

# Mode Selection and Mode-Dependency Modeling for Power-Aware Embedded Systems

Dexin Li, Pai H. Chou, and Nader Bagherzadeh  
Dept. of ECE, University of California, Irvine, CA 92697-2625 USA  
{dli,chou,nader}@ece.uci.edu

## Abstract

*Among the techniques for system-level power management, it is not currently possible to guarantee timing constraints and have a comprehensive system model supporting multiple components at the same time. We propose a new method for modeling and selecting the power modes for the optimal system-power management of embedded systems under timing and power constraints. First, we not only model the modes and the transitions overhead at the component level, but we also capture the application-imposed relationships among the components by introducing a mode dependency graph at the system level. Second, we propose a mode selection technique, which determines when and how to change mode in these components such that the whole system can meet all power and timing constraints. Our constraint-driven approach is a critical feature for exploring power/performance tradeoffs in power-aware embedded systems. We demonstrate the application of our techniques to a low-power sensor and an autonomous rover example.*

## 1 Introduction

Recent trends in mobile and autonomous embedded systems are giving rise to a new class of *power-aware* systems. Unlike low-power systems, whose goal is to minimize power usage, power-aware systems are more general in that they must make the best use of the available power by adapting their behavior to the constraints imposed by the environment, user requests, or their power sources. Power-aware systems must use components that are capable of multiple modes of operation. Many of these components offer modes for power management, while other components allow the user to control the voltage or frequency as other forms of power modes. The selection of mode is thus the primary means of controlling power usage, and it is often done in conjunction with scheduling.

New off-the-shelf components are offering increasingly sophisticated modes for power management. However, the

system-level power manager has only limited control over the modes. Some modes can be set by writing commands to a control register of a device. However, the power manager may not be able to arbitrarily select the modes it wishes at all times. It may be forced to wait or request a change through a sequence of intermediate modes. Even if a desired mode is available, changing mode can incur nontrivial overhead both in terms of time and power. The overhead translates into penalty in performance or power, and it can cause a system to miss an important deadline.

Another key issue for power management is that mode selection cannot be done in isolation. The choice of mode in one component must be coordinated with that in other components, or else the whole system may not function correctly. For example, if the mode selection involves a particular encoding scheme, then the rest of the system that depends on the data representation must also change mode in order to handle the encoding correctly.

It can be difficult for designer to track details with modes. The problem is further exacerbated by the fact that the number of components and the available modes are increasing rapidly. Today's methodologies either limit the complexity by using only a small subset of the available modes (e.g., on, sleep, off), or they are unable to guarantee timing or power constraints.

Power management of embedded systems must consider all components in the system. Significant power reduction in one components may not translate into desirable power reduction for the whole system. In mission critical applications, peripheral devices including mechanical and thermal devices can actually dominate power consumption and must be an integral part of power management.

We believe that a new methodology for mode modeling and selection is sorely needed in order to effectively manage the power of the next generation embedded systems. This paper first introduces a new *mode dependency graph* for modeling the *enabling* relationships among modes within a component and between components in a system. Second, this paper presents a new mode selection algorithm that produces a mode schedule that satisfies timing and power con-

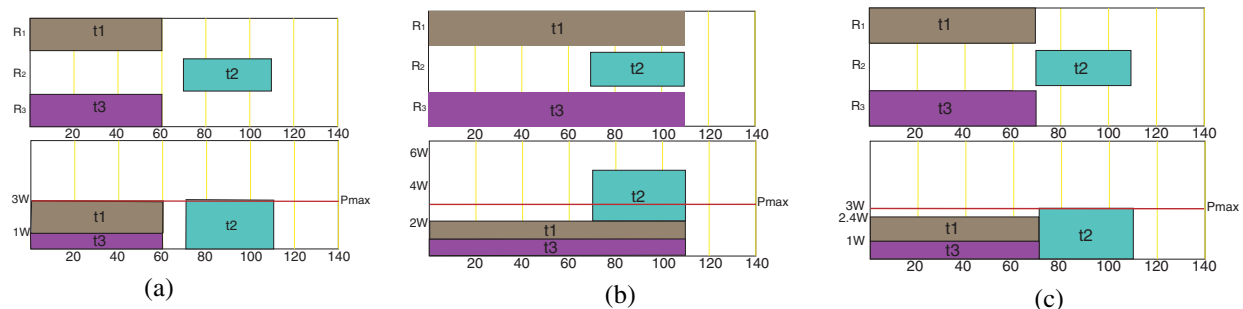


Figure 1. An application scenario that has resource dependency.

straints on multiple processors and devices. It takes advantage of the mode dependency graph in effectively pruning the search space, making it practical to incorporate into an on-line power manager. The advantage with our constraint-driven approach is that it is not hardwired to a specific objective such as power minimization. This is a crucial feature for power-aware embedded systems, for which the ability to make power/performance tradeoffs is more important than just power reduction.

