System-level Power Estimation And Optimization

Luca Benini Robin Hodgson Polly Siegel
Hewlett-Packard Laboratories
benini@hpl.hp.com hodgson@hpl.hp.com polly_siegel@hp.com

Most work to date on power reduction has focused at the component level, not at the system level. In this paper, we propose a framework for describing the power behavior of system-level designs. The model consists of a set of resources, an environmental workload specification, and a *power management policy,* which serves as the heart of the system model. We map this model to a simulation-based framework to obtain an estimate of the system's power dissipation. Accompanying this, we propose an algorithm to optimize power management policies. The optimization algorithm can be used in a tight loop with the estimation engine to derive new power-management policy algorithms for a given system-level description. We tested our approach by applying it to a real-life low-power portable design, achieving a power estimation accuracy of ~10%, and a 23% reduction in power after policy optimization.

1 Introduction

Power reduction is taking on increasing importance in electrical designs, in large part because of its contribution to the overall cost of the system. As clock speeds increase, so does power dissipation and accompanying thermal heat dissipation problems. For portable systems, the problems are even more critical. Thermal dissipation is made more difficult by the confined spaces of a portable device. Additionally, the portable usage paradigm depends on long battery life and low weight. Excessive power consumption results in compromises to one or both of these parameters.

Our work focuses on power reduction at a very high level of abstraction—the system level. Specifically, we look at optimizing system-level *dynamic power management policies* as a way to reduce power. In dynamic power management we exploit the knowledge that complex systems are almost never fully utilized. Because most systems are designed to perform multiple functions, much of the system's hardware is not required at a given point in time. Many systems are designed for peak performance under a maximal workload, which is the only situation where all hardware resources are fully utilized. By dynamically shutting down unused resources during periods of underutilization, idleness can be exploited to save system power [19, 20].

Any power management policy that we derive must consider resource transition penalties, system partitioning, and environmental assumptions. The simplest dynamic power management policy—shutting off unused resources as soon as they become idle—is not always the best, for a number of reasons. Shutting off certain types of resources incurs a performance cost. Many resources, such as disk drives, take time to transition from a fully operational state to the lowest power state and vice versa. In the disk drive example, because of the transition time required to spin a disk down and back up, if the disk drive in a notebook computer were turned off as soon as it became idle, the overall system performance would be compromised because disk drive usage patterns are very bursty [1, 8]. In general, an idle resource should be shut down only if the performance penalty is acceptable and the inactive time will be long enough to amortize the additional power costs incurred in the transition from active to sleep and back again.

Our work addresses both estimation and optimization of system-level power management policies. We propose a simulation-based framework to synthetically describe power managed systems and estimate their power dissipation. Accompanying this, we propose algorithms to optimize power management policies. The optimization algorithms can be used in a tight loop with the estimation engine to derive new power-management policy algorithms for a given system-level partition.

Our approach is general, allowing a large class of systems to be described. Because the model supports high-level abstract system descriptions, estimation and optimization can be performed very early in the design process when design decisions have the biggest impact on power. The system description can be refined as design progresses, providing increasingly accurate and detailed information on how design choices impact power dissipation.

We tested our approach by applying it to a real-life example, a low-power portable device designed in our group. The analysis of a complete case study serves as an important benchmark for our methodology. The availability of a fully-functional portable system enabled accurate validation of the prediction accuracy of the estimates, and gave valuable feedback on our approach.

2 Background

Extensive research has been conducted on power estimation and optimization in the last few years [2, 3, 4]. Some groups have focused on specific application domains such as wireless communication devices, portable multimedia equipment, computers, and instrumentation, and have developed prototypes of low-power devices using a mix of design techniques applied to various levels of abstraction, from the circuit level up to the architectural level. Several efficient and accurate circuit-level and gate-level simulation tools have been developed to estimate power consumption. A few register-transfer-level (RTL) power estimation engines have been proposed as well [9, 10, 11, 12, 13]. All power estimation approaches are limited by the low abstraction level used to represent the target circuits. Accurate power estimates can be obtained at great performance cost, late in the design cycle, only after a detailed design is available. Multi-chip heterogeneous systems are well beyond the limits of these power estimation tools.

