

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

Category: 1) Design: low power  
2) Design: system level

## Abstract

Among the many techniques for system-level power management, it is not currently possible to guarantee timing constraints and have a comprehensive system model at the same time. Specifically, dynamic power management (DPM) techniques can model systems consisting of multiple devices with multiple mode settings. However, if hard real-time constraints are required, then DPM techniques are usually not applicable, because they are based on prediction or timeout and are designed for applications without hard constraints. Instead, low-power scheduling (LPS) techniques may be used to satisfy timing constraints, but they assume a single processor with voltage or frequency scaling. These greedy schedulers are not generalizable to multiple processors or devices with more general modes.

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, including sleep mode, doze mode, etc, 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. Mode selection poses several new problems, including mode modeling at the component level and mode dependency at the system level.

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 and how they change. 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. This is because when a component is in a certain mode, certain transitions are prohibited. The power manager 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 high 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. Mode selection must be coordinate even within a component, not just at the system level. For example, voltage and frequency scaling may appear arbitrarily controllable individually, but they are not arbitrarily controllable when combined: it may be necessary to change the frequency first before voltage when speeding up, or in reverse order when slowing down, depending on the process technology.

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. As a result, the complexity of a naive implementation of a system-level power manager can

grow exponentially, making it impractical for on-line power management or difficult to validate its correctness. 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. We envision that the intra-component level mode dependency can be modeled as a component library, while the system-level dependency can be automatically extracted from a high-level description of the application. Second, this paper presents a new mode selection algorithm that produces a mode schedule that satisfies timing and power constraints 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 ?? reviews related work. Section ?? presents the mode dependency graph, while Section ?? describes a mode selection algorithm that takes advantage of mode dependency modeling. We discuss the experimental results in Section ??.

## 2 Related Work

Many techniques for power management and optimization for embedded systems have been presented in the literature. One class of techniques, low-power scheduling (LPS), scales down voltage to save power without violating timing constraints. However, LPS's notion of a system is currently limited to a single processor without considering peripheral devices, and the results are not generalizable to multiple processors. On the other hand, dynamic power management (DPM) techniques, which are based on timeout or event prediction, assume multiple components with multiple modes. DPM can be effective for minimizing energy and time penalties on average, but it cannot make strong timing guarantees. This section reviews related work in these two categories.

### 2.1 Low-Power Scheduling (LPS)

Voltage/clock scaling techniques [?] have been applied to real-time task scheduling problems to obtain power/energy savings [?, ?, ?, ?]. 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 [?] proposed a heuristic for scheduling real-time tasks on a single variable voltage processor. Shin [?] exploited both execution time variation and idle time intervals for fix-priority tasks. Shin's algorithm in [?] determines the lowest maximum processor speed for each job to achieve power reduction. Quan and Hu [?] further greedily determine the lowest voltage for a set of tasks to achieve more energy savings.

What these LPS 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 LPS techniques are not generalizable to multiple resources under power constraints, as shown in the following example.

*Example: (LPS fails in multi-resource)*

Fig. ??(a) shows a Gantt chart (on top) and the corresponding power profile (bottom) for a system with three resources:  $R_1$  is capable of voltage/clock scaling, while  $R_2$  and  $R_3$  are not voltage scalable. The task  $t_1$  on  $R_1$  has a deadline at time 110. Furthermore, the behavior of

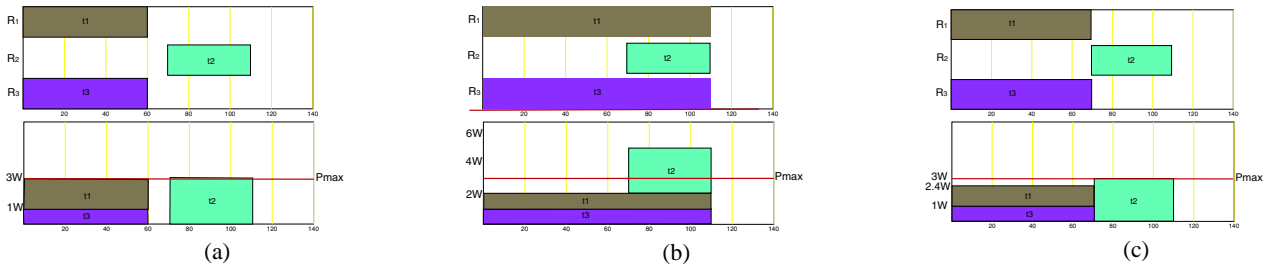


Figure 1: An application scenario that has resource dependency. (a) Initial schedule and power profile; (b) greedy voltage/clock scaling results in a power spike that violate maximum power constraint; (c) a feasible solution meets both power and timing constraints, and saves energy as well.

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

Figure 2: Comparison of three schedules, greedy voltage/clock scaling may violate the power constraint.

the application dictates that  $R_1$  and  $R_3$  be *co-active* (e.g., the two devices that must work together). The system also has a maximum power constraint of  $3W$ .