This paper is organized as follows. Section 2 reviews related work. Section 3 presents the mode dependency graph, while Section 4 describes a mode selection algorithm that takes advantage of mode dependency modeling. We discuss the experimental results in Section 5.

## 2 Related Work

Many low-power techniques have been developed at all levels for system-level designs, since the components are largely off-the-shelf or already designed, the applicable techniques include dynamic voltage scaling (DVS) and dynamic power management (DPM).

### 2.1 Dynamic Voltage Scaling (DVS)

Developed for variable-voltage processors, DVS can achieve significant energy saving while still enabling the processor to continue making progress [12, 2]. Although DVS means running slower, they typically slow down just enough without violating timing constraints, and many are based on real-time task scheduling cores [2, 8, 9, 7].

It has been shown that maximal energy saving is achieved by running the processor at the slowest possible constant speed, rather than running tasks at full processor speed and changing the processor to a lower power mode when idle. Hong et al [2] proposed a heuristic for scheduling real-time tasks on a variable voltage processor. Shin [8] exploited both execution time variation and idle time intervals for fix-priority tasks. Shin's algorithm in [9] determines the lowest maximum processor speed for each job to

achieve power reduction. Quan and Hu [7] further greedily determine the lowest voltage for a set of tasks to achieve more energy savings.

What these DVS techniques have in common is that they are greedy and assume a single processor. A power-aware embedded system, however, consists of multiple resources, which may be one or more processors and peripheral devices. Unfortunately, greedy DVS techniques are not generalizable to multiple resources under power constraints, as shown in the following example.

#### Example: (DVS fails in multi-resource)

Fig. 1(a) shows a Gantt chart (top) and the power profile (bottom) for a system with three resources:  $R_1$  is capable of voltage scaling, while  $R_2$  and  $R_3$  are not. The task  $t_1$  on  $R_1$  has a deadline at 110. The system has a max power constraint of  $3W$ . Furthermore, the behavior of the application dictates that  $R_1$  and  $R_3$  be *co-active*. Co-activation means the execution of one task requires the power consumption of other dependent services or tasks. A simple example is that when the CPU is running, it imposes a co-activation dependency on the memory, but co-activation can be much more general between sets of tasks.

Fig. 1(b) shows the schedule and the power profile obtained by greedily slowing down  $R_1$ . Even though all timing constraints are satisfied, it violates power constraints and it is not minimum energy. When it is stretched out,  $t_1$  overlaps  $t_2$  during time 70-110, and their total power exceeds the max power constraint. It is not minimum energy due to the co-activation dependency between  $R_1$  and  $R_3$ : the energy saving by  $R_1$  due to voltage scaling is more than offset by  $R_3$ , whose execution is prolonged by  $R_1$ .

The optimal schedule and power profile are shown in Fig. 1(c). Resource  $R_1$  is slowed down without overlapping  $t_2$  on  $R_2$ . No max power is violated. Although  $t_3$  is stretched with  $t_1$  and therefore consumes more energy than in Fig. 1(a),  $t_1$  saves even more energy due to voltage scaling of resource  $R_1$ . As a result, the system achieves minimal energy while satisfying all constraints. Fig. 2 summarizes the energy costs.

| Schedule  | Timing violation | Power violation | Energy cost |
|-----------|------------------|-----------------|-------------|
| Fig. 1(a) | No               | No              | 300         |
| Fig. 1(b) | No               | Yes             | 320         |
| Fig. 1(c) | No               | No              | 288         |

**Figure 2. Comparison of three schedules.**

Another problem not highlighted with this example is that mode changes may incur nontrivial power or timing overhead. If so, overhead must be considered in determining the feasibility of the mode schedule.

Luo and Jha [5] presents scheduling for multiple processing elements by reordering tasks and applying voltage scaling in this post-processing step after scheduling. Our approach is similar in that it can also be a post processing step, and handles precedence and timing constraints, but we treat power as a hard constraint. Furthermore, we handle co-activation and other mode-dependency relationships.