Industrial designs are large, complex, heterogeneous systems, often including multiple chips as well as electro-mechanical, optical, and magnetic components. These systems can be effectively represented and analyzed only at a high level of abstraction. Abstraction is required both to manage complexity and to cope with uncertainty. During the initial design of a complex system, many details of the implementation are not defined or are subject to change. Clearly, when raising the abstraction level of the representation, some accuracy is lost.

A few approaches to system-level power estimation have been investigated [5, 6, 7]. Interestingly, these approaches are all based on a *constant additive model*. In the constant additive model, a constant power dissipation value is associated with each component in the design. The total power dissipation is computed by simply summing the power dissipation of all components.

The main limitation of the constant additive approach is that it leaves a large fraction of the estimation process to the designer. In many cases workload estimation is not a straightforward task, and requires deep insight into the system structure and interaction among components. Our modeling methodology tries to overcome this limitation by using a more complex high-level system model where workload computations can be automated by specifying the interaction among components.

3 Power Modeling

In this section we focus on developing an abstract model of the system that allows fast estimation of system-level power for use during the component selection and system partitioning phase of the design process. After explaining the model, we describe the estimation "tool" that we developed to implement the model and results derived from applying the model and tools to a real design.

3.1 System-level power model

By abstracting away all but the power behavior of the system, we can create a model that satisfies our previously mentioned objectives. The model must include information about the system's power behavior, the power behavior of its functional blocks and its block interactions, along with information about the environment, which drives the behavior of the system, as seen in Figure 1.

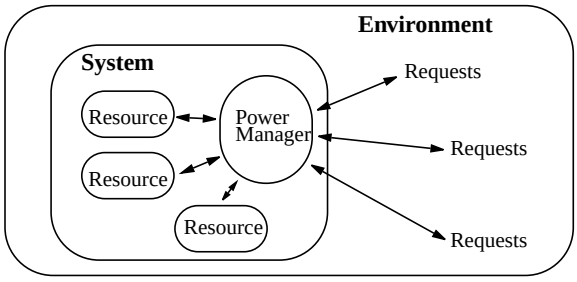

Figure 1: Simplest system-level power model

Each resource is described by a simple state machine, containing only information relevant to its power behavior. At any given point in time, each resource is in a specific power state, which consumes a discrete amount of power. To get the total system power usage at a given point in time just requires summing up the current power values for all resources within the system. From there, average or peak power dissipation can easily be determined by looking at the power usage over time for a given set of environmental conditions.

Because we will be using the model not only to do estimation, but also to optimize the power behavior of the system, we add an entity called the *power manager*, which translates environmental stimuli into requests to system resources to change their power states. The power manager represents an abstract view of the control policies implemented by the system. In actual practice, the power manager may represent a distributed thread of operation within a system's CPU. Or, it may be implemented in a distributed fashion within the bodies of the components themselves. But it is very convenient for both estimation and optimization to abstract away everything but the power state change information into a single abstract entity like the power manager.

In the simple model shown in Figure 1, the structural information consists only of the set of resources instantiated in the design. Information regarding communication between resources is not provided. This high-level model has a *star structure*, where all communication is to and from the power manager. We will stay with this simple model, to illustrate its basic elements.

3.1.1 Resource power state machine

Each resource in the system need only contain enough information to describe its power behavior. In most cases, a resource's power behavior can be represented by a simple state machine, its *power state machine*. At any given point in time, a resource will be in one of its power states, as controlled by inputs from the power manager. In the simplest model, each state in a resource's power state machine has a value associated with it representing the power consumption in that state, where the power consumption may be a designer estimate, a derived value, or a manufacturer's specification in the case where a specific component has been selected.

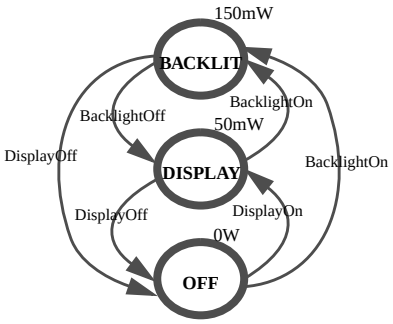

Figure 2: Sample resource power state machine for a display

Figure 2 shows a simple power state machine for a backlit LCD display. The display has three power states: OFF, DISPLAY, and BACKLIT. In the DISPLAY state, only the LCD display is on, consuming 50 mW of power. In the BACKLIT state, an LED backlight is turned on in addition to the LCD display, causing the device to consume 150 mW of power. The display consumes no power when it's off. The display will remain in a given power state until it receives a command from the power manager, possibly in response to environmental conditions. For example, if the display is in the BACKLIT state and it receives a BacklightOff command, then its power state machine will transition to the DISPLAY state, which consumes 50 mW of power.