Fig. ??(b) shows the schedule (top) and the power profile (bottom) obtained by greedily slowing down  $R_1$  to achieve energy saving. Even though all timing constraints are satisfied, it violates power constraints and it is not minimum energy. The max power constraint is violated because when task  $t_1$  is stretched out, and when it overlaps task  $t_2$  during the time interval from 70 and 110, 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 voltage is not scalable (as stated in previous paragraph), but its execution is prolonged by  $R_3$ .

The optimal schedule and power profile are shown in Fig. ??(c). Resource  $R_1$  is slowed down without overlapping task  $t_2$  on Resource  $R_2$ . This way, no max power is violated. Although task  $t_3$  on resource  $R_3$  is stretched with task  $t_1$  and therefore consumes more energy than in Fig. ??(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. ?? summarizes the energy costs.

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.

In [?, ?], Luo and Jha present static scheduling for multiple processing elements (PEs). They re-order tasks and apply voltage scaling in this post-processing step after scheduling to smooth the system-level power profile. Their tasks have precedence and timing constraints, though power is a design goal rather than a constraint. Our approach is similar in that it can also be a post processing step after scheduling, 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 powerup [?, ?]. Stochastic models [?, ?] are used to address the uncertainty in system behaviors. Simunic et al [?] combine stochastic-modeled power management techniques together with dynamic voltage scaling techniques and achieve significant power reduction in portable systems. Although the above techniques are effective to reduce system power consumption, they do not treat power or timing as hard constraints, but as costs or penalties. Most of DPM techniques assume single resource scenarios and therefore did not consider

	DPM		LPS		MS
	[?, ?]	[?, ?]	[?, ?, ?, ?]	[?, ?]	
Timing as constraint	N	N	Y	Y	Y
Power as constraint	N	N	N	N	Y
Timing overhead	Y	Y	N	N	Y
Power overhead	Y	Y	N	N	Y
Multiple resources	N	Y	N	Y	Y

Figure 3: Comparison of dynamic power management (DPM), low power scheduling (LPS), and mode selection (MS).

inter-resource dependency. Qiu, Qu and Pedram in [?] model multiple service providers and their Generalized Stochastic Petri Net (GSPN) model can potentially model dependency among resources. However, only one server is needed to process an incoming request, and their GSPN model is mainly for the request/dispatch behavior of servers rather than dependency among the servers themselves.

Fig. ?? summarizes the features of the techniques surveyed here. The last column shows our new approach, mode selection, which 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, which forces another resource to select a given mode; it may also be exclusion, enabling, and many other possible types of dependency. These dependencies may be extracted from application level specifications or policies for safety, security, fault-tolerant, power-saving, or may be explicitly specified as mode correlation [?]. 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:  $F : M_\gamma \times M_\gamma \rightarrow T \times En$ , where  $M_\gamma$  is the set of modes of resource  $\gamma$ , and  $T, En$  are time and energy, respectively. The average power can be obtained from energy and time information.

For example, a processor may have **active**, **idle**, and **sleep** modes. Changing from any mode to other modes incurs time and energy overhead. Usually wakeup from **sleep** to **active** mode needs more time and energy than from **idle** to **active** mode. At architectural level, mode change overhead may be time and energy of changing hardware configurations only. At application level, it may include overhead to restore the context, reinitialize the OS or load a program.

The modes are not limited to simple power modes like on, off, doze, nap, or sleep, but also encompass cache configurations for processors, different encoding/decoding techniques for radio antenna, variations of compiling techniques for software components, and different transmission protocols for bus drivers.

**Definition 2 (Mode combination  $\pi \in \Pi$ )** Given  $N$  resources  $(\gamma_1, \gamma_2, \dots, \gamma_N)$ , a mode combination of the resources (tasks) is a combination of  $N$  modes  $\pi = (m_1, m_2, \dots, m_N)$ .  $m_i$  is a mode assigned to resource  $\gamma_i$  (task  $\tau_i$ ),  $1 \leq i \leq N$ .

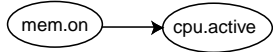


Figure 4: Mode dependency example: the memory is on only if the CPU is in active mode.

AND			OR			XOR		
A	B	C	A	B	C	A	B	C
T	T	T	T	T	T	T	T	F
T	F	F	T	F	T	T	F	T
F	T	F	F	T	T	F	T	T
F	F	F	F	F	F	F	F	F
X	U	U	X	U	U	X	U	U
U	X	U	U	X	U	U	X	U

Figure 5: Truth table for AND, OR, and XOR operators. C is the output of A op B. T: True; F: False; X: don't care; U: undetermined.

### 3.2 Mode Dependency Graph

A mode dependency graph (MDG)  $G_m(V, E)$  represents the inter-resource dependency relationships, where a vertex  $V$  is a resource mode, and a directed edge  $E$  connects two vertices. Each vertex is represented by a circle with a label in the format of “ $res.mod$ ,” where  $res$  is the resource and  $mod$  is the mode of the resource. If two vertices have the same labels, we considered them identical.