## 2.2 Dynamic power management (DPM)

Previous work on DPM mainly aimed to achieve power reduction by predicting the system idle time or event distribution and shutting down resources when idle. The simplest power management policy is time-out based on a fixed or predicted amount of time before the system's shutdown or power-up [3]. Stochastic model [1] is used to address the uncertainty in system behaviors. DPM techniques can be effective for minimizing energy and time penalties on average, but they have several limitations. First, most treat either power or timing as an *objective* or penalty, rather than a *constraint*. In real systems, the max power is a real, hard constraint, whose violation can lead to malfunction. Second, they have not considered inter-component dependency in a system, with the exception of Qiu, Qu and Pedram in [6], which models multiple service providers and their Generalized Stochastic Petri Net (GSPN) model can capture some dependencies among resources. However, their model is mainly for the request/dispatch behavior of servers rather than dependency among the servers themselves.

Our new approach, mode selection, combines the advantages of existing approaches. It is entirely constraint driven, enabling us to make power/performance tradeoffs without hardwiring any specific goal or policy in the algorithm.

## 3 Modeling Resource Dependency

Selecting (or not selecting) a mode of a resource may impact the modes that other resources are allowed to select. The impact may be co-activation, exclusion, enabling, and many other possible types of dependency. These dependencies may be extracted from application level specifica-

tions or policies for safety, security, fault-tolerant, power-saving. In any case, a *legal* mode combination of the resources is one that respects all of these dependencies, and a *feasible* mode combination is one that is legal and satisfies all the constraints (namely timing and power). We use a data structure called the mode dependency graph (MDG) that enables efficient generation of legal mode combinations in an order that facilitates the search for feasible combinations that are also low cost.

### 3.1 Definitions

**Definition 1 (Resource  $\gamma \in \Gamma$ )** A resource  $\gamma$  is defined as a graph  $R_\gamma(M_\gamma, H_\gamma)$ , where  $M_\gamma$  is a set of vertices, and  $H_\gamma \subseteq M_\gamma \times M_\gamma$  is a set of edges. A vertex  $m \in M_\gamma$  is a power mode of resource  $\gamma$ . An edge  $(m, n) \in H_\gamma$  represents a mode change from mode  $m$  to mode  $n$ . We define the timing and energy function for a mode change as:  $F : M_\gamma \times M_\gamma \rightarrow T \times E$ , where  $M_\gamma$  is the set of modes of resource  $\gamma$ , and  $T, E$  are time and energy, respectively. The average power can be obtained from energy and time information.

**Definition 2 (Power and delay functions)** Power consumption of a resource  $\gamma$  is represented as a function,  $\pi$ , mapping from power mode to a power number. Formally,  $\pi : M_\gamma \rightarrow \mathbb{R}^+$ . Delay of a mode transition is defined as a function,  $\delta$ , mapping from start mode and end mode of a transition to a delay number. Formally,  $\delta : M_\gamma \times M_\gamma \rightarrow \mathbb{R}^+$ .

**Definition 3 (Mode combination  $\lambda \in \Lambda$ )** Given  $N$  resources  $(\gamma_1, \gamma_2, \dots, \gamma_N)$ , a mode combination is  $\lambda \in M_{\gamma_1} \times M_{\gamma_2} \times \dots \times M_{\gamma_N}$ .

### 3.2 Mode Dependency Graph

A mode dependency graph (MDG)  $G(M, D)$  characterizes the inter-resource dependency relationships, where  $M = \bigcup_{\gamma \in \Gamma} M_\gamma$  is a set of vertices representing power modes, and  $D$  is a set of edges standing for dependencies. A vertex is represented by a circle with a label in the format of " $\gamma.m$ ," where  $\gamma \in \Gamma$  is a resource and  $m \in M$  is a mode of the resource. If two vertices have the same labels, we considered them identical.

The *value* of a vertex  $v \in M$  is defined as:

$$|v| = \begin{cases} True & \text{if } \gamma \text{ is in mode } m, \\ False & \text{if } \gamma \text{ is in other mode,} \\ Undetermined & \text{if } \gamma \text{ has not been selected a mode.} \end{cases} \quad (1)$$

An edge in the MDG represents dependency between two modes. Suppose an edge  $(u, v) \in E$ ,  $u = \gamma_1.m_1$ ,  $v = \gamma_2.m_2$ . The two modes  $m_1$  and  $m_2$  satisfy the mode dependency graph if:

| $u$          | $v$          | Violation |
|--------------|--------------|-----------|
| <i>True</i>  | <i>False</i> | YES       |
| <i>True</i>  | <i>True</i>  | NO        |
| <i>False</i> | <i>True</i>  | NO        |
| <i>False</i> | <i>False</i> | NO        |

**Figure 3. A table for violation checking.**

```

Check_MDG(mode dependency graph  $G$ , resource  $\gamma$ , mode  $m$ ):
1  find all vertices  $v \in V$  that contain resource  $\gamma$ 
2  for each  $v \in V$  {
3      find vertex  $u$  that  $u$  points to  $v$ , if any
4      if violation checking for  $(u, v)$  according to Fig. 3 is YES {
5          return False
6      }
7  }
8  return True

