMAL	TIMEOUT
120.64J	121.59J	87.13J	136.47J

single hyperperiod. Using LEDES, we obtain an energy savings of 45% for the GAP taskset. With MUSCLES, an energy savings of 80% is obtained for the INS taskset. We see that owing to the low utilizations of real-life tasksets, significant energy savings can be obtained by intelligently performing state transitions for the devices.

We finally compare the energy consumptions of LEDES and MUSCLES with an optimal scheduling algorithm from [23] and a simple timeout-based shutdown scheme. These results are presented in Table VIII. We assume a timeout value of two units, i.e., if any device is not used for two units of time, it is a candidate for shutdown. Furthermore, in order to maintain a consistent basis for comparison, we assume that device transitions in the timeout-based scheme are performed only at task starts and completions. This is not the case for the optimal scheduling algorithm, where device transitions are completely flexible. For the sake of simplicity, we use a simple two-task task set (which results in 17 jobs) with a hyperperiod of 45 units. The device-usage lists were assigned arbitrarily.

VI. CONCLUSION

Energy consumption is an important design parameter for embedded computing systems that operate under stringent battery lifetime constraints. In many embedded systems, the I/O subsystem is a viable candidate to target for energy reduction. In this paper, we have described in detail two low-energy I/O device-scheduling algorithms. The first algorithm, called LEDES, assumes that the devices present in the system possess two power states—a high-powered working state and a low-powered sleep state. Even under this somewhat restrictive assumption, our experimental results show that energy savings of over 40% can be obtained. LEDES can also handle devices with the ability to switch from the working state to any of several sleep states, if this feature is supported by the device. We have also presented a more general device-scheduling method called MUSCLES that handles devices with more than two power states. MUSCLES extends LEDES by handling devices with multiple power states that can switch from a given state to only either the next higher- or lower-powered state. We show that both these algorithms can be implemented in an efficient manner by deriving upper bounds on the amount of lookahead required to guarantee that no real-time task deadline will be missed due to device wake-up latencies. We have also presented case studies for real-life tasksets to show that energy savings of over 50% can be obtained by targeting the I/O subsystem for power-reduction techniques. We observe that the amount of energy that can be saved decreases with increasing taskset utilization. However, even for high-utilization tasksets, we obtain savings of over 40% with our device-scheduling algorithms.

We are currently developing algorithms that generate provably optimal device schedules for given tasksets. The task schedules that are generated using these techniques can then be combined with existing online dynamic voltage scaling algorithms to further reduce energy consumption. In this way, two major consumers of energy in embedded systems—the CPU and I/O subsystem—can be efficiently targeted for energy reduction.

REFERENCES

[1] Advanced Configuration and Power Interface (ACPI) Available: *http://www.teleport.com/~acpi* [Online]

[2] AMD Am79C874 NetPHY-1LP Low-Power 10/100 Tx/Rx Ethernet Transceiver Tech. Datasheet.

[3] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 813–833, June 1999.

[4] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Norwell, MA: Kluwer, 1997.

[5] E.-Y. Chung, L. Benini, and G. De Micheli, "Dynamic power management using adaptive learning tree," in *Proc. Int. Conf. Computer-Aided Design*, 1999, pp. 274–279.

[6] Fujitsu MHL2300AT Hard Disk Drive Available: *http://www.fujitsu.jp/hypertext/hdd/drive/overseas/mhl2xxx/mhl2xxx.html* [Online]

[7] R. Golding, P. Bosh, C. Staelin, T. Sullivan, and J. Wilkes, "Idleness is not sloth," in *Proc. Usenix Tech. Conf. UNIX Adv. Comput. Syst.*, 1995, pp. 201–212.

[8] I. Hong, M. Potkonjak, and M. B. Srivastava, "On-line scheduling of hard real-time tasks on variable-voltage processor," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 653–656.

[9] C. Hwang and A. C.-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. Int. Conf. Computer-Aided Design*, 1997, pp. 28–32.

[10] S. Irani, S. Shukla, and R. Gupta, "Competitive analysis of dynamic power management strategies for systems with multiple power states," in *Proc. Design Automation Test Eur. Conf.*, 2002, pp. 117–123.

[11] D. Katcher, H. Arakawa, and J. Strosnider, "Engineering and analysis of fixed priority schedulers," *IEEE Trans. Software Eng.*, vol. 19, pp. 920–934, Sept. 1993.

[12] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin, "Visual assessment of a real-time system design: case study on a CNC controller," in *Proc. Real-Time Syst. Symp.*, 1996, pp. 300–310.