The value of a vertex  $|V|$  is defined as:

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

An edge in the MDG represents dependency between two modes. Suppose an edge  $(u, v) \in E$ ,  $u = res1.mod1$ ,  $v = res2.mod2$ . The two modes  $mod1$  and  $mod2$  satisfy the mode dependency graph if:

$$|u| \text{ is } True \text{ only if } |v| \text{ is } True. \quad (2)$$

In other words, if  $|v|$  is *False*,  $|u|$  cannot be *True*. For example, we can represent the dependency between a CPU and a memory chip such that the memory is on only if the CPU is in active mode (see Fig. ??). If the CPU is in sleep mode and the memory is on, then it violates the mode dependency. If the CPU is in sleep mode, and the memory is off, then it does not violate the mode dependency. Of course, if the CPU is in active mode and the memory is on, it satisfies 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. An operator vertex  $V_{op}$  has at least two fan-in, and at least one fan-out, meaning that  $V_{op}$  has at least two vertices pointing to it and it points to at least one vertex.

The value of an operator vertex can be obtained by evaluating the logic functions that the graph represents. We define the operators AND, OR, XOR, and MUTEX. The truth tables of operator AND, OR, and XOR are listed in Fig. ?. The meaning of AND, OR and XOR follows the normal boolean functions in the same names except when any input is “undetermined,” the output is “undetermined.” When the operation of MUTEX (not listed in Fig. ??) is binary, its meaning is the same as XOR; when MUTEX has multiple inputs (more than two), the output is True if and only if one input is True and the rest of input are False.

The extended mode dependency graph also follows “only-if” interpretation. For example, the top left item in Fig. ??(a) shows the dependency among a sensor (S), a radio device (R) and a processor (A). The semantics expressed here is that S and R are both off only if A is in sleep mode. If, for example, S is off, R is off, and A is in active mode, it violates the dependency. If S is on, R is off, and A is in sleep mode, it does not violate the dependency.

Fig. ??(a) shows the mode dependency graph of the microsensor example[?]. The microsensor system consists of a sensor to sense the environment, a processor, a memory module, and a radio modem. In this system, the behaviors and dependencies of the devices can be

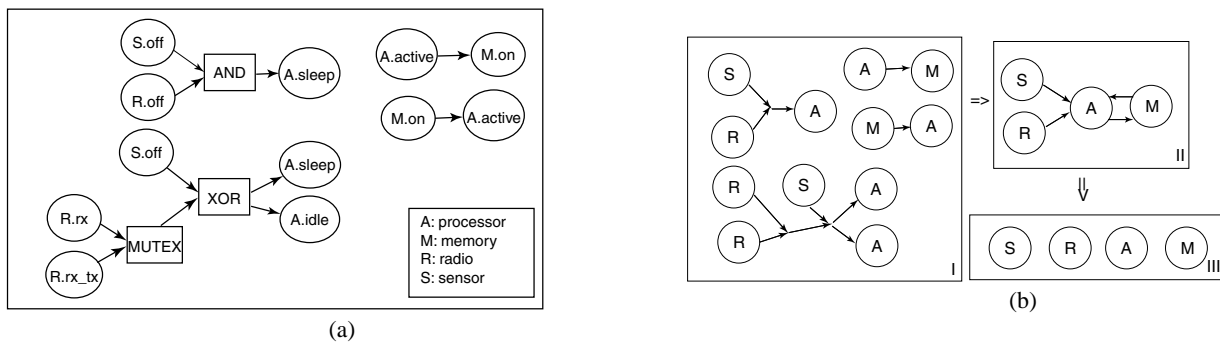


Figure 6: (a) An Mode Dependency Graph example: micro sensor. (b) Reduce an MDG to a resource list for generating mode combinations. Box I: from the MDG, shrink each operator vertex to a point, and remove mode name in each mode vertex. Box II: remove the redundant vertices and edges. Box III: break the cycle by removing one edge in the cycle, and apply topological sort to obtain a resource list.

derived from high-level power management policies: the sensor and the radio are both off only if the processor is in sleep mode; either of them is on only if the processor is in sleep mode or idle mode; both the sensor and the radio are on only if the processor is in active mode; the memory is on if and only if the processor is active.

### 3.3 Generating Mode Combinations

This section shows how to efficiently generate legal mode combinations using the mode dependency graph (MDG). The core algorithm (Fig. ??(a)) assumes no cycle in the MDG, whereas an extended version (Fig. ??(b)) handles cyclic dependencies.

We reduce an MDG to a resource list which will be used to generate mode combinations. From the given MDG, we first remove mode names from all mode vertices, and shrink each operator vertex to a point but keep all the edges (see Fig. ??(b)). Then we remove the redundant vertices and edges but keep their connections. Now we obtain a new graph that each resource has only one vertex instance. If this graph is cyclic, we remove one edge from each cycle and obtain an acyclic graph, then we apply topological sort algorithm to the acyclic graph and finally obtain a resource list. Due to the limitation of the paper length, we omit the details of this process.