```

**Figure 4. Check satisfaction of a MDG.**

$$\begin{aligned}
|u| \text{ is } True \text{ only if } |v| \text{ is } True \text{ } ( \dashrightarrow ), \\
|v| \text{ is } False \text{ implies } |u| \text{ is } False \text{ } ( \Leftarrow ).
\end{aligned} \tag{2}$$

In other words, if  $|u|$  is *True*,  $|v|$  MAY be *True*; but if  $|v|$  is *False*,  $|u|$  MUST be *False*. For example, we represent the dependency between a CPU and a memory chip such that the memory is on only if the CPU is in active mode. If the CPU is not in active mode, then the memory must not be on. If both of the above conditions are met, we say that the CPU and the memory satisfy the mode dependency. Otherwise, they violate the mode dependency. Fig. 3 summarizes the conditions that (do not) violate the mode dependency.

To expand the capability of mode dependency graph, we introduce the logic operators as another kind of vertices. An operator vertex is represented by a square with an operator label in it. For the operator vertex with multiple outgoing edges, the  $\dashrightarrow$  direction combines disjunctively, and the  $\Leftarrow$  direction combines conjunctively. For example, a vertex  $u$ , whose value  $|u|$  is *True*, points two vertices  $v_1$  and  $v_2$ . If either  $v_1$  or  $v_2$ , or both, are *True*, then they satisfy mode dependency. When  $v_1$  and  $v_2$  are both false, they violate mode dependency. The value of an operator vertex can be obtained by evaluating the logic function it represents. We define the operators *AND*, *OR* and *XOR*. The functions of the operators follow the normal boolean functions in the same names except when any input is “undetermined,” the output is “undetermined.” Given an MDG, a resource  $\gamma$  and one of its mode  $m$ , we can use the routine in Fig. 4 to check whether mode  $m$  satisfies the MDG.

### 3.3 Generating Mode Combinations

This section shows how to efficiently generate legal mode combinations using the MDG.

We transform and reduce an MDG to a resource list. The purpose is to sequence the resources so that the modes of a resource do not depend on those of the succeeding resources. From the MDG, we shrink each operator vertex to a point, and remove mode name in each mode vertex. We then remove the redundant vertices and edges, break the cycle by removing one edge in the cycle, and apply topological sort to obtain a resource list.

If the MDG is acyclic, then legal mode combinations can be generated by a special version of topological traversal. Starting from the first resource of the list, we check modes of each resource against the MDG and identify the legal modes. We keep them and select one for the current resource  $\gamma$ , and move to the next resource. We are able to determine a mode of  $\gamma$  because upon checking the resource, all the modes of its dependent resources have been already determined since they are all located before  $\gamma$ . We progressively generate a mode combination as we check legality of modes and select one at each resource. As we reach the end of the list, we obtain a legal mode combination. We enumerate the rest of legal modes at the end resource, backtrack to previous resources, and enumerate their legal modes to generate other legal mode combinations.

Note that there may be cycles in an MDG, which implies that in the resource list obtained above, modes of a resource may depend not only on preceding resources, but also on succeeding resources. We call such resources *dirty-resource*. In this scenario, we keep track of which resources the current resource  $\gamma$  are dependent on. When the modes of all dependent resources are determined, we evaluate a mode of  $\gamma$  to determine whether the mode satisfies the MDG. Fig. 6 shows the detailed algorithm, which is the general case for both acyclic and cyclic MDGs.

### 3.4 Example: Microsensor

A microsensor system is a node in a distributed microsensor network [10]. It consists of a sensor, a processor, memory chips, radio frequency module and other auxiliary parts. The microsensor obtains information from environment and sends processed data to a base station. The sensor and the memory each has two modes, on and off. The processor has three modes, active, idle and sleep. The radio has three modes, transmit-and-receive (tx\_rx), receive-only (rx), and off. There are a total of 36 mode combinations for these components.