This basic model must be augmented to overcome limitations in practical situations. First, the model must be able to express performance costs when switching power states. Secondly, there are certain resource types for which the simple model produces very complex state machines. We address these issues next.

3.1.1.1 Power state machines with transition penalties

With certain types of resources there may be transition times associated with transitioning from one power state into the next. For example, moving a CPU component from the SLEEP state to the IDLE state may incur both a time and power penalty if it requires a crystal clock oscillator to start up and stabilize. Likewise, shutting off a disk drive may save significant system power, but major time and power penalties are incurred when the drive is restarted. The simplest way to model the performance price paid for power transitions is through specification of transition delays.

Figure 3 shows the power state machine for a disk drive [15]. Several of the state transitions are annotated with transition penalties, because of the time delays incurred in transitioning from one state to another. Notice that in going from STNDBY to IDLE, a performance penalty of 10 seconds is incurred, representing the amount of time it takes to spin up the disk. Similarly, a performance penalty is incurred when shutting the disk off. These penalties must be taken into account when setting the power management policy.

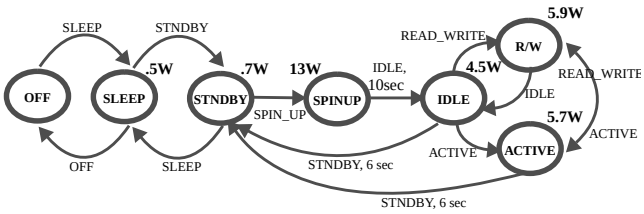

Figure 3: Disk power state machine w/transition penalties

3.1.1.2 Modeling utilization-dependent resources

The power used by certain types of resources, such as memories, is highly dependent upon the level of activity. To accurately model the power of a memory with a simple power state machine, would require a very high level of detail to be specified in either the workload or power manager. In real operation, the memory will change state frequently. Specifying the exact set of read and write accesses would be impossible at this early stage of design, because to specify the memory traffic in detail would require that the design be refined and the firmware written. Even if the requisite level of detail was available, the simulation of a model which accurately specified memory accesses would be too slow to use during optimization.

Each power state of a utilization-dependent resource power state machine has a corresponding activity level. When the power manager sends a command to a resource to change states, it also specifies an activity level. The power used in a given state is computed by the function P_s=*activity_level * peak_power*$_s$, where the peak power is taken either from measurements or from a component data sheet. The power manager can change the activity level of a resource during operation, without changing the resource's power state. The new power figure is then computed.

This type of specification is typical for memory components, because the interactions between a CPU and memory components in a system can be specified in terms of their activity levels. When the CPU is active, the memory components are also active to some degree. The power manager can communicate with the memory components, describing the level of activity that is appropriate for the level of activity of the CPU, to arrive at a good approximation of the power consumption.

As an example of this, Figure 4 illustrates the power state machine for an SRAM [17]. When communicating with the SRAM, the power manager specifies both the state change and the activity level of the SRAM. So, when the SRAM is in state RW, if its activity level at a given point in time is 60%, then the power is 150mW.

3.1.2 Power Manager

The power manager serves as the heart of the system model, translating requests from the external environment and internal resources into resource power state change requests. The power manager is an abstract entity that may or may not correspond to a component in the real system. In the real system, the power manager might be implemented by a software thread within the system's CPU. In other cases, the power manager may be implemented by a separate ASIC.

The *power management policy* implemented by the power manager is the algorithm that decides which resource to send a command to and when to issue a command to a component. A policy can be as simple as keeping everything on all the time, or turning off a resource as soon as it becomes idle. However, more realistic and efficient policies take into account the trade-off between power and performance by adapting to actual workloads.

Notice that in our model all detailed functional information is abstracted away. No notion of what a component must do is expressed in the model, nor are the components' detailed data communication protocols. Details about how the power manager is implemented are also not expressed in the model. The only functionality actually expressed by the model is how the system reacts to incoming requests and with what power consumption. In other words, only the minimum amount of functionality necessary to understand the power behavior is included in the model.