If the MDG is acyclic, then legal mode combinations can be generated by a special version of topological traversal. By checking it against an MDG, modes of a resource  $\gamma$  that do not satisfy the MDG can be identified because upon checking the resource  $\gamma$ , all the modes of its dependent resources have been already determined since they are all located before  $\gamma$ . We discard illegal modes and keep only legal modes for checking with succeeding resources and for generating mode combinations. We progressively generate an mode combination as we check legality of modes at each resource and select one of the legal modes. As we reach the end of the resource 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. The algorithm to generate mode combination is shown in Fig. ??(a).

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  $\gamma$  *P-resource*. Additional information needs to be recorded to ensure that we never neglect checking any constraints in MDG. In this scenario, we basically keep track of which resources  $\gamma$  is dependent on, and when the modes of all dependent resources are determined, we evaluate a mode of  $\gamma$  to determine whether the mode satisfies the MDG. The detailed algorithm is shown in Fig. ??(b).

### 3.4 Example: Microsensor

A microsensor is a node in a distributed microsensor network. It consists of a sensor, a processor, memory chips, radio frequency module and other auxiliary parts. When the sensor obtains information from environment, it sends data to the processor, the data is processed and sent to a based station or other network node via RF module.

```

ModeGen_From_Acyclic_MDG( $G_m$ ):
1  /* input: mode dependency graph  $G_m$  */
2  /* output: legal mode combination  $\pi$  */
3  preprocess to get a resource list  $L$ 
4   $p \leftarrow 0$ , reset mode combination  $\pi \leftarrow \emptyset$ 
5  while  $p \geq 0$  {
6    while  $0 \leq p < \text{length}(L)$  {
7      if found an unmarked mode  $m$  for resource  $L[p]$  {
8        if  $\text{check\_MDG}(m, G_m) = \text{TRUE}$  {
9           $\pi[p] \leftarrow m, p \leftarrow p + 1$ 
10         }
11         mark the mode  $m$ 
12       } else {
13         unmark all modes of current resource  $L[p]$ 
14          $p \leftarrow p - 1$ 
15       }
16     if  $p = \text{length}(L)$  { output  $\pi$  }
17      $p \leftarrow p - 1$ 
18     if found an unmarked mode  $m$  for the resource  $L[p]$  {
19       unmark all modes for current resource
20        $p \leftarrow p - 1$ 
21     } else {
22       if  $\text{check\_MDG}(m, G_m) = \text{TRUE}$  {
23          $\pi[p] \rightarrow m$ , output  $\pi, p \leftarrow p + 1$  }
24     }
25 }

```

(a)

```

ModeGen_From_Cyclic_MDG( $G_m$ ):
1  /* input : mode dependency graph  $G_m$  */
2  /* output : legal mode combination  $\pi$  */
3  preprocess to get a resource list  $L$ 
4  mark out P-resources in  $L$ 
5   $p \leftarrow 0$ , reset a mode combination  $\pi \leftarrow \emptyset$ 
6  while  $p \geq 0$  {
7    while  $0 \leq p < \text{length}(L)$  {
8      if  $L[p]$  is not a P-resource {
9        if found an unmarked mode  $m$  for task  $L[p]$  {
10         if  $\text{check\_MDG}(m, G_m) = \text{TRUE}$  {
11           if  $L[p].\text{cached}$  is not empty {
12             if all cached resources satisfy  $G_m$  is TRUE {
13                $\pi[p] \leftarrow m, p \leftarrow p + 1$  }
14             } else {  $\pi[p] \leftarrow m, p \leftarrow p + 1$  }
15           }
16           mark the mode  $m$ 
17         } else { unmark all modes of resource  $L[p], p \leftarrow p - 1$  }
18       } else {
19         locate the last resource  $L[q]$  in  $L$  that  $L[p]$  is dependent on
20          $L[q].\text{cached} \leftarrow L[p]$ 
21       }
22     }
23      $p \leftarrow p + 1$ 
24   }
25   if  $p = \text{length}(L)$  { output  $\pi$  }
26    $p \leftarrow p - 1$ 
27   if found an unmarked mode  $m$  for the resource  $L[p]$  {
28     unmark all modes for current resource
29      $p \leftarrow p - 1$ 
30   } else {
31     if  $\text{check\_MDG}(m, G_m) = \text{TRUE}$  {
32        $\pi[p] \leftarrow m$ , output  $\pi, p \leftarrow p + 1$  }
33   }

```

(b)

Figure 7: (a) Generate mode combinations for acyclic MDG. (b) Generate mode combinations for cyclic MDG.

component	mode
sensor (S)	on, off
processor (A)	active, idle, sleep
memory (M)	on, off
Radio (R)	rx, tx_rx, off

Figure 8: Modes of components in a microsensor.