The behavior and dependencies of the devices in this system can be derived from high-level power management policies: the sensor and the radio may be both off only if the

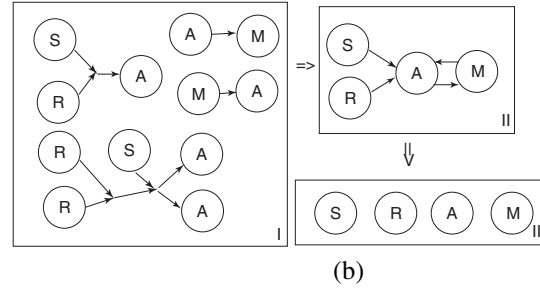
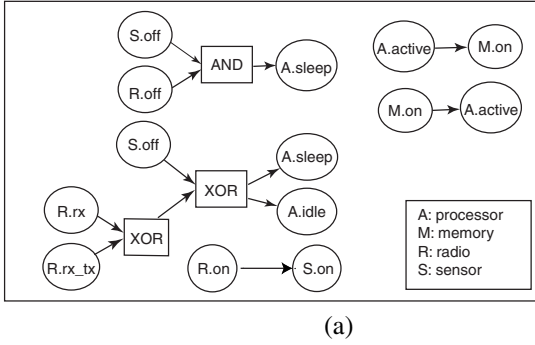


Figure 5. (a) A MDG example: microsensor. (b) Reduce the MDG to a resource list.

ModeGen\_From\_Cyclic\_MDG(mode dependency graph  $G$ ):

```

1  reset the list of mode combinations  $\Lambda \leftarrow \emptyset$ 
2  reset a mode combination  $\lambda \leftarrow \emptyset$ 
3  transform  $G$  into a resource list  $L[0 \dots N-1]$ 
4  mark out dirty-resources in  $L$ 
5   $p \leftarrow 0$ 
6  while  $p \geq 0$  {
7    while  $0 \leq p < N$  {
8      if  $L[p]$  is not a dirty-resource {
9        if found an unmarked mode  $m$  for task  $L[p]$  {
10         if check_MDG( $G, \gamma, m$ ) = TRUE {
11           if  $L[p].cached$  is not empty {
12             if all cached resources satisfy  $G$  is TRUE {
13                $\lambda[p] \leftarrow m, p \leftarrow p+1$  }
14             } else {  $\lambda[p] \leftarrow m, p \leftarrow p+1$  }
15           }
16         mark the mode  $m$ 
17       } else { unmark all modes of  $L[p], p \leftarrow p-1$  }
18     } else {
19       locate the last resource  $L[q]$  that  $L[p]$  is dependent on
20        $L[q].cached \leftarrow L[p]$ 
21     }
22      $p \leftarrow p+1$ 
23   }
24   if  $p = \text{length}(L)$  { push  $\lambda$  into  $\Lambda$  }
25    $p \leftarrow p-1$ 
26   if found an unmarked mode  $m$  for the resource  $L[p]$  {
27     unmark all modes for current resource
28      $p \leftarrow p-1$ 
29   } else {
30     if check_MDG( $m, G$ ) = TRUE {
31        $\lambda[p] \leftarrow m, \text{push } \lambda \text{ into } \Lambda, p \leftarrow p+1$  }
32     }
33   }
34   return  $\Lambda$ 

```

Figure 6. Generate mode combinations for cyclic MDG.

| mode | S   | R     | A      | M   |
|------|-----|-------|--------|-----|
| M1   | on  | tx_rx | active | on  |
| M2   | on  | rx    | idle   | off |
| M3   | on  | rx    | sleep  | off |
| M4   | on  | off   | sleep  | off |
| *M5  | off | tx_tx | active | on  |
| *M6  | off | rx    | idle   | off |
| *M7  | off | rx    | sleep  | off |
| M8   | off | off   | sleep  | off |

Figure 7. Mode combinations of microsensor.

processor is in sleep mode; either of them may be on only if the processor is in sleep mode or idle mode; both the sensor and the radio may be on only if the processor is in active mode; the memory is on if and only if the processor is active. Fig. 5(a) shows the MDG of the microsensor.

Using the MDG, our algorithm automatically generate eight mode combinations that satisfy the given MDG (see Fig. 7). Suppose we want the microsensor to work in a proactive way: when it is off, the system can only be waken up by the sensor when it senses information from environment. The radio cannot wake up the system, for example, by receiving a remote command. We add another item “the radio may be on only if the sensor is on” (in dashed box in Fig. 5(a)) to the MDG in Fig. 5(a). Then we run our algorithm on the new MDG and obtain five mode combinations (without \* in Fig. 7). This result exactly matches the mode combinations in manually designed results [10].

Through this simple example, we show our algorithm is able to systematically generate legal mode combinations, and by editing the mode dependency graph, we can obtain mode combinations without manually going through all possible mode combinations.

## 4 Mode Selection

Mode selection works as a post-processing stage after scheduling. It validates and improves the schedule with more architectural knowledge than the scheduler. Our approach is a constraint-driven search algorithm that considers resource/task dependency and mode change overhead, and tries to find a mode schedule that satisfies system timing and power constraints.