3.1.3 Environment

The final piece of the system power model is the environment, which is represented by a system workload. The environment plays a key part in any accurate representation of a real system's behavior. The environment model consists of a series of external requests over time. Because the systems being modeled are reactive, typical requests include button pushes and external communication events. These requests must take place over a representative time period for the overall model to have validity. For portable systems, enough time should be modeled to give an accurate picture of the system's battery life.

Because the environmental assumptions play such an important part in making proper design decisions, a lot of thought should go into putting together this part of the model. Variants in environmental conditions can be used to test the system's sensitivity to certain assumptions, which helps in producing a robust design.

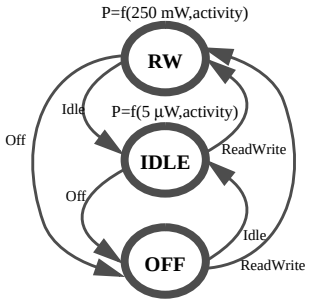

Figure 4: SRAM power state machine with activity levels

Consider a system that processes LAN packets. For that system, the environment is the LAN connection. The environment generates a workload, which is the stream of incoming packets. The availability of realistic workload models is a requirement for obtaining accurate estimates of the power consumed by the system and for optimizing the power management policy. For this system, the workload can be modeled by collecting actual traces of packet arrival times on a real LAN, and by using these traces to drive the model. Alternatively, a stochastic model can be used to produce a stream with given characteristics (such as expected inter-arrival times).

3.2 System-level power estimation

The power model described in the previous subsection maps nicely into an event-driven simulation paradigm. We can implement this model using an existing hardware description language, such as VHDL or Verilog, and use an existing event-driven behavioral simulator to run the model and evaluate the system power under a given set of environmental conditions. One advantage of using a standard HDL and commercial simulator to implement the power model is that the power and functional models for a resource can be developed concurrently, and power estimates can be refined as the design is refined.

There are practical issues that arise when mapping the model to a behavioral simulation. Most practical systems have CPUs driven by firmware. Software running on a CPU can be modeled with a state machine as well, with the only caveat being that interrupts must be modeled. Details of the methodology can be found in [18].

Realistic estimates of battery life depend not only on the average power draw from the battery, but also on the discharge rate over time [14]. Our model can account for these effects because it is event-driven and can provide the battery model with current draw over the workload's time period. The model can give feedback to the designer on optimizing the current draw of the design to obtain the longest battery life from a given battery chemistry.

3.3 A Case Study: PaperClip

We applied our modeling methodology to a real design to validate its accuracy, performance and usability. PaperClip is a hand-held battery-powered electronic clipboard that was developed at HP Labs for electronically entering handwritten information using an inking pen on ordinary paper. The PaperClip user attaches a pad of forms or note paper to the clipboard and enters data on the paper using the attached inking pen. Inside the clipboard is a digitizer which captures the ink electronically while the user is writing. The electronic ink is stored in the clipboard for later transmission to a PC for off-line processing. Its basic system operation can be described as an interconnection of eight basic operational states, as seen in Figure 5.

PaperClip was designed to go into a low-power sleep mode (DEEP-SLEEP) when idle. Touching the stylus to the front of the device (the platen) and beginning to write wakes up the processor (transitioning to the DIGITIZE state) and begins writing ink data into a FLASH ROM. When the stylus is lifted from the platen, PaperClip transitions into the DIGITIZE GAP state and waits for a predetermined period of time until either digitizing resumes or the time-out expires. Once the time-out expires, the system transitions back into QUIESCENT, and then into DEEP SLEEP after another predetermined time period.

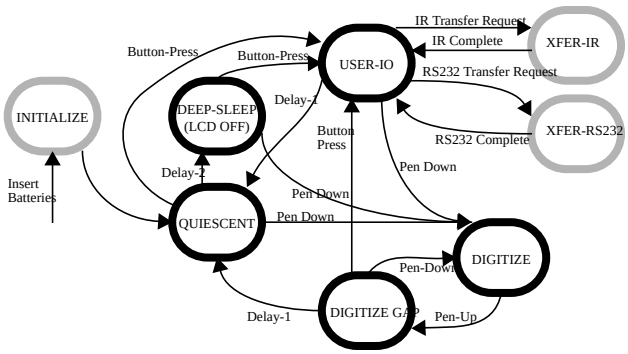

Figure 5: Operation states of PaperClip

Because the lightly-shaded states in the diagram were rarely used in actual system operation, we ignored them during implementation of the power model. The INITIALIZE state was ignored since it is only executed once during an entire simulation (on power up). The data transfer states were ignored since they represent a small fraction of actual system usage. The resulting model was constructed using the five remaining states.

