Dynamic Power Management of Electronic Circuits and Systems

Luca Benini Giovanni De Micheli

Stanford University
Stanford, CA 94305

Abstract— Dynamic power management is a design methodology aiming at controlling performance and power levels of digital circuits and systems, with the goal of extending the autonomous operation time of battery-powered systems, providing graceful performance degradation when supply energy is limited, and adapting power dissipation to satisfy environmental constraints.

We present different approaches to power management, and we discuss issues related to the design of computer-aided design tools for power management. In particular, we first review techniques applicable to control-units, such as clock-gating, pre-computation, and partitioning. Next we report on power management techniques used in conjunction with high-level synthesis and applicable to data path and control units. Last we consider system-level power management, describe the "Advanced Configuration and Power Interface" standard, and some of the research challenges in designing computer-aided design algorithms and tools in this area.

I. Introduction

Interest in techniques for designing integrated circuits and systems with low-power consumption has been steadly growing. Research in this field is fueled by two major markets: portable electronics and high-performance systems. In the former case, low-power circuits are needed to provide a reasonable operation time to battery-operated devices. In the latter case, environmental factors, such as heat dissipation, may pose a practical limitation to the use of high-performance processors if power consumption is not controlled and bounded.

Low-power consumption in integrated circuits and systems can be achieved through the combination of different techniques, including architectural design choices [8], logic and physical design [14, 16], choice of circuit families and implementation technology [18]. Most power saving is achieved by ingenious architectural organization, even though computer-aided design (CAD) techniques have been shown to be effective in synthesizing low-power circuits.

We review design techniques and synthesis algorithms for circuits and systems with reduced power consumption, with specific emphasis on *dynamic power management* techniques. We view dynamic power management as a design methodology to control performance and power consumption that exploits idleness in circuit and system's components. We believe that dynamic power management can yield the largest power savings when applied to the system architectural design as well as to the detailed logic design of its hardware components.

Design techniques and computer-aided design solutions for power management are described extensively in [2]. All approaches are based on the principle of exploiting idleness of circuits, systems, or portions thereof. They involve both detection of idle conditions and the freezing of power-consuming activities in the idle components.

Sequential circuits, such as control-units, display often a large degree of idleness due to their reactive nature. Techniques based on clock gating and/or on partitioning have been shown to be successful in reducing power consumption. These methods have been extended to cope with power management in networks representing either data path or control. The power reduction solutions merge seamlessly with circuit synthesis, which is used routinely for circuit design.

Recent initiatives to handle system-level power management include Microsoft's *OnNow* initiative [13] and the *Advanced Configuration and Power Interface* (ACPI) standard proposed by Intel, Microsoft and Toshiba [12]. These approaches migrate power management to the software layer running on hardware platforms, thus providing a flexible and self-configurable solution to adapting the power/performance tradeoff to the needs of mobile (and fixed) and computing and communication.

Despite the effort in standardizing the interface, little work has been done, to the best of our knowledge, to design algorithms and tools for optimal control of power-managed system. We will conclude this review by highlighting the major issues in modeling and designing power-management schemes.

II. Idleness and shutdown mechanisms

The basic principle of a dynamic power manager is to detect inactivity of a unit and shut it down. A fundamental premise is that the idleness detection and power management circuit consumes a negligible fraction of the total power.

We classify idleness as *external* or *internal*. The former is strongly tight to the concept of observability of a unit's outputs, while the latter can be related to the notion of internal state, when the unit has one.

A circuit is externally idle if its outputs are not observed during a period of time. During such period, the unit is functionally redundant and can be shut down, thus reducing power consumption. A unit is internally idle, when it produces the same output over a period of time. Thus, the outputs can be latched and the unit shut down. Internal idleness can be caused by a stationary input stream, or by the combined effect of input conditions applied when the unit is in particular internal state.

While external idleness is a general concept applicable to electro-mechanical (e.g., disks) and electro-optical (e.g., displays) devices, internal idleness is typical of digital circuits.

There are several mechanisms for shutting down a unit. Their merit is related to the power savings during shutdown and to the time required to shut down and restore a unit.

A simple method, commonly used in digital design, is disabling registers by lowering the enable input. By freezing the information on registers, data propagation through combinational logic is halted, with a corresponding power savings. (This saving may be significant in CMOS static technologies, where power is consumed mainly during transitions).

An extension of this method is gating the clock, i.e., by providing registers with a qualified clock which is held at logic zero when a disactivation signal (detecting idleness) is present. Clock-gating provides superior power savings as compared to schemes based on enabled-registers, because registers are not clocked and do not dissipate power, and the clock distribution networks dissipate less power too. Note that both enabled registers and gated clocks allow a circuit to restart operation at the cycle ending the idleness period.

A radical approach to shutdown is to turn off power to a unit. While this mechanism is conceptually simple and applicable in general, it usually involves a non-negligible time to restore operation. Thus this scheme is better applicable to system components (e.g., memories, disks, displays). Some components can be shut down at different levels, each one corresponding to a power consumption level and to a delay to restore operation. For example, a disk [19] may have an operational state, in addition to an idle, a low-power idle, a standby, and a sleep state. In the idle states the disk is spinning, but the some of the electronic components of the drive are turned off. The transition from idle to active is extremely fast, but only 50-70% of the power is saved in these states. In the standby and sleep states, the disk is spun down, thus reducing power consumption by 90-95%. On the other hand, the transition to the active state is not only slow, but in causes additional power consumption.

