POWER ESTIMATION AND OPTIMIZATION

In recent years, there has been great demand for portable electronic systems, such as cellular phones, two-way pagers, personal digital assistants (PDA), and general portable audio and video communications equipment. If current market trends persist, as is generally believed, then these systems will probably continue to be in high demand in the future. A common feature of all portable systems is that they are usually powered by batteries, and that the power consumption of the system determines how long one can use the equipment before having to replace or recharge the batteries, the so-called battery life. Naturally, users of portable electronics prefer to use equipment with the longest possible battery life. In order to increase battery life, one can either use a longer-lasting battery or design the system so that it consumes less power. Unfortunately, progress in battery design has been slow, and longer-lasting batteries can be heavier, which is undesirable in portable equipment. This has led equipment manufacturers to pursue aggressively new and innovative designs that consume as little power as possible.

Modern portable equipment contains a variety of power-consuming components, such as integrated circuits (IC) or *chips,* disk drives, and display devices (typically liquid-crystal displays for computers and PDAs). In the pursuit of low-power design, it is natural to start by reducing the power of the most power-consuming components of a system. Thus, much has been done lately to reduce the power consumption of displays and disk drives, as these parts can consume large amounts of power if not well designed.

This article will focus on integrated circuits, which are pervasive throughout the electronics industry, and not just in portable components. Indeed, low-power ICs are desirable not only for prolonging battery life, but also for better IC reliability, in both portable and line equipment. This is because if the heat generated in a high-power IC is not properly dissipated (through the use of expensive IC packages), the chip temperature will rise, degrading the circuit performance and aggravating various failure mechanisms that can cause the chip to fail much sooner than it otherwise would.

Due to the complexity of modern integrated circuits (millions of transistors per chip), ICs are designed by using sophisticated computer-aided design (CAD) software systems. These consist of myriad software tools (CAD tools) that can be used to enter a description of the design in terms of its parts or behavior, simulate the design using computer simulation models in order to verify its correctness, and then transform the design into the low-level physical specifications (layout) required to manufacture the chip. The design process of state of the art ICs is a very complex procedure involving design groups of hundreds of people, for periods of a year or a few years. This involves the use of many CAD tools, and the overall project requires careful planning and management. The overall flow of activities is referred to as the *design methodology.* Thus, in recent years, the industry has been slowly shifting toward the use of a *low-power design methodology.* While these efforts are still continuing, it is clear that part of such a methodology must be CAD capabilities for *estimating* the power requirements of a proposed design, and for *optimizing* (reducing) the power consumption of a proposed design. Only recently have such power estimation and optimization tools been introduced into the market place, and it is certain that better tools will be released in the future as a result of current research and development efforts in both industry and academia.

Electronic circuits are usually classified as either digital or analog. Digital circuits are those that implement logic functions, and are pervasive throughout modern electronic equipment. Analog circuits process signals that do not have discrete (logic) states, but are more continuous in nature. Amplifiers are a common example of analog circuits. This article will focus on digital circuits, for the following reasons. Analog circuits are typically small (in terms of their number of components), so that designers can easily predict their

power from their knowledge of the design. Furthermore, designers have a few good design practices that lead to low-power analog circuits, beyond which there is little scope for computer-aided optimization. Finally, since most systems are becoming predominantly digital, it is the digital parts that are consuming the most power, and it makes sense to focus on digital chips.

The most power-consuming logic ICs are the very large scale integration (VLSI) components. ICs of this type can contain millions of transistors and consume several tens of watts of power. Next generation VLSI circuits are forecasted to consume around 100 W, which requires very costly packaging and system design solutions. VLSI circuits are built mostly in complementary metal-oxide-semiconductor (CMOS) technology transistors. Even though CMOS was originally introduced for its desirable low-power properties, the sheer number of transistors per chip and the high clock frequencies used have led to the present situation, where modern VLSI CMOS micro-processor chips are dissipating up to 60 W or more.

It is instructive to consider why it is that modern chips consume so much power, and how the technology has evolved up to this point. In broad terms, power in VLSI CMOS is directly proportional to the supply voltage, transistor count, and clock frequency. Although the supply voltage has been steadily and slowly reduced over the last few years in order to control the chip power, the exponential increase in transistor count and clock frequency since the early 1970s has been a phenomenal trend; this trend is the main cause for the high levels of power dissipation in modern ICs. The Semiconductor Industry Association (SIA) road-map for 1998–2001 forecasts transistor widths around 0.18 μm, clock frequencies of 300 to 600 MHz, and 28 to 64 million transistors per chip. Some of these predictions have already been met or exceeded.