3.3.1 System-level resources

In mapping the real PaperClip hardware into a simulation model, we abstracted away features of the hardware which were uninteresting from a power point of view, such as operations that would only be executed once during the time period of interest. For each component, a VHDL module was created. All miscellaneous logic and pull-up and pull-down resistors were combined into a single pseudo-component with an aggregated power figure which was provided by the designer.

For most PaperClip components, the behavioral model consisted solely of its power state machine description. Only the CPU model contained additional information about functionality. The functional process of the CPU describes the modes of operation of the PaperClip, and controls the operation of all other components, as well as the interaction with the external workload. In other words, the functional model of the CPU describes the software running on it and how the software directs the system activity.

3.3.2 Experimental Results

Overall system power was estimated by simulating the VHDL model under a set of traces extracted from actual usage sessions. A time-dependent power waveform was obtained. The simulation results were compared against measurements from the PaperClip hardware to evaluate the accuracy of the model.

To validate the model against the real hardware, a PaperClip was instrumented, and current was measured for each component in every power state. Because the power states were already defined by the model, the first step was to determine an operational sequence that would put the real hardware into each power state so a measurement could be made.

Because the system spends most of its time in one of four power states, we only measured power during those states. They were:

- **DEEP SLEEP:** Everything is OFF.
- **QUIESCENT**: Everything is OFF except for the small informational LCD display.
- **DIGITIZE**: This state represents the typical digitizing conditions when most ink is good, mimicking actual usage.

	QUIESCENT (mW)			DIGITIZE (mW)			DIGITIZE GAP (mW)		
	estimated	simulated	measured	estimated	simulated	measured	estimated[a]	simulated	measured
LCD	8.5	5	5	8.5	5	5	—	5	5
CPU	0	5	4	75	61	58	—	35	31
Xilinx FPGA	82	2	3	82	16.5	17	—	11	10
SRAM	0	0	0	5	6.5	6	—	0	0
FLASH	0	0	0	2.7	40	40	—	0	0
Other	7.8	8.2	6	7.8	8.2	32	—	8.8	16
Digitizer	0	0	0	30	18	15.5	—	18	1.7
+12V converter in	0	0	0	35	21	18	—	21	2
+5 converter out	98.3	20.2	18.0	216	158.2	176	—	80.8	64

TABLE 1. PaperClip power measurements

a. Designer did not estimate

- **DIGITIZE GAP**: This state represents the 10 second time period after the stylus is lifted from the platen where PaperClip stays awake awaiting the next digitized point. If no further activity is detected with the 10 second period, the PaperClip moves to QUIESCENT. DIGITIZE GAP exists in the actual hardware to avoid losing points when the pen is picked up to dot an ‘i’ or cross a ‘t’, because the processor loses a few points at the beginning of an initial stroke due to processor latency.

Given these definitions for the power states, the PaperClip under test was manually put into the appropriate state and measurements were taken for each state as recorded in Table 1. Because power consumed in DEEP SLEEP differs from power consumed in QUIESCENT by only the LCD power usage, we omitted that state from the table. In the table, the +5 Converter Output row represents the sum of the power from all +5V components, and the power from the system’s +12 volt converter. The stated efficiency of the +12V converter was 80% and was taken from the component’s data sheet. The differences between the power consumed by the digitizer, which is the only active +12V component, and the power absorbed at the input to the +12V converter represents the loss due to inefficiency of the converter. Converter losses represented a significant source of power consumption in the design.

The power consumption numbers used for each component were initially taken from the components’ data sheets. However, after initial measurements we found that the component datasheets were often inaccurate. As a result, we measured individual component power numbers and modified our simulation models to reflect the actual power consumption figures.