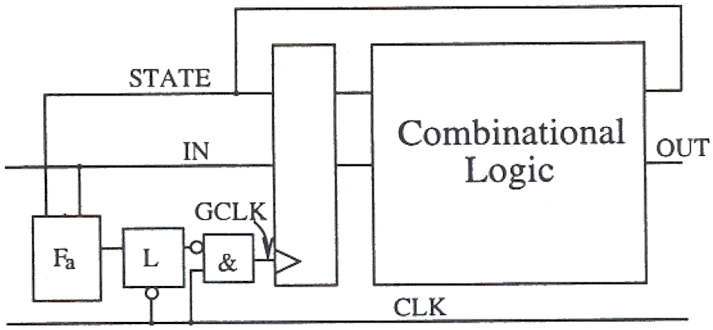

Fig. 1. Gated-clock architecture

III. Power management for functional blocks

In this section, we consider power management for functional units within VLSI circuits, and we concentrate on the fundamental mechanisms and on CAD related issues.

A. Gated Clocks

While the idea of clock-gating has been known for a while, the first set of CAD tools for designing gated clock circuits was proposed by Benini *et al.* in [4, 6]. The method is applicable to control units, modeled as finite-state machines (FSMs). While the original method [4] required an explicit FSM description (e.g., state table), this technique has been extended to cope also with networks [5].

The circuit is augmented by an activation block (Figure 1), which issues a signal that selectively stops the clock, thus preventing power dissipation in the combinational logic and in the registers. (This signal is latched to avoid spurious transitions.)

Clock gating exploits internal idleness. The activation block is synthesized automatically, using the knowledge of the FSM transition table. Next, its implementation is optimized with the goal of reducing its own power dissipation, while preserving idleness detection in a predefined fraction of idle conditions. Power savings in control circuit are in the order of 30%, even though larger savings are possible on reactive circuits, such as most logic controllers employed in telecommunication systems.

A technical difficulty in designing circuits with gated clocks is due to the lack of explicit FSM representations (or the inability to extract them) for circuits with 20 registers or more, because of the exponential growth of the number of states. This problem is overcome by using implicit methods [5] that can extract the activation circuit from a network description of the controller. With this extension, more complex controllers can be handled.

Kitihara *et al.* [11] developed CAD tools for gated clock design that have been successfully used in an industrial environment.

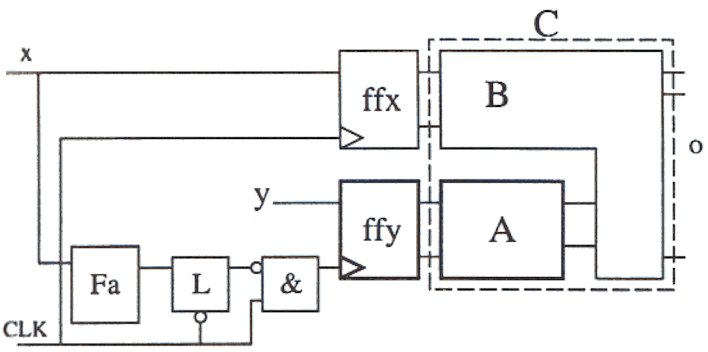

Fig. 2. Pre-computation architecture

B. Pre-computation

The *pre-computation* approach to power management is due to Alidina and co-workers [1]. The method relies on the idea of duplicating part of the logic with the purpose of pre-computing the circuit output values one clock cycle before they are required, and then use these values to reduce the total amount of switching in the circuit during the following clock cycle (See Figure 2.) By knowing the output values one clock cycle in advance, the original logic can be turned off during the next time frame, thus eliminating any switching of internal capacitances.

Pre-computation exploits the external idleness of a circuit. The use of pre-computation-based architectures has shown to be particularly effective for the optimization of pipelined circuits, whose structure consists of a combinational logic block with latched inputs and outputs. Comparisons between the pre-computation and the gated-clock approach are presented in a companion paper [7].

C. Control-unit partitioning

A control unit may be implemented as a collection of interacting sub-units, each executing a portion of the overall control function. The advantage of using a partitioned controller versus a monolithic unit is that each sub-unit can be selectively clocked. In other words, the clock of the idle sub-units can be halted, with a corresponding power saving. Usually only one sub-unit is active at a given time. When the corresponding control function terminates the unit sends a start message to another sub-unit and then disables itself.

Since partitioning the control-unit implementation leads to an area and interconnect overhead, the granularity of the the partition affects significantly the overall results. Coarse-grained partitions are preferable because they reduce the communication overheads among the sub-units.