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 9: Mode combinations generated from MDG.

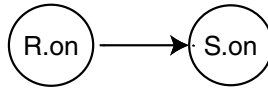


Figure 10: Additional requirement for the microsensor.

The modes for the component is summarized in Fig. ???. 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, receive-only (**rx**), transmit-and-receive (**tx\_rx**), and **off**. There are a total of 36 mode combinations for these components.

The inter-component relationships are specified using MDG as shown in Fig. ??(a). The graph follows the “only-if” interpretation, and the graph semantics comes from system level power saving policies. For example, the sensor and radio are both **off** only if the processor is in **sleep** mode. Either of them (not both) is **on** only if the processor is in **sleep** or **idle** mode. The processor is **active** only if the memory is **on**, and vice versa.

Using the MDG, our algorithm automatically generate eight mode combinations that satisfy the given MDG (see Fig. ??). The generated mode combinations can be used by system power manager as legal modes in system level power management.

Suppose we want the microsensor to work in a proactive way. That is, if the system is off, it can only be waken up by the sensor when it senses information from environment. The radio cannot wake up the system by receiving a remote command. Thus when the sensor is **off**, there is no reason for the rest of system (including the radio) to be **on**. We add another item “the radio is **on** only if the sensor is **on**” (in Fig. ??) to the MDG in Fig. ??(a). Then we run our algorithm on the new MDG and obtain five mode combinations (modes without \* in Fig. ??). This result exactly matches the mode combinations in manually designed results [?].

Through this simple example, we show our algorithm is able to systematically generate legal mode combinations that satisfy inter-component dependencies, 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, validate and improve the schedule with more application-level and architectural knowledge than the scheduler. Our approach is a constraint-driven searching algorithm that considers resource/task dependency and mode change overhead, and try 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  $G_m$ , power constraints  $P_{max}$  and  $P_{min}$ , and timing constraints represented by constraint graph  $G_c$  [?]. The output is a mode schedule  $\sigma'$  that meets system power and timing constraints by means of legal mode combinations.

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  $P_x(t)$  depend on the workload  $\omega_x$  and the selected modes  $m$  of resource  $\gamma$ .

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. 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  $m \in M_\gamma$ . Note that  $X'$ , the new task set, consists of all of the original task set  $X$  unioned with *overhead tasks*, which are inserted whenever there is a mode change on a given resource.

```

MODE_SELECTION( $\sigma, P_{max}, P_{min}, G_c, G_m$ ):
0  /* input : schedule  $\sigma$  */
1  /*      power constraints  $P_{max}$  and  $P_{min}$  */
2  /*      timing constraint graph  $G_c$  */
3  /*      mode dependency graph  $G_m$  */
4  /* output: a feasible mode schedule  $\sigma'$  */
5  while ModeGen_From_MDG( $G_m$ ) not finished {
6      obtain a mode combination  $\pi$  that satisfies  $G_m$ 
7      bind  $\pi$  to tasks  $T$  in schedule  $\sigma$ , derive a new schedule  $\sigma_1$ 
8      if check_timing( $\sigma_1, G_c$ ) = TRUE {
9          decompose  $\sigma_1$  into time intervals  $S$ 
10         while find modes for idle intervals not finished {
11             for each  $ts \in S$  {
12                 find modes for idle intervals that all resources satisfy  $P_{max}$  and  $P_{min}$ 
13             } # so far we obtain a mode schedule  $\sigma'$ 
14             add mode change overhead as new tasks into  $\sigma_1$ , get a new schedule  $\sigma_2$ 
15             if check_timing( $\sigma_2, G_c$ ) = TRUE AND
16                 check_power( $\sigma_2, P_{max}, P_{min}$ ) = TRUE {
17                 return  $\sigma'$  }
18         }
19     }
20 }

```

Figure 11: Top level search algorithm.

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

$$rP_{min} \leq \sum_{\gamma \in \Gamma} P_{\gamma}(t) \leq P_{max} \quad \forall \text{ time } t \quad (3)$$

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

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

## 4.2 Algorithm

Our searching algorithm is an iteration of two steps. First we find modes for tasks that satisfy task dependency as well as timing constraints. Second we determine modes for those 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 frequencies due to voltage/clock scaling), or due to task dependency. When determining modes for idle intervals, we check whether selecting one mode leaves enough time for mode changes, and we also check system power constraints once all resources have been selected modes. 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 average power consumption in an ascending order and search from the smallest one. By doing this 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. ??.