The power models of the memories are workload dependent. Initial measurements revealed that our memory workloads were inaccurate, mainly because we did not have detailed information about the firmware. We modified the model to reflect the accurate workloads taken from measured data. In a memory-intensive design, careful attention should be paid to estimating memory workloads, as they can be a big source of inaccuracy.

After back-annotating measured component numbers into the individual component models, the model gave an error of approximately 10% for all significant states. This demonstrates that the model is expressive enough to accurately model the power consumption of a design at a high level of abstraction. The residual error is due to the model’s abstraction level, as shown in the “other” line in the table. This represents the power consumed by the discrete components in the system, such as diodes and pull-up resistors, which were not accounted for in our simulation model.

One benefit of modeling a system at this level accrues from being able to compare the model against the actual hardware. This can help a designer ferret out hardware implementation defects, when power measurements don’t match the original simulation numbers.

Our power estimation approach has several advantages over a simple spreadsheet. First, the model can be used to demonstrate and evaluate workload sensitivity (i.e., sensitivity to different usage models). Second, the model can be used to perform a sensitivity analysis on specific components.

Third, our modeling framework can be used to explore alternative policies in a trial-and-error fashion. The system designer specifies an algorithm and a performance metric, encoding the power management algorithm in the body of the power manager. The event driven simulation measures both power and performance. The distinguishing feature of this approach is that it provides information on the effectiveness of a power management policy very early in the design process. If the results of the simulation are not satisfactory, the designer can either decide to change the policy or to change the design itself to make it more power manageable.

Finally, the model’s accuracy increases as more functional details are specified in the design. The power model automatically keeps in sync with the functional model, even as the functional model becomes more complicated, because the power states and communication with the functional model don’t change as the design gets refined. Thus, the power modeling process can be carried through the entire design process, with little extra work from the designer.

4 Power Management Policy Optimization

In this section we deal with the problem of optimizing the power management policy of a given system. We assume that component selection is fixed; hence, our task becomes to optimize power by turning off unused components and executing functional tasks with the minimum acceptable performance.

Our approach relies on two fundamental assumptions. First, the target system’s workload is dynamic. Hence, periods of high utilization and tightly constrained performance are mixed with periods of low or no utilization and loose performance constraints. Second, the system components are power-manageable, and can operate in more than one power state, trading off power dissipation, performance and reactivity. The mode of operation of a power manage-

able component can be controlled by externally-issued commands from the power manager.

Because of space limitations, we only outline the approach used to optimize the power management policy for PaperClip. Refer to [18] for more details on the policy optimization algorithm.

PaperClip was designed for low-power operation, because battery life was one of the most important performance parameters. Most of the important hardware components are power-manageable and provide at least one low-power shutdown state.

PaperClip's power management policy can be described as follows: "*If the system has been idle for T_d, then transition into the quiescent state (DIGITIZE GAP). If the system has been idle for T_s, then transition to the sleep state*." Although the transition from the "sleep" state to one of the "digitize" states is only a few milliseconds, even a short delay can cause the loss of the first few points of a new pen stroke, imposing a performance constraint. This simple algorithm defines a class of parameterized policies. The space of all policies of this class is defined by varying the values of the two tuning parameters T_d and T_s.

To optimize PaperClip's power-management policy, we applied a simple, greedy optimization algorithm to find values of T_d and T_s that minimize power while satisfying performance constraints. The initial values were set to T_d = 10 seconds and T_s = 1 minute, the values that were used in the real design. The algorithm incrementally modifies the values, runs the simulation to compute performance and power metrics, and chooses the direction of change in parameters values for which maximum power improvement was obtained without violating performance constraints. The performance degradation caused by power management is measured by how many points within a pen stroke are lost due to the delay incurred by transitioning the PaperClip from one of its lower power inactive states to the fully active digitize state.

More formally, we can defined the performance as

$$F = \frac{1}{N_{strokes}} \sum_{i=1}^{N_{strokes}} \frac{T_{active}^{\ i}}{T_{stroke}^{\ i}}$$

where $T_{active}^{\ i}$ is the time spent in the digitize state for stroke *i* and T_{stroke} is the duration of the entire stroke. If the PaperClip is on when the stroke begins, the ratio between the two times is 1. Thus F is upper-bounded by 1 and lower bounded by 0. This performance metric measures how many points are lost for each stroke.