The search for a good partition of a control unit can be reduced to a power-oriented decomposition of the corresponding FSM models. Such a decomposition can be performed using the information of the sequential behavior of

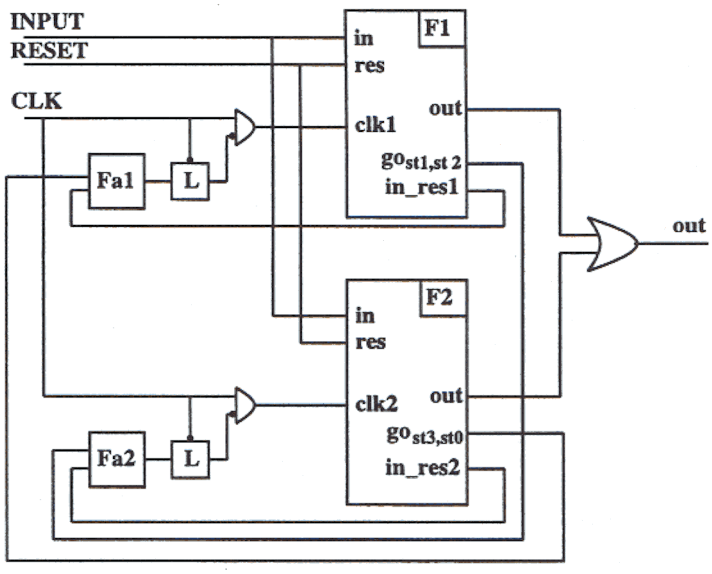

Fig. 3. Partitioned control unit

the control unit [2] (i.e., the state table of the controller) or by analyzing the control-flow of behavioral models of the overall circuits [3]. In the latter case, the control flow analysis can reveal mutually-exclusive sections of the circuit operation (due to serialization or branching) which can be implemented by sub-units with no concurrent execution.

D. Power management in high-level synthesis

Several power management techniques, that operate in conjunction with high-level synthesis, have been recently proposed. Their appeal stems from the potential power savings, which is larger at the high-level of abstraction as compared to the logic level, because of the granularity of the objects being power managed. Nevertheless, difficulties come from the accuracy of power consumption information in behavioral models, as well as from the disuniform acceptance of high-level synthesis methodologies. (E.g., the scope of synthesis varies from one high-level synthesis tool to another. Some tools operate on RTL descriptions and some on behavioral models restricted by synthesis policies). We will give some illustrative examples.

Theeuven and Seelen [20] proposed a method that searches RTL models for internal idleness. They introduced a technique to identify registers that are in hold mode for a large fraction of the operation time. For these registers, a hold expression is computed, which is used to synthesize a power management circuit that controls their clock.

External idleness on data-path busses can be efficiently detected in RTL descriptions. Reducing switching activity on busses is important, because of the usually large capacitance being switched. At this level of abstraction, we can classify data path modules as computational units

We classify idleness as *external* or *internal*. The former is strongly tight to the concept of observability of a unit's outputs, while the latter can be related to the notion of internal state, when the unit has one.

A circuit is externally idle if its outputs are not observed during a period of time. During such period, the unit is functionally redundant and can be shut down, thus reducing power consumption. A unit is internally idle, when it produces the same output over a period of time. Thus, the outputs can be latched and the unit shut down. Internal idleness can be caused by a stationary input stream, or by the combined effect of input conditions applied when the unit is in particular internal state.

While external idleness is a general concept applicable to electro-mechanical (e.g., disks) and electro-optical (e.g., displays) devices, internal idleness is typical of digital circuits.

There are several mechanisms for shutting down a unit. Their merit is related to the power savings during shutdown and to the time required to shut down and restore a unit.

A simple method, commonly used in digital design, is disabling registers by lowering the enable input. By freezing the information on registers, data propagation through combinational logic is halted, with a corresponding power savings. (This saving may be significant in CMOS static technologies, where power is consumed mainly during transitions).

An extension of this method is gating the clock, i.e., by providing registers with a qualified clock which is held at logic zero when a disactivation signal (detecting idleness) is present. Clock-gating provides superior power savings as compared to schemes based on enabled-registers, because registers are not clocked and do not dissipate power, and the clock distribution networks dissipate less power too. Note that both enabled registers and gated clocks allow a circuit to restart operation at the cycle ending the idleness period.

A radical approach to shutdown is to turn off power to a unit. While this mechanism is conceptually simple and applicable in general, it usually involves a non-negligible time to restore operation. Thus this scheme is better applicable to system components (e.g., memories, disks, displays). Some components can be shut down at different levels, each one corresponding to a power consumption level and to a delay to restore operation. For example, a disk [19] may have an operational state, in addition to an idle, a low-power idle, a standby, and a sleep state. In the idle states the disk is spinning, but the some of the electronic components of the drive are turned off. The transition from idle to active is extremely fast, but only 50-70% of the power is saved in these states. In the standby and sleep states, the disk is spun down, thus reducing power consumption by 90-95%. On the other hand, the transition to the active state is not only slow, but in causes additional power consumption.

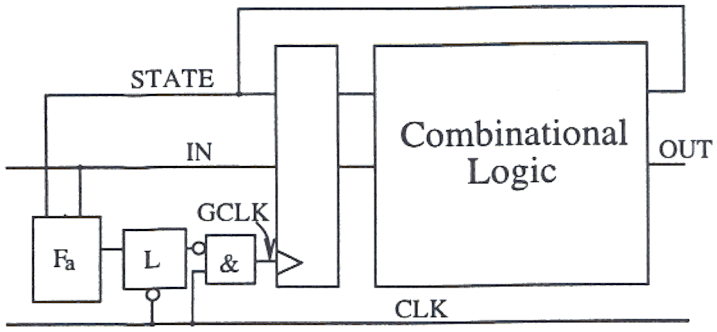

Fig. Gated-clock architecture

III. Power management for functional blocks

In this section, we consider power management for functional units within VLSI circuits, and we concentrate on the fundamental mechanisms and on CAD related issues.