[13] D. Li, P. Chou, and N. Bagerzadeh, "Mode selection and mode-dependency modeling for power-aware embedded systems," in *Proc. Asia South Pacific Design Automation Conf.*, 2002, pp. 697–704.

[14] C. L. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[15] J. W. S. Liu, *Real-Time Systems*. Englewood Cliffs, NJ: Prentice-Hall, 2000.

[16] J. Liu, P. H. Chou, and N. Bagerzadeh, "Communication speed selection for embedded systems with networked voltage-scalable processors," in *Proc. Int. Symp. Hardware/Software Codesign*, 2002, pp. 169–174.

[17] D. C. Locke, D. Vogel, and T. Mesler, "Building a predictable avionics platform in Ada: a case study," in *Proc. Real-Time Syst. Symp.*, 1991, pp. 181–189.

[18] Y.-H. Lu, L. Benini, and G. De Micheli, "Operating system directed power reduction," in *Proc. Int. Conf. Low-Power Electron. Design*, 2000, pp. 37–42.
[19] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proc. Design Automation Conf.*, 2001, pp. 828–833.
[20] ——, "Minimum-energy fixed-priority scheduling for variable-voltage processor," in *Proc. Design Automation Test Eur. Conf.*, 2002, pp. 782–788.
[21] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proc. Design Automation Conf.*, 1999, pp. 134–139.
[22] T. Simunic, L. Benini, P. Glynn, and G. De Micheli, "Event driven power management," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 840–857, July 2001.
[23] V. Swaminathan and K. Chakrabarty, "Pruning-based, energy-optimal device scheduling for hard real-time systems," in *Proc. Int. Symp. Hardware/Software Codesign*, 2002, pp. 175–180.
[24] TMS320C6411 Power Consumption Summary Available: *www-s.ti.com/sc/techlit/spra373* [Online]
[25] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. IEEE Annu. Foundations Comput. Sci.*, 1995, pp. 374–382.

Vishnu Swaminathan received the B.E. degree in computer science and engineering from the University of Madras, Chennai, India, in 1996, and the M.S. and Ph.D. degrees in electrical and computer engineering from Duke University, Durham, NC, in 1999 and 2003, respectively.

He is currently affiliated with Duke University as a Researcher. His research interests are in low-power system design, dynamic power management, and real-time operating systems.

Krishnendu Chakrabarty (S'91–M'95–SM'00) received the B.Tech. degree from the Indian Institute of Technology, Kharagpur, India, in 1990, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1992 and 1995, respectively, all in computer science and engineering.

He is currently an Associate Professor of Electrical and Computer Engineering at Duke University, Durham, NC. From 2000 to 2002, he was also a Mercator Visiting Professor at the University of Potsdam, Potsdam, Germany. His current research projects (supported by the National Science Foundation, and the Defense Advanced Research Projects Agency, the Office of Naval Research, the Army Research Office, and industrial sponsors) are in system-on-a-chip test, embedded real-time operating systems, distributed sensor networks and modeling, simulation, and optimization of microelectrofluidic systems. He is a coauthor of two books: *Microelectrofluidic Systems: Modeling and Simulation* (Boca Raton, FL: CRC Press, 2002) and *Test Resource Partitioning for System-on-a-Chip* (Kluwer, 2002), and an editor of the book *SOC (System-on-a-Chip) Testing for Plug and Play Test Automation* (Norwell, MA: Kluwer, 2002). He has published over 120 papers in archival journals and refereed conference proceedings, and he holds a US patent in integrated circuit testing.

Dr. Chakrabarty is a member of ACM and the ACM Special Interest Group on Design Automation, and a member of Sigma Xi. He is a recipient of the National Science Foundation Early Faculty (CAREER) Award, the Office of Naval Research Young Investigator Award and a best paper award at the 2001 Design, Automation, and Test in Europe Conference. He is an Associate Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: ANALOG AND DIGITAL SIGNAL PROCESSING, and an Editor of the *Journal of Electronic Testing: Theory and Applications (JETTA)*. He served as the Guest Editor of a special issue of *JETTA* on system-on-a-chip testing, published in August 2002. He was also a Guest Editor of a special issue of the *Journal of the Franklin Institute* in 2001 on distributed sensor networks in 2001. He serves as Vice Chair of Technical Activities of IEEE's Test Technology Technical Council and is a Member of the Program Committees of several IEEE/ACM conferences and workshops.