### 4.1 Problem Statement

The input to the problem consists of a set of tasks  $X$ , a schedule  $\sigma$ , a mode dependency graph  $G$ , power constraints  $P_{max}$  and  $P_{min}$ , and timing constraints represented by constraint graph  $G_c$  [4]. The output is a mode schedule  $\sigma'$  that meets system power and timing constraints by means of legal mode combinations.

**Definition 4 (Task  $x \in X$ )** A task  $x$  is defined by a tuple  $(\tau_x, \omega_x)$ , where  $\tau_x$  is a task identifier, and  $\omega_x \in \Omega$  is the workload of the task. In the context of this paper, we assume each task  $x$  has already been mapped to a resource  $\gamma$ . The operation delay  $d_x$  and power profile  $P_x(t)$  of a task  $x$  depend on the workload  $\omega_x$  and the selected modes  $m$  of resource  $\gamma$ .

Depending on the nature of the resource, workload  $\omega_x$  can be the number of cycles for a processor, the number of atomic actions for a device, e.g., the number of steps for a step motor, or simply the time to perform a task.

**Definition 5 (Schedule  $\sigma$ )** A schedule  $\sigma$  maps each task to its start time. An *idle interval* with respect to a schedule  $\sigma$  and a resource  $\gamma$  is a time interval during which no task is scheduled to run on  $\gamma$ . Note that during an idle interval, the resource can still consume nonzero power, depending on the mode.

**Definition 6 (Mode schedule  $\sigma'$ )** A mode schedule  $\sigma'$  maps each task  $x \in X'$  (which is mapped to resource  $\gamma$ ) to the task's start time and a mode  $m \in M_\gamma$ , where  $X' = X \cup X_o$ .  $X_o$  is a set of *overhead tasks*, which are inserted whenever there is a mode change on a given resource.

A mode schedule  $\sigma'$  is *feasible* if all mode combinations are legal (Section 3) and all timing and power constraints are satisfied at all times:

$$P_{min} \leq \sum_{\gamma \in \Gamma} P_\gamma(t) \leq P_{max} \quad 0 \leq t \leq t_{end} \quad (3)$$

$$T_{min}(u, v) \leq \sigma'(v) - \sigma'(u) \leq T_{max}(u, v) \quad \forall u, v \in \text{task set } X(4)$$

```

MODE_SELECTION( $\sigma, G, P_{max}, P_{min}, G_c$ ):
0  /* input : schedule  $\sigma$  */
1  /*      power constraints  $P_{max}$  and  $P_{min}$  */
2  /*      timing constraint graph  $G_c$  */
3  /*      mode dependency graph  $G$  */
4  /* output: a feasible mode schedule  $\sigma'$  */
5  run ModeGen_From_Cyclic_MDG( $G$ )
6  for each  $\lambda$  in  $\Lambda$  {
7      map  $\lambda$  to tasks  $T$  in  $\sigma$ , get  $\sigma_1$ 
8      if check_timing( $\sigma_1, G_c$ ) = TRUE {
9          decompose  $\sigma_1$  into time intervals  $S$ 
10         for each  $s \in S$  {
11             select modes for idle intervals
12             while  $P_{max}$  and  $P_{min}$  not satisfied {
13                 select other modes for idle intervals
14             }
15         } So far we obtain a mode schedule  $\sigma'$ 
16         add mode change overhead as new tasks into  $\sigma_1$ , get  $\sigma_2$ 
17         if check_timing( $\sigma_2, G_c$ ) = TRUE AND
18             check_power( $\sigma_2, P_{max}, P_{min}$ ) = TRUE {
19             return  $\lambda \cup \text{modes selected for idle intervals}$  }
20         }
21     }
22 }

```

Figure 8. Top level search algorithm.

where  $t_{end}$  is the overall schedule length, and  $P_{min}$  and  $P_{max}$  are the minimum and maximum power constraints, respectively. The reason for a minimum power constraint has been discussed elsewhere [4]. It can be used for not only power/performance tradeoffs but also for jitter control.

### 4.2 Algorithm

Our search algorithm contains a loop with two steps. First we find modes for tasks that satisfy task dependency and timing constraints. Second we determine modes for the idle intervals on each resource. Note that after the first step, the operation delay for certain tasks may be changed due to certain mode selected (i.e., modes of different clock rate due to voltage scaling) or task dependency. An advantage of selecting task modes and idle interval modes separately is that we can apply different kinds of system constraints, which help prune out illegal mode combinations efficiently. We reorder the modes for each resource by their power consumption in increasing order and search from the smallest one. By doing so we both speed up our search process and find solutions very close to the energy-optimal solution. The top level algorithm is shown in Fig. 8.