A. Gated Clocks

While the idea of clock-gating has been known for a while, the first set of CAD tools for designing gated clock circuits was proposed by Benini *et al.* in [4, 6]. The method is applicable to control units, modeled as finite-state machines (FSMs). While the original method [4] required an explicit FSM description (e.g., state table), this technique has been extended to cope also with networks [5].

The circuit is augmented by an activation block (Figure 1), which issues a signal that selectively stops the clock, thus preventing power dissipation in the combinational logic and in the registers. (This signal is latched to avoid spurious transitions.)

Clock gating exploits internal idleness. The activation block is synthesized automatically, using the knowledge of the FSM transition table. Next, its implementation is optimized with the goal of reducing its own power dissipation, while preserving idleness detection in a predefined fraction of idle conditions. Power savings in control circuit are in the order of 30%, even though larger savings are possible on reactive circuits, such as most logic controllers employed in telecommunication systems.

A technical difficulty in designing circuits with gated clocks is due to the lack of explicit FSM representations (or the inability to extract them) for circuits with 20 registers or more, because of the exponential growth of the number of states. This problem is overcome by using implicit methods [5] that can extract the activation circuit from a network description of the controller. With this extension, more complex controllers can be handled.

Kitihara *et al.* [11] developed CAD tools for gated clock design that have been successfully used in an industrial environment.

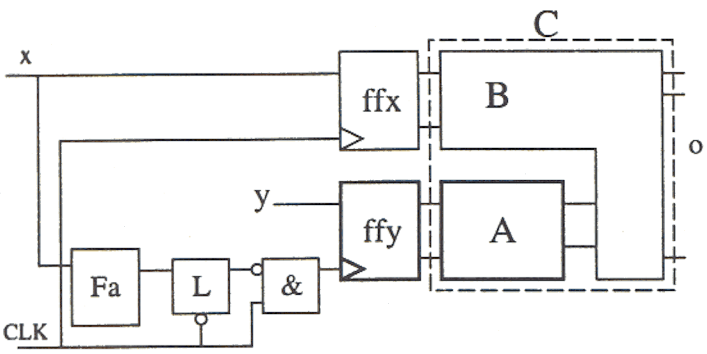

Fig. 2. Pre-computation architecture

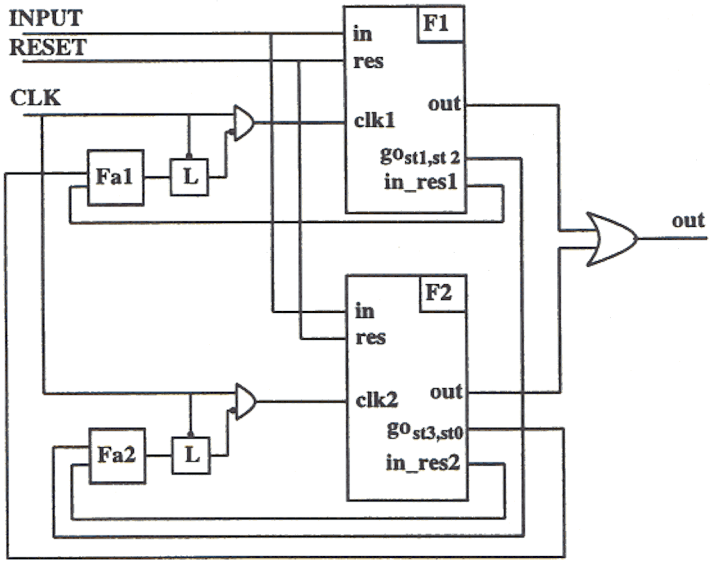

Fig. 3. Partitioned control unit

B. Pre-computation

The *pre-computation* approach to power management is due to Alidina and co-workers [1]. The method relies on the idea of duplicating part of the logic with the purpose of pre-computing the circuit output values one clock cycle before they are required, and then use these values to reduce the total amount of switching in the circuit during the following clock cycle (See Figure 2.) By knowing the output values one clock cycle in advance, the original logic can be turned off during the next time frame, thus eliminating any switching of internal capacitances.

Pre-computation exploits the external idleness of a circuit. The use of pre-computation-based architectures has shown to be particularly effective for the optimization of pipelined circuits, whose structure consists of a combinational logic block with latched inputs and outputs. Comparisons between the pre-computation and the gated-clock approach are presented in a companion paper [7].

C. Control-unit partitioning

A control unit may be implemented as a collection of interacting sub-units, each executing a portion of the overall control function. The advantage of using a partitioned controller versus a monolithic unit is that each sub-unit can be selectively clocked. In other words, the clock of the idle sub-units can be halted, with a corresponding power saving. Usually only one sub-unit is active at a given time. When the corresponding control function terminates the unit sends a start message to another sub-unit and then disables itself.

Since partitioning the control-unit implementation leads to an area and interconnect overhead, the granularity of the the partition affects significantly the overall results. Coarse-grained partitions are preferable because they reduce the communication overheads among the sub-units.

The search for a good partition of a control unit can be reduced to a power-oriented decomposition of the corresponding FSM models. Such a decomposition can be performed using the information of the sequential behavior of the control unit [2] (i.e., the state table of the controller) or by analyzing the control-flow of behavioral models of the overall circuits [3]. In the latter case, the control flow analysis can reveal mutually-exclusive sections of the circuit operation (due to serialization or branching) which can be implemented by sub-units with no concurrent execution.