### 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 a resource dependency and task dependency, which represent time-invariant and time-variant dependency of resources (see Fig. ??). We can still use the algorithm introduced in last section, to generate mode combinations of tasks. Single vertices that connect to no other vertices represents resources that their modes are independent on that of any other resources. Co-activation of two tasks can be identified by

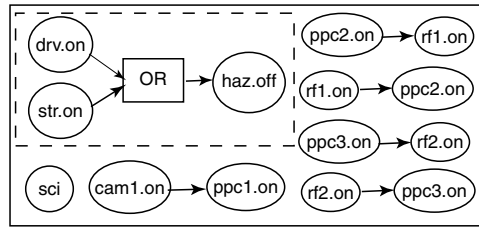


Figure 12: The mode dependency graph of the Microrover example: sub-graph in dashed box represents *resource* dependency among a hazard detector (haz), a driving motor (drv) and a steering motor (str). The rest of the graph shows the *task* dependency among tasks on a processor (ppc1, ppc2, and ppc3), tasks on radio device (rf1 and rf2), and tasks on a camera (cam1). Single vertex sci represents a resource that is independent on any other resource.

cycles in the MDG. For example, in Fig. ??, task ppc2 and rf1, task ppc3 and rf2 are co-active tasks.

Once a legal mode combination is generated, we are able to find out operation delay of tasks under their selected modes and under their co-activation dependency as well. Then we obtained a new schedule. We check timing constraints for the new schedule. If it fails, we generate next legal mode combinations of tasks and check again; if it passes, we use the mode combination for mode selection of idle intervals.

#### *Selecting modes for idle intervals*

Idle intervals are the intervals between tasks on each resource. Overhead may exist at the mode changes from a task to its next idle interval and from the idle interval to the next task. On each resource, we find out a set of modes for each idle interval that the time overhead of the mode changes is less than the length of the idle intervals. We sort the modes in each set in an ascending order, and use modes in these sets to select modes for idle intervals and check power constraints.

We look at overhead as additional tasks to the schedule we obtained when we have selected modes for tasks. Usually the power profile of mode changes is not available for most resources. 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 event or end event of a task). Decomposition of the schedule facilitate checking system power constraints. The decomposition is done in the following way: We find the start events and end events of all tasks. All the events cut the time axis into non-overlapping segments. Each segment forms a time interval.

From beginning to the end of the new schedule, we check system power constraints in each time interval. If the schedule fails power constraints at any time interval, we attempt a mode change on resources that currently have an idle interval, and we check power constraints again. If all the modes for those resources 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 for tasks and go back to select modes for idle intervals again.

Our algorithm applies heuristic ordering by selecting modes from lowest power values. This way we are able to find solutions with less total energy efficiently.

## 5 Experimental Results

We apply our algorithm to an example based on the Mars rover [?]. The rover travels on the surface of Mars to perform scientific experiments and shoot images. Its resources consists of a camera (CAM), scientific devices (SCI), a radio-frequency modem (RF), a microprocessor (PPC), a hazard detector (HAZ), driven 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,

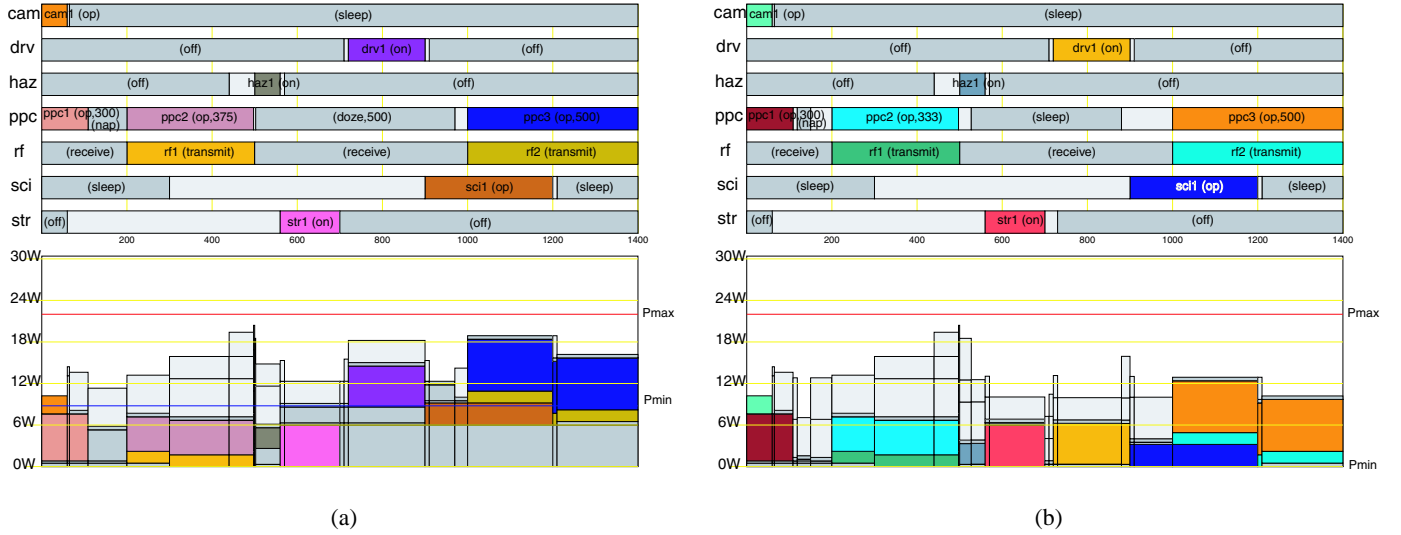


Figure 13: (a) Mode schedule with maximum and minimum power constraints, the processor PPC cannot be greedily slow down due to the maximum power constraint. (b) A mode schedule with maximum power constraint only. The result is similar to a low power schedule.