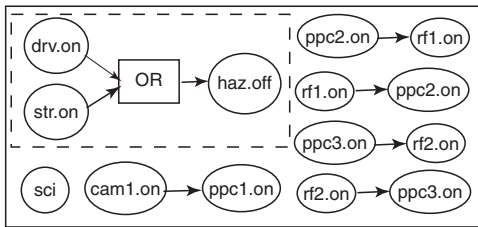


Figure 9. The MDG for the microover.

### Selecting modes for tasks

We select modes for tasks by generating legal mode combinations of tasks that satisfy the MDG. Note that the MDG used for a schedule may be a mix of resource dependency and task dependency, which represent time-invariant and time-variant dependency of resources. For example, in Fig. 9, the sub-graph in the dashed box represents *resource* dependency, whereas the rest of the graph shows the *task* dependency. We can still use the algorithm introduced in last section to generate legal mode combinations of tasks. Once a legal mode combination is determined, we can obtain a new schedule since the operation delay of tasks become known under their selected modes and under their co-activation dependency. We check timing constraints for the new schedule. If it fails, we generate another legal mode combinations and check again; if it passes, we use the mode combination for mode selection of idle intervals.

### Selecting modes for idle intervals

On each resource, overhead may exist at the mode changes. We find a set of modes for each idle interval such that the time overhead of the mode changes is less than the length of the idle interval. We treat overhead as additional tasks to the schedule we obtained. We characterize those overhead tasks with time and average power, which can be derived from time and energy information. We decompose the new schedule into *time intervals* such that within each time interval there is no task event (start or end event). The decomposition is done in the following way: We find the start and end events of all tasks. All the events cut the time axis into non-overlapping segments. Each segment forms a time interval. We check system power constraints in each time interval. If the schedule fails power constraints, we attempt a mode change on resources that currently have an idle interval, and check power constraints again. If all the modes fail the power constraints, we backtrack to the previous time interval. If we backtrack to the beginning of the schedule and still cannot find feasible modes, we attempt the next legal mode combination and select modes for idle intervals again.

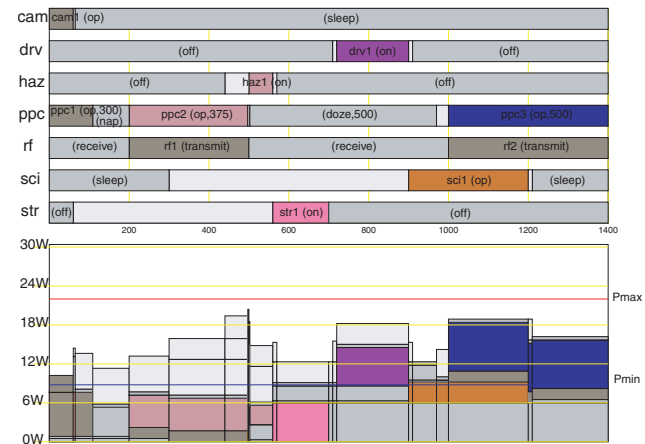


Figure 11. A mode schedule for microover.

## 5 Experimental Results

We apply our algorithm to an example based on the Mars rover [11]. The rover travels on the surface of Mars to perform scientific experiments and shoot images. Its resources consist of a camera (CAM), scientific devices (SCI), a radio-frequency modem (RF), a microprocessor (PPC), a hazard detector (HAZ), driving motors (DRV) and steering motors (STR). CAM takes a picture, sends the picture data to PPC for processing, PPC outputs to RF, and then the rover moves to another location (HAZ, DRV, STR) to perform scientific experiments (SCI, PPC, RF).

PPC can work at a number of different clock rates (with a full speed of 500MHz) and can be set to *doze*, *nap* or *sleep* modes. RF can be in *rx* only mode, *tx-rx* mode and *sleep* modes. The other resources have only two modes each, *on* and *off*. Mode-change overhead is significant for some resources. Due to the low temperature on Mars, DRV must be pre-heated for some time before turned on. Similar reason applies to STR, RF, and SCI. The inter-resource relationships are shown in Fig. 9. For example, when HAZ is working, neither DRV nor STR should be working. RF may be in *tx-rx* mode if and only if the processor is operating.