D. Power management in high-level synthesis

Several power management techniques, that operate in conjunction with high-level synthesis, have been recently proposed. Their appeal stems from the potential power savings, which is larger at the high-level of abstraction as compared to the logic level, because of the granularity of the objects being power managed. Nevertheless, difficulties come from the accuracy of power consumption information in behavioral models, as well as from the disuniform acceptance of high-level synthesis methodologies. (E.g., the scope of synthesis varies from one high-level synthesis tool to another. Some tools operate on RTL descriptions and some on behavioral models restricted by synthesis policies). We will give some illustrative examples.

Theeuven and Seelen [20] proposed a method that searches RTL models for internal idleness. They introduced a technique to identify registers that are in hold mode for a large fraction of the operation time. For these registers, a hold expression is computed, which is used to synthesize a power management circuit that controls their clock.

External idleness on data-path busses can be efficiently detected in RTL descriptions. Reducing switching activity on busses is important, because of the usually large capacitance being switched. At this level of abstraction, we can classify data path modules as computational units

and steering modules. Computational units are arithmetic and logic circuits, while steering modules are multiplexers, registers and three-state drivers. Kapadia [10] proposed a technique based on monitoring the observability of data-path busses, and by freezing their values by controlling the steering modules.

Other power management techniques exploit the ability of high-level synthesis of mapping behavioral descriptions into circuits. *Operand isolation* is a power recovery technique that exploits external idleness. It considers data-path units which do not produce observed results in some steps of the schedule. The inputs to such units can be frozen, by adding appropriate latches and by using a corresponding control circuit, with the goal of eliminating switching activity when the unit is idle. Note that it is straightforward to extract idle conditions from a schedule. Whereas operand isolation is used in manual design practice, a computer-aided tool for using this idea was presented first by Raghunathan *et al* [17] based on controller specification and steering logic restructuring.

Memory segmentation [9] is a scheme that reduces power by exposing idleness in memory accesses. By partitioning a memory into segments with independent clock and refresh signals, idle segments can be put in sleep mode (i.e., their clock can be stopped), or their refresh signal can be shut down, thereby minimizing their power dissipation. The partition of the memory can be driven by information of memory usage, available in high-level synthesis.

Finally, Monteiro and Devadas [15] proposed a scheduling technique where branching conditions are evaluated early in the schedule, so that power can be saved by operand isolation of the unit in the inactive branch. Although results show power reduction of 30% for a few benchmark circuits, the applicability of this technique is limited by the potential performance loss.

IV. Power management for systems

In this section we consider power management for electronic systems, that contain a plurality of chips as well as other components of different nature. Analog, electromechanical and optical components contribute to a significant fraction of the overall power budget. For example, the power breakdown for a well-known laptop computer [21] shows that, on average 36% of the total power is consumed by the display, 18% by the hard drive, 18% by the wireless LAN interface, 7% by non-critical components (keyboard, mouse etc.), and only 21% by digital VLSI circuitry (mainly memory and CPU). Reducing the power in the digital components of this laptop by *10X* would reduce the overall power consumption by less than 19%.

Dynamic power management is successfully used by system designers, but the manual design of complex systems that support dynamic power management is a difficult, long and error-prone task. Unfortunately, system-level computer-aided design environments and tools are still in their infancy, and EDA vendors are lagging far behind the needs of this segment of the electronic industry.

To compensate for this lack of support, several system developers and vendors [12, 13] are aggressively pursuing a long-term, wide-scope strategy to greatly simplify the task of designing large and complex power-managed systems. The strategy is based on a standardization initiative known as the *advanced configuration and power interface* (ACPI), described in the next section. Although the initiative targets *personal computers* (PCs), it contains useful guidelines for a more general class of systems. The characterizing feature of ACPI is that it recognizes dynamic power management as the key to reducing overall system power consumption, and it focuses on making the implementation of dynamic power management schemes in personal computers as straightforward as possible.

The ACPI specification forms the foundation of the *OnNow initiative* launched by the Microsoft Corporation. The *OnNow* initiative is specific to the design of personal computers (PCs) and proposes the migration of power management algorithms and policies into the computer's operating system (OS). An *OnNow*-compliant PC platform must conform to a set of requirements requirements [13], including: i) the PC is ready for use as soon as the user turns it on; ii) the PC appears as off when not in use, but it must be capable of responding to wake-up events; iii) software tracks hardware status changes and adjusts accordingly; iv) all hardware devices participate in the power management scheme.

Present personal computers do not meet the requirements of *OnNow* yet. Nevertheless, the migration of power management to the operating system level will yield a profound improvement of the performance, power consumption and quality of service of personal computers, because it will give the control of the system to the component (i.e., OS) that can make the most informative decisions. *OnNow* relies on the ACPI infrastructure to interface the software to the hardware components to be managed.

A. ACPI

ACPI [12] is an OS-independent, general specification that applies to desktop, mobile and home computers as well as to high-performance servers. The specification has emerged as an evolution of previous initiatives that attempted to integrate power management features in the low-level routines that directly interact with hardware devices (firmware and BIOS). It also provides some form of *backward compatibility* since it allows ACPI-compliant hardware resources to co-exist with legacy non-ACPI-compliant hardware.