RF).

Each resource has its own working modes. The microprocessor can work at a number of different clock rates (with a full speed 500MHz) and can be set to **doze**, **nap** or **sleep** modes. RF modem can be in **receive only** mode, **transmit-and-receive** mode and **sleep** modes. The other resources have only two modes, **on** and **off**. Mode-change overhead is significant for some resources. Due to the low temperature on Mars, before they are turned on, the driving motors must be pre-heated for some time. Similar reasons apply to steering motors, RF modem, and scientific devices. Moreover, the power and timing overhead of the above resources are not constant, but are functions of the ambient temperature. For example, the pre-heating time  $d_{off\_on}$  to change mode of driving motors from **off** to **on** is defined as:

$$d_{off\_on} = \begin{cases} (-1.875 \cdot t + 10^\circ C) & \text{if } t < 0, \\ 10^\circ C & \text{if } t \geq 0. \end{cases} \quad (5)$$

The inter-resource relationships are shown in Fig. ???. For example, when hazard detector is working, neither the driving motor and steering motor should be working. The RF modem is in transmit and receive mode if and only if the processor is processing data and communicating with the RF modem.

A feasible mode schedule is presented in Figure ??(a). The upper part of the figure shows the time view of the mode schedule, where each horizontal track represents an instance of a resource. The lower part shows the corresponding power profile, which is obtained by summing up the power consumption of all tasks, idle intervals and overhead. Each task is colored and labeled with task name and mode name, while the gray areas are idle intervals labeled only with mode names, and the light gray areas represent mode change overhead and are unlabeled.

Task *ppc2* on the processor cannot be further slowed down because the processor and the RF modem must be co-active. If the processor is greedily slowed down, it will violate max power constraint during the interval 500 - 560 when the hazard detector is on. The tasks *drv1*, *haz1* and *str1* are not overlapped due to the system requirement specified by mode dependency graph. The steering motor and scientific device need significant time to pre-heat, which is adequately considered (the light gray areas in their tracks). Idle interval between task *rf1* and *rf2* on RF modem is set to **receive only** mode rather than **off** mode because the timing overhead of mode changes (power-down, power-up and pre-heating) is larger than the length of the interval. Idle interval before *rf1* is set to **receive only** mode for the same reason. The idle interval between tasks *ppc2* and *ppc3* on the microprocessor is set to **doze,500** mode rather than **sleep** mode in order to meet the min-power constraint.

Scenario	Task sequence	$Cost_{simple}$	$Cost_{greedy}$	$Cost_{modesel}$	percentage(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%

Figure 14: Comparison among different working scenarios. Mission tasks: CAM: shoot pictures; MOV: walk to another location; SCI: perform scientific experiments. Approaches: simple: assume two modes; greedy: greedily voltage/clock scaling; modesel: our mode selection algorithm.

Figure ??(b) shows the result when the min-power constraint is set to zero. This allows the microprocessor to select **sleep** mode during the idle interval between task *ppc2* and *ppc3*, allows other devices to select **off** mode during most of other idle intervals.

The availability of multiple modes of resources gives us opportunities to find better solutions. Our algorithm utilizes multiple modes and can effectively search mode schedules under system constraints. We compared our mode selection algorithm with two other approaches: approach one never utilizes multiple modes, but assumes only two modes, **on** and **off** (for a processor, they are **full-speed** mode and **off** mode.); approach two greedily applies voltage/clock scaling technique whenever possible (we allow power constraint violation in this approach). The results are shown in Figure ?. Three different application scenarios are tested. Data in the first three columns are the results from approach one, approach two and our approach, respectively. It is easy to see that 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/clock scaling technique, but its greediness pays the cost since its saving by slowing down the processor is compensated by extra energy consumed on the RF modem. Note that in all the three scenarios, approach two violates maximum power constraint. Our algorithm gives best results because we utilized multiple modes of resources and apply voltage/clock scaling on the processor. At the same time, we avoid extra energy cost on RF modem by identifying co-activation dependency between the two resources and performing mode selection to find the feasible solution. The reason the energy savings from our approach look not so significant (from 6.4% to 10.5%, average 7.8%) is because the energy savings mainly come from the processor while its power consumption takes only roughly 30% of whole system power consumption.