Fig. 11 shows a feasible mode schedule, in both time view and power view. Task *ppc2* on PPC cannot be further slowed down because PPC and RF must be co-active. If PPC is greedily slowed down, it will violate max power constraint during the interval 500 - 560. Task *drv1*, *haz1* and *str1* are not overlapped due to the system requirement specified in the mode dependency graph. STR and SCI need significant time to pre-heat, which is adequately considered (the light areas in their tracks). Idle interval between *rf1* and *rf2* on RF is set to *rx* only rather than *off* because the timing overhead of mode changes (including pre-heating)

| Scenario | Task sequence | $Cost_{simple}$ | $Cost_{greedy}$ | $Cost_{modesel}$ | Relative energy (simple/greedy/modesel) |
|----------|---------------|-----------------|-----------------|------------------|---|
| A        | CAM/MOV/SCI   | 19002           | 18442           | 17935            | 100% /97.0%/93.4%                       |
| B        | MOV/CAM/SCI   | 16381           | 15013           | 14667            | 100% /91.6%/89.5%                       |
| C        | CAM/SCI/MOV   | 20294           | 19505           | 19014            | 100%/96.1%/93.6%                        |

Mission tasks: CAM: shoot images; MOV: move to another location; SCI: perform scientific experiments.  
Approaches: simple: assume two modes; greedy: greedily voltage scaling; modesel: our algorithm.

**Figure 10. Comparison among different working scenarios.**

is larger than the length of the interval. The idle interval before  $rf1$  is set to  $rx$  only for the same reason.

We compared our algorithm with two other approaches: approach one assumes only two modes, on and off; approach two greedily applies voltage scaling technique whenever possible (we allow power constraint violation in this approach). The results are shown in Fig. 10. Approach one gives the worst results because it never utilizes available modes. Approach two is better than approach one since it saves energy by applying voltage scaling technique, but its greediness pays the cost since its saving by slowing down the processor is more than offset by the extra energy consumed on the RF modem. And in all the scenarios, approach two violates max power constraint. Our algorithm gives the best results because we utilize multiple modes of resources and apply voltage scaling on the processor. At the same time, we avoid extra energy cost on RF by identifying co-activation dependency between the two resources and performing mode selection to find the feasible solution.

## 6 Conclusions

This paper presents a method for capturing mode dependency and an algorithm for mode selection in power-aware embedded systems. The mode dependency graph introduced in this paper enables legal combinations of modes to be systematically derived. Today's designers perform this task manually. However, as components offer increasingly sophisticated modes for power management, while at the same time imposing even more restrictions on mode changes, the complexity will grow quickly beyond what humans can handle. Our MDG represents a structured approach to controlling the complexity of power management. We also present a search algorithm that takes advantage of the MDG. By considering power/timing constraints and overhead on transitions, this technique gives designers more confidence in the feasibility of the synthesized results in real-life applications. Furthermore, our algorithm incorporates heuristic ordering to optimize for the energy cost of the solution, and it shows realistic, *system-level* improvements over previous techniques that either do not handle constraints or multiple components.

## Acknowledgment

This research was sponsored by DARPA under contract F33615-00-1-1719. It represents a collaboration between the University of California at Irvine and the NASA/Cal Tech Jet Propulsion Laboratory. Special thanks to Dr. N. Aranki, Dr. B. Toomarian, Dr. M. Mojarradi and Dr. J. U. Patel at JPL and Kerry Hill at AFRL for their discussion and assistance.

## References

- [1] L. Benini, G. Paleologo, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. *IEEE Trans. Computer-Aided Design*, 18:813–833, June 1999.
- [2] I. Hong, D. Kirovski, G. Qi, M. Potkonjak, and M. B. Srivastava. Power optimization of variable voltage core-based systems. In *Proc. of DAC*, pages 176–181, 1998.
- [3] C.-H. Hwang and A. Wu. A predictive system shutdown method for energy saving of event-driven computation. In *Proc. of DAC*, November 1997.
- [4] J. Liu, P. Chou, and N. Bagherzadeh. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *Proc. 38th DAC*, pages 840–845, June 2001.
- [5] J. Luo and N. K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. In *Proc. of DAC*, pages 444–449, 2001.
- [6] Q. Qiu, Q. Wu, and M. Pedram. Dynamic power management of complex systems using Generalized Stochastic Petri Nets. In *Proc. of DAC*, pages 352–356, 2000.
- [7] G. Quan and X. S. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proc. of DAC*, pages 828–833, 2001.
- [8] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proc. of DAC*, pages 134–139, 1999.
- [9] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proc. of ICCAD*, pages 365–368, 2000.
- [10] A. Sinha and A. Chandrakasan. Operating system and algorithmic techniques for energy scalable wireless sensor networks. In *Proc. 2nd Int. Conf. on Mobile Data Management*, pages 199–209, January 2001.
- [11] H. Stone. Mars Pathfinder Microrover: A low-cost, low-power spacecraft. In *Proc. the 1996 AIAA Forum on Advanced Developments in Space Robotics*, August 1996.
- [12] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. *IEEE Annual Foundation of Computer Science*, pages 374–382, 1995.