ACPI is the key element for implementing *operating system power management* (OSPM) strategies, such as *OnNow*. It is an open standard that is made available

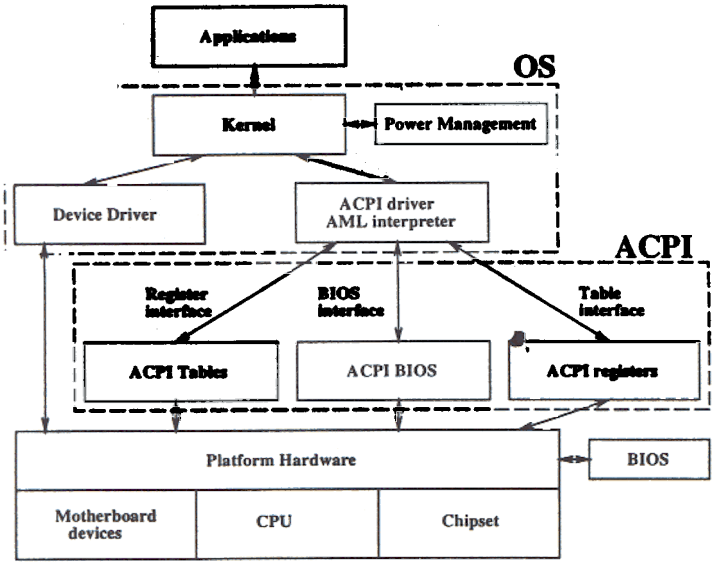

Fig. 4. ACPI interface and PC platform

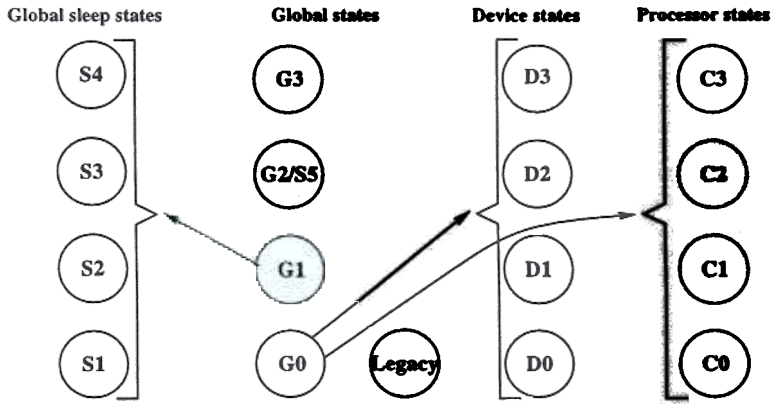

Fig. 5. State definitions for ACPI

for adoption by hardware vendors and operating system developers. The main goals of ACPI are to: i) enable all PCs to implement motherboard dynamic configuration and power management; ii) enhance power management features and the robustness of power managed systems; iii) accelerate implementation of power-managed computers, reduce costs and time to market.

The ACPI specification defines the interfaces between OS software and hardware. The software and hardware components relevant to ACPI are shown in Figure 4. Applications interact with the OS kernel through *application programming interfaces* (APIs). A module of the OS implements the power management policies, as discussed in the previous section. The power management module interacts with the hardware through kernel services (system calls). The kernel interacts with the hardware using device drivers. The front-end of the ACPI interface is the *ACPI driver*. The driver is OS-specific, it maps kernel requests to ACPI commands, and ACPI responses/messages to kernel signals/interrupts. Notice that the kernel may also interact with non-ACPI-compliant hardware through other device drivers.

At the bottom of Figure 4 the hardware platform is shown. Although it is represented as a monolithic block, it is useful to distinguish three types of hardware components. First, hardware resources (or *devices*) are the system components that provide some kind of specialized functionality (e.g., video controllers, modems, bus controllers). Second, the *CPU* can be seen as a specialized resource that need to be active for the OS (and the ACPI interface layer) to run. Finally, the *chipset* (also called core logic) is the motherboard logic that controls the most basic hardware functionalities (such as real-time clocks, interrupt signals, processor busses) and interfaces the CPU with all other devices. Although the CPU runs the OS, no system activity could be performed without the chipset. From the power management standpoint, the chipset, or a critical part of it, should always be active, because the system relies on it to exit from sleep states. Hence, ACPI does not define power management strategies for the chipset itself. The power consumption of the chipset should be managed at the firmware level, transparently to the OS.

It is important to notice that ACPI specifies neither how to implement hardware devices nor how to realize power management in the operating system. No constraints are imposed on implementation styles for hardware and on power management policies. Implementation of ACPI-compliant hardware can leverage any technology or architectural optimization as long as the power-managed device is controllable by the standard interface specified by ACPI.

In ACPI, the hardware is seen as a monolithic system, with five *global power states*. Namely:

- *Mechanical off* state $G3$, with no power consumption.
- *Soft off* state $G2$ (also called $S5$). A full OS reboot is needed to restore the working state.
- *Sleeping* state $G1$. The system appears to be off and power consumption is reduced. The system returns to the working state in an amount of time which grows with the inverse of the power consumption.
- *Working* state $G0$, where the system is ON and fully usable.
- *Legacy* state, which is entered when the system does not comply with ACPI.

The global states are shown in Figure 5. They are ordered from top to bottom by increasing power dissipation.