Given different working scenarios, our algorithm allows the designer to explore multiple design choices and easily find a better solution. Figure ?? summarizes three different working scenarios with the same temperature profile (not shown) from a Mars noon to the afternoon. The temperature in the afternoon is much lower than that at noon causing mechanical devices dissipate more energy to accomplish the same amount of work. With feasible mode schedules found by our algorithm and generated power profile and total energy information, the designer can make better high-level scheduling/planning decisions.

## 6 Conclusions

This paper presents a method for capturing mode dependency and an algorithm for mode selection in power-aware embedded systems. It is critical for all power management techniques to consider dependency between multiple components at the system level, because many previous techniques that assume a single-processor architecture fail to generalize to multiple processors, multiple peripheral devices, or more sophisticated sets of available power modes. It is equally important that system-level power management be driven by timing constraints and power constraints, rather than being hardwired to specific goals.

The mode dependency graph (MDG) 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. In practice, the number of mode combinations generated with our MDG will grow additively rather than multiplicatively among the number of modes in the components. This will be extremely valuable even if mode selection was performed manually or with an alternative algorithm.

We also present a mode selection algorithm that takes advantage of the MDG. By considering power/timing constraints and overhead on transition, this technique will give 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.

## References

- [1] H. Stone. Mars pathfinder microover: A low-cost, low-power spacecraft. In *Proc. the 1996 AIAA Forum on Advanced Developments in Space Robotics*, August 1996.
- [2] A. Chandrakasan, S. Sheng, and R. Brodersen. Low-power cmos digital desig. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, April 1992.
- [3] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of Design Automation Conference*, pages 134–139, 1999.
- [4] W. Namgoong, M. Yu, and T. Meng. A high-efficiency variable-voltage cmos dynamic dc-dc switching regulator. In *IEEE International Solid-State Circuits Conference*, pages 380–381, 1997.
- [5] I. Hong, D. Kirovski, G. Qi, M. Potkonjak, and M. B. Srivastava. Power optimization of variable voltage core-based systems. In *Proceedings of Design Automation Conference*, pages 176–181, 1998.
- [6] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *International Conference on Computer-Aided Design*, pages 365–368, 2000.
- [7] G. Quan and X. S. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of Design Automation Conference*, pages 828–833, 2001.
- [8] J. Luo and N. K. Jha. Power conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In *Proc. Int. Conf. Computer-Aided Design*, pages 357–364, 2000.
- [9] J. Luo and N. K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. In *Proceedings of Design Automation Conference*, pages 444–449, 2001.
- [10] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, June 2000.
- [11] M. Srivastava, A. Chandrakasan, and R. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Trans. VLSI Syst.*, 4(1):42–55, March 1996.
- [12] C.-H. Hwang and A. Wu. A predictive system shutdown method for energy saving of event-driven computation. In *Proc. 1997 Design Automation Conference*, November 1997.
- [13] 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.
- [14] Q. Qiu, Q. Wu, and M. Pedram. Stochastic modeling of a power-managed system construction and optimization. In *Proc. 1999 International Symposium on Low Power Electronics and Design*, pages 194–199, August 1999.
- [15] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli. Dynamic voltage scaling and power management for portable systems. In *Proceedings of the Design Automation Conference*, pages 524–529, June 2001.
- [16] Q. Qiu, Q. Wu, and M. Pedram. Dynamic power management of complex systems using generalized stochastic petri nets. In *Proc. 2000 Design Automation Conference*, pages 352–356, 2000.
- [17] D. Ziegenbein, K. Richter, R. Ernst, J. Teich, and L. Thiele. Representation of process mode correlation for scheduling. In *Proceedings of International Conference on Computer Aided Design*, pages 54–61, November 1998.
- [18] A. Sinha and A. Chandrakasan. Operating system and algorithmic techniques for energy scalable wireless sensor networks. In *Proceedings of the 2nd International Conference on Mobile Data Management*, January 2001.
- [19] P. Chou and G. Borriello. Software scheduling in the co-synthesis of reactive real-time systems. In *Proceedings of the Design Automation Conference*, pages 1–4, June 1994.

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

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

Dept. of Electrical & Computer Engineering  
University of California at Irvine

Category: 1) Design: low power  
2) Design: system level

Format: Conference paper

Contact: Dexin Li  
305 Engineering Tower  
Irvine, CA 92697-2625  
phone: (949) 824-1421  
fax: (949) 824-3203  
email: dlicece.uci.edu

Estimated # of pages: 5

Keywords: power-aware design, mode dependency, embedded systems architecture, mode selection

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

Among the many techniques for system-level power management, it is not currently possible to guarantee timing constraints and have a comprehensive system model at the same time. Specifically, dynamic power management (DPM) techniques can model systems consisting of multiple devices with multiple mode settings. However, if hard real-time constraints are required, then DPM techniques are usually not applicable, because they are based on prediction or timeout and are designed for applications without hard constraints. Instead, low-power scheduling (LPS) techniques may be used to satisfy timing constraints, but they assume a single processor with voltage or frequency scaling. These greedy schedulers are not generalizable to multiple processors or devices with more general modes.

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.