Note that the value of F is strongly influenced by the power management policy, which decides when to turn off the PaperClip. If the policy never turns off the system, the performance metric will be 1. In contrast, an eager power-management policy (i.e., go into the deepest sleep state as soon as the system becomes idle) would most likely lead to values of F smaller than one. Another interesting point is that the impact of a policy on the cost metric is also strongly dependent on the workload. If, for example, the writing consists of many short strokes with short pauses in between, the metric becomes much more sensitive to the value of T_d.

We applied this simple optimization algorithm to the power management policy for several performance constraints and with the workload sample used during estimation. We ran our algorithm on the PaperClip design using a performance constraint of F = 0.9, with an initial solution of T_d = 10 sec, T_s = 60 sec, as implemented in the actual PaperClip. The optimization took less than 5 minutes (80 evaluations) to run, and yielded a solution that satisfied the performance constraint while reducing power by 23%.

5 Conclusions

We have proposed a methodology and algorithms for system-level power estimation and power management policy optimization. Rather than building a new tool, we showed how the model can be implemented in a behavioral simulation language, such as VHDL, and how power estimation can be done using existing behavioral simulation engines. We validated our modeling approach by applying it to an existing design and comparing it to power actually measured, yielding results that were within ~10% of the power measurements. These results were more accurate than original designer estimates with a small penalty paid in added complexity.

We applied a greedy optimization algorithm to the design's power management policy to validate the modeling and estimation approach as a basis for optimization. The algorithm quickly produced a policy which reduced power by 23%. Long term, the promise lies in marrying this model and accompanying tools with a system synthesis or automatic design-space exploration tool, to do design-space exploration and component selection in the face of performance, cost *and* power constraints.

6 References

[1] R. Golding, P. Bosh et al, "Idleness is not sloth," *Proceedings of Winter USENIX Technical Conference*, pp. 201-212, Jan. 1995.

[2] W. Nebel and J. Mermet, *Low power design in deep submicron electronics*, Kluwer 1997.

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

[4] J. Rabaey, M. Pedram, *Low power design methodologies*, Kluwer 1995.

[5] D. Lidsky and J. Rabaey, "Early power exploration - A World Wide Web application," *DAC*, 22-37, June 1996.

[6] D. Liu and C. Svensson, "Power consumption estimation in CMOS VLSI chips," *JSSC*, vol. 29, no. 6, 663-670, June 1994.

[7] R. San Martin and J. Knight, "Power-Profiler: optimizing ASICs power consumption at the behavioral level," *Proceedings of the Design Automation Conference*, 42-47, June 1995.

[8] L. Benini and G. De Micheli, *Dynamic power management: design techniques and CAD tools*, Kluwer 1997.

[9] S. Gupta and F. Najm, "Power macromodeling for high level power estimation," *DAC*, 365-370, June 1997.

[10] C-T. Hsieh, C-S. Ding et al, "Statistical sampling and regression estimation in power macromodeling," *ICCAD*, 583-588, 1996.

[11] P. Landman and J. Rabaey, "Architectural power analysis, the Dual Bit Type method," *TVLSI*, vol. 3, no. 2, 173-187, 1995.

[12] A. Raghunathan, S. Dey and N. K. Jha, "Glitch analysis and reduction in register transfer level power optimization," *DAC*, 331-336, June 1996.

[13] L. Benini, A. Bogliolo, M. Favalli and G. De Micheli, "Regression models for behavioral power estimation," *PATMOS*, 179-187, Sept. 1996.

[14] T.L. Martin and D.P. Siewiorek, "A power metric for mobile systems," *ISLPED*, 37-42, Aug. 1996.

[15] Maxtor CrystalMax Manual, Part #1354C, 1/21/97.

[16] Intel Flash Memory Data Manual, Order Number 290151-005, 11/95.

[17] Sony datasheet: CXK58257AP/ASP/AM-xxL,LL SRAM, 1/93.

[18] L. Benini, R. Hodgson, and P. Siegel, "System-level power estimation and optimization," HP Labs Technical Report, HPL-98-30, 1998.

[19] M. Srivastava, A. Chandrakasa, and R. Broderson, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *TVLSI*," vol. 4, no. 1, 42-55, March 1996.

[20] C.-H. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *Proceedings of the International Conference on Computer-Aided Design*, 28-32, 1997.