The ACPI specification refines the classification of global system states by defining four sleeping states within state $G1$, as shown in Figure 5:

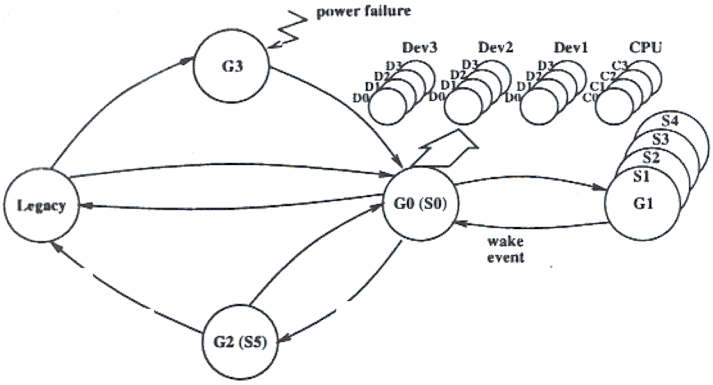

Fig. 6. Global and power states and substates

- $S1$ is a sleeping state with low wake-up latency. No system context is lost in the CPU or the chipset.
- $S2$ is a low wake-up latency sleeping state. This state is similar to the $S1$ sleeping state with the exception that the CPU and system cache context is lost.
- $S3$ is another low wake-up latency sleeping state where all system context is lost except system memory.
- $S4$ is the sleeping state with the lowest power and longest wake-up latency. To reduce power to a minimum, all devices are powered off.

Additionally, the ACPI specification defines states for system components. There are two types of system components, *devices* and *processor*, for which power states are specified. Devices are abstract representations of the hardware resources in the system. (See Figure 5). The processor is the central processing unit that controls the entire PC platform. Special devices are *embedded controllers*, that function as resources for the main CPU.

ACPI defines a specialized interface for embedded controllers. Although from a power management point of view embedded controllers are treated as normal resources, they have specialized drivers because they may be used to monitor power-related system characteristics, perform low-level complex calculations, and they may provide data that is required to implement power management policies. For example, an embedded controller can be used to control board temperature sensors and provide valuable data for thermal management.

States and transitions for an ACPI-compliant system are shown in Figure 6. Usually the system alternates between the working ($G0$) and the sleeping ($G1$) states. When the entire system is idle or the user has pressed the power-off button, the OS will drive the computer into one of the states on the left side of Figure 6. From the user's viewpoint, no computation occurs. The sleeping sub-states differ in which *wake* events can force a transition into a working state, and how long the transition should take. If the only wake-up event of interest is the activation of the user turn-on button and a latency of a few minutes can be tolerated, the OS could save the entire system context into non-volatile storage and transition the hardware into a soft-off state ($G2$). In this state, power dissipation is almost null and context is retained (in non-volatile memory) for an arbitrary period of time. The mechanical off state ($G3$) is entered in the case of power failure or mechanical disconnection of power supply. Complete OS boot is required to exit the mechanical off state. Finally, the *legacy* state is entered in case the hardware does not support OSPM.

It is important to note that ACPI provides only a framework for designers to implement power management strategies, while the the choice of power management *policy* is left to the engineer. This leaves important research problems open. Namely the study of very high-level performance/power models, the design of policies for dynamic control, and their validation. These problems are important subjects of research for developing the corresponding computer-aided design tools.

B. Modeling, design and validation

We consider now the power management problem from the perspective of a tool developer who is interested in algorithms and tools for designing power management schemes and validating them.

In essence, we need to abstract the ACPI scheme as a finite-state system, where each state is associated with a specific power and performance level. Moreover, transitions among states have a cost in term of power and performance as well. Consider for example spinning up a hard disk. The disk acceleration requires a peak of electrical power, and the data will be available from the disk only after it has settled to an operational speed.

System-level design requires deriving power and performance measures from abstract models, with limited information about the internals. The difficulty in achieving precise measures is compounded by the complexity of the interactions of the system's component, which makes some values uncertain, despite the accuracy of the model being used. The uncertainty about performance and power data can be modeled by considering average values for state and transitions, as well as their statistical distributions [2].

Another source of uncertainty is the behavior of the user, which cannot be modeled deterministically. Nevertheless the model of the user plays a significant role in evaluating the power/performance behavior. Consider for example two users of two identical laptop computers: their activities may require different power profiles because of the applications being used.

Stochastic models of systems and users may serve two purposes. First, simulating the overall behavior of a product in action, with the goal of validating its performance/power levels. Second, deriving appropriate power

management schemes. Whereas such schemes may be arbitrarily complex, we focus our attention on a simple yet effective abstraction which we call policy. A policy is a sequence of decisions about the state transitions based on the previous history of the system activity. A simple example of a policy is to determine the sequence in time of commands to spin up or down a hard disk. Note that the finite-state system abstraction, as supported by ACPI, is a prerequisite for the definition and computation of a policy.

In engineering practice, *eager* policies have been used as well as policies based on timeouts. An eager policy shuts off a unit as soon as it is not used. When timeouts are used, resources are kept in the operational state for a prescribed fixed time after becoming idle. Needless to say, these strategies do not guarantee to minimize the overall power consumption. Therefore an important problem is to determine optimal (or optimum) policies that minimize power consumption subject to performance constraints, or vice versa. While this problem is hard in general, some simplifying assumptions about the system's model may make their solution tractable [2]. It is important to stress that CAD tools for determining optimum/optimal policies would be very useful, because policies need to be determined for different systems and system/user configurations.

Validation techniques, based on stochastic simulators, may be also of great use to validate policies by verifying that the abstract modeling assumptions do not impair the validity of the results. Overall, policy determination, implementation and validation constitute the cornerstones of system-level power management tools [2].

V Conclusions

In this review we have considered power management at different levels of abstraction. First, we considered logic level models of digital circuits, and presented schemes for clock-gating, pre-computation, and partitioning of control units. These techniques have been shown to be practical when applied to control-dominated circuits, or to the control portion of a circuit. Unfortunately, the power consumed by the control unit may be small when compared to the overall circuit dissipation.

Next we considered power management in high-level synthesis, and we commented on operand isolation, memory segmentation and scheduling. These techniques are applicable to both data path and control units, and thus may lead to significant power savings within a digital chip.

Finally we described power management for systems, and we considered operating system power management as proposed by the *OnNow* initiative and supported by the ACPI standard. These approaches are important for system-level design where the digital component consumes only a fraction of the total power.

By enlarging the scope of applicability of power management techniques, we discover larger potential power savings. Nevertheless, the higher the abstraction level is, the harder it is to estimate performance and power. Therefore intersting areas of research are the use of stochastic models for system-level power estimation as well as the development of algorithms and tools for management policies and their validation.

VI Acknowledgements

We acknowledge support from NSF under contract MIP-942119 and from Toshiba Corporation.

References

[1] M. Alidina, J. Monteiro et al., "Precomputation-based sequential logic optimization for low power," *IEEE Transactions on VLSI Systems*, vol. 2, no. 4, pp. 426–436, Jan. 19

[2] L. Benini and G. De Micheli, *Dynamic Power Management of Circuits and Systems: Design Techniques and CAD Tools,* Kluwer, 1997.

[3] L. Benini, P. Vuillod, C. Coelho and G. De Micheli, "Synthesis of low-power selectively-clocked systems from high-level specification," in *IEEE International Symposium on System Synthesis*, pp. 57–62, Oct. 1996.

[4] L. Benini and G. De Micheli, "Automatic synthesis of low-power gated-clock finite-state machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 6, pp. 630–643, June. 1996.

[5] L. Benini, G. De Micheli, E. Macii, M. Poncino and R. Scarsi, "Symbolic techniques for power optimization of large control-oriented synchronous networks," in *IEEE European Design and Test Conference*, pp. 514–520, March 1997.

[6] L. Benini, P. Siegel and G. De Micheli, "Automatic synthesis of gated clocks for power reduction in sequential circuits," *IEEE Design and Test of Computers*, pp. 32–40, Dec. 1994.

[7] L. Benini, G. De Micheli, E. Macii, M. Poncino, R. Scarsi. "Integrating Logic-level Power Management Techniques," *SASIMI*, 1997.

[8] A. Chandrakasan and R. Brodersen, *Low power digital CMOS design.* Kluwer, 1995.

[9] A. H. Farrahi, G. E. Tellez and M. Sarrafzadeh "Memory segmentation to exploit sleep mode operation," *Proceedings of the Design Automation Conference*, pp. 36–41, June. 1995.

[10] H. Kapadia, *Private communication*, 1997.

[11] T. Kitihara, F. Minami, S. Nishio, M. Murakata and T. Mitsuhashi, "A Clock-Gating Method for Low-Power LSI Design," *Private communication*, 1997.

[12] *http://www.intel.com/ial/powermgm/specs.html*, Intel, Microsoft and Toshiba, "Advanced Configuration and Power Interface specification", Dec. 1996.

[13] *http://www.microsoft.com/hwdev/pcfuture/ONNOW.HTM*, Microsoft, "OnNow: the evolution of the PC platform," Aug. 1997.

[14] J. Monteiro and S. Devadas, *Computer-aided techniques for low power sequential logic circuits.* Kluwer 1997.

[15] J. Monteiro, S. Devadas, P. Ashar and A. Mauskar, "Scheduling techniques to enable power management," in *Proceeding of the Design Automation Conference* pp. 349–352, June 1996.

[16] J. M. Rabaey and M. Pedram (editors), *Low power design methodologies.* Kluwer, 1996.

[17] A. Raghunathan and N. K. Jha, "Behavioral synthesis for low power," in *Proceedings of the International Conference on Computer Design*, pp. 318–322, Oct. 1994.

[18] T. Sakurai and T. Kuroda, "Low-power circuit design for multimedia CMOS VLSI," in *Workshop on Synthesis and System Integration of Mixed Technologies*, pp. 3–10, Nov. 1996.

[19] *http://www.storage.ibm.com/storage/oem/data/travvp.htm* Technical specification of hard-drive IBM Travelstar VP 2.5-inch, 1996.

[20] F. Theeuven and E. Seelen, "Power reduction through clock gating by symbolic manipulation," in *Symposium on Logic and Architecture Design*, pp. 184–191, Dec. 1996.

[21] S. Udani and J. Smith, "The power broker: intelligent power management for mobile computing," *Technical report MS-CIS-96-12*, Dept. of Computer Information Science, University of Pennsylvania, May 1996.