Managing the power of a chip design adds to the list of problems that IC designers have to contend with. In the 1970s and 1980s, designers had to worry about two dimensions, namely circuit delay and circuit area. In the 1990s and beyond, power has become the third dimension, significantly complicating the overall design methodology. This article covers the CAD techniques and tools that have been developed or proposed to help designers overcome this problem, specifically for digital CMOS VLSI chips. In some cases, where CAD techniques are not available, certain design decisions and choices that can be made to reduce the power are briefly mentioned. This has been an active area of research in the recent past. Some good survey articles have been written on the overall methodology (1), the estimation problem (2,3), and the optimization problem (4,5).

SOURCES OF POWER DISSIPATION

A CMOS VLSI chip dissipates power because it draws current from its power supply. Indeed the product of the average supply current and the (approximately constant) power supply voltage gives the average power dissipation of the chip. The supply current consists of four components, listed here in decreasing order of importance:

1. Capacitive current
2. Short-circuit current
3. Standby current
4. Leakage current

These will be explained with the help of the CMOS inverter circuit in Fig. 1. During a low-to-high output transition, power supply current is required in order to charge up the output capacitance C_L. This is indicated by the current path labeled I_1 in the figure. In the next transition (high-to-low), the output capacitance is discharged through the local discharge current path I_2. Since the charge is not returned to the power supply, there is a net transfer of charge from the power supply to the ground rail. The component of current required to charge or discharge circuit capacitances is called the *capacitive current,* and the power resulting from it is called the *capacitive power.* This component of power represents the biggest component of IC power. It depends only on the magnitude and number of capacitors, and on how often they are charged/discharged.

Still referring to the CMOS inverter in Fig. 1, the short-circuit current is the current that flows from the power supply directly to ground through the two p and n transistors during a logic transition. This current takes the form of a current pulse, and it is there because for a short time interval during a logic transition both transistors will be conducting. If the circuit is well designed, so that the gate input signal makes a fast logic transition, then the short-circuit current pulse will be narrow, and the short-circuit power will be relatively small. In this case, for typical circuits, this component of the power would represents a small fraction (about 15%) of the total circuit power. Otherwise, if the gate input signal is slow, this component can be more important. In any case, the total short-circuit power of a logic circuit depends on the number of gates, how often they switch, and how fast their inputs switch.

The sum of the short-circuit and capacitive power contributions is called the *dynamic power* of the design, because this power is dissipated only during logic transitions. In contrast, the power due to the standby and leakage current is not directly linked to logic transitions, and the sum of the standby and leakage power is called the *static power.*

Standby current refers to current that is continuously drawn from the power supply, but excluding leakage current. While a fully complementary static CMOS gate, as in Fig. 1, consumes no standby current, the chip may contain other

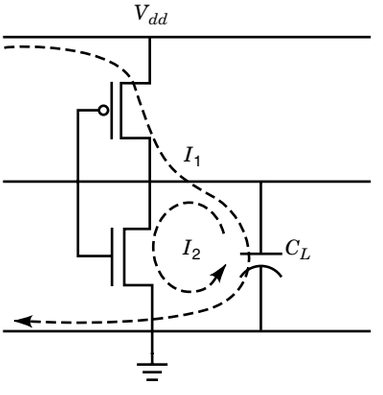

Figure 1. Current flow in a CMOS inverter during a logic transition.

types of circuitry (memory circuits, pseudo-nMOS gates, pass-transistor logic), which can draw standby current from the power supply.

Finally, the leakage current is of two kinds, diode leakage and subthreshold leakage. The diode leakage current is the small, often negligible, current which flows in a reverse-biased pn-junction. Reverse biased pn-junctions abound in CMOS circuits, since all drain and source diffusions constitute reverse biased diodes. This current is small, typically in the 10^{-12} A range for a single diode. It is, therefore, often neglected. The subthreshold leakage current is the current that flows in an MOS transistor when it is supposed to be off (open circuit, due to $v_{gs} = 0$). Even though this current is also small, it is becoming more important nowadays because it increases significantly for low-threshold transistors that are being proposed for future low-power IC technology.

THE POWER DESIGN SPACE

If a capacitor C is being charged from 0 to a voltage V and then discharged, and if this happens f times per second, then the average power being delivered by the charging circuitry is CV^2f. Since capacitive power is the dominant component of the total power, this simple analysis makes it clear that the power dissipation of a VLSI chip depends on the node capacitances, the power supply voltage, and the node-switching frequencies. Thus, in order to develop low-power chip designs, the industry has attacked this problem on three fronts:

1. *Reduce the Power Supply Voltage.* This option is most attractive because of the quadratic dependence on V. However, this is not easy to implement. For one thing, with a new supply voltage, the whole transistor manufacturing process may have to be modified so that the transistors have lower threshold voltages and reasonable noise margins. This can be very expensive. Another complication is that it would complicate the overall system design if some chips are at 5 V while others are at 3 V. Thus, if a microprocessor design is to be executed with a 3.3 V supply, then enough support chips must be made available on the market so that the whole board or system is at 3.3 V. The industry has by now migrated from 5 V to 3.3 V, although 5 V technology is still in use, and is forecasted to slowly migrate to 2 V, or even 1 V supplies. Currently, semiconductor companies are struggling with the technology design issues for transistors with such low supply voltages. In summary, this *voltage scaling* is expensive and slow, and it is imperative that other options be explored.
2. *Reduce the Node Capacitance, by Scaling (Shrinking) the Technology.* Smaller transistors lead to logic gates with less output capacitance, which leads to faster gates and overall higher clock speeds. This is desirable for improving chip and system performance, but the smaller capacitances also provide for lower power dissipation per logic transition. Sometimes, though, the increase in transistor count and clock frequency that results from technology scaling offsets the power gains from reduced capacitance and the overall effect can be an increase in power.
3. *Reduce the Node Activity-Capacitance Product by Design.* The third option is to reduce the product of node switching activity and node capacitance. This leads to power reduction because the capacitive power dissipation of the whole circuit is proportional to the summation of product terms of the form C_if_i, where C_i is the capacitance at node i and f_i is its average switching frequency. This can be achieved by either reducing f_i, without reducing the overall clock speed, or reducing C_i, by reducing the fanout, or both. This can be done through proper design of the circuit. In contrast, the circuit/logic designer has no control over the supply voltage, and on the basic capacitance per unit area of the metal, both of which are determined by the manufacturing process.

VLSI DESIGN ABSTRACTION LEVELS

The design process for VLSI spans many levels of abstraction. When the design is specified as an interconnection of logic gates, we refer to that as being a logic-level view of the design. When the design is described as an interconnection of transistors, it is said to be at the transistor-level. When described in terms of its layout, the design is said to be at the physical or layout level. Designs are usually specified at high levels of abstraction, and then the design effort is aimed at translating that description to lower levels of abstraction, all the way to the layout level. Higher levels of abstraction include the architectural-level (also called *register-transfer-level,* or RTL). At this level, the *structure* of the design is specified, typically in terms of functional blocks and memory elements. Yet another higher level of abstraction is the behavioral-level, where the chip *behavior,* rather than structure, is described in terms of interconnections of behavioral/functional modules. Digital signal processing (DSP) chips are typically represented at the behavioral level, while general microprocessors are specified at the architectural level. DSP designs are transformed from the behavioral to architectural level using high-level behavior-preserving transformations. In reality, complex chips are specified as a mix of behavioral and architectural constructs, because some parts of the system may be amenable to an algorithmic (behavioral) descriptions, while others may not.

Since most designs are implemented by transforming some high-level specifications to lower-level implementations, a gate or transistor level representation of the chip may not be available until the design process is almost complete. If one were to get to that point and only then discover (using a power estimation tool) that the power consumption is unacceptably high, then it would be too expensive to make design changes. The circuit may require significant rework, involving perhaps changes to the overall architecture of the chip, so that the whole design effort may have to be repeated. For this reason, it would be very beneficial to have a power estimation capability at a high level of abstraction. However, as we will see below, estimation from a high level of abstraction is potentially inaccurate, while low-level power estimation can be very accurate. Therefore, a power estimation capability is needed at *every* level of abstraction, in order to check the design at every step in the design flow.

As for optimization, it is also generally believed that higher reductions in power would be possible at higher levels

of abstraction, simply because there would be more scope for changing and optimizing the design at a higher level. Once the logic design has been specified, for instance, there is very little that an optimization tool can do besides rewire certain connections or resize certain gates. While, at the architectural level, an optimization algorithm may change the design to use a different style of implementation altogether, say by using a Booth rather an array multiplier, possibly leading to large reductions in power. Thus, optimization capabilities are most beneficial at higher levels of abstraction, but are also needed at *every* abstraction level.

In the remainder of this article, we will discuss power estimation and optimization techniques at the behavioral, architectural, and logic levels. In the literature, estimation from a transistor level and optimization at the layout level are also discussed, but these techniques will not be covered, due to lack of space and also because there is more interest in the industry in estimation and optimization at the higher levels of abstraction.

POWER ESTIMATION

By *power estimation* we will generally refer to the problem of estimating the *average* power dissipation of a digital circuit. This is different from estimating the *worst case* instantaneous power (6–8), also referred to as the *voltage drop problem,* or the worst case power per cycle (9). These problems will not be discussed in this article. Instead, we will focus on average power estimation, which is directly related to chip heating and temperature and to battery life.

Given a transistor-level description, a simple and straightforward method of average power estimation is to simulate the circuit, say using a circuit simulator, to obtain the power supply voltage and current waveforms, from which the average power can be computed. Techniques of this kind were the first to be proposed. Since they are based on circuit simulation, these techniques can be quite expensive. In order to improve computational efficiency, several other simulation-based techniques were also proposed using various kinds of RTL, gate-, switch-, and circuit-level simulation. Given a set of input patterns or waveforms, the circuit is simulated, and a power value is reported based on the simulation results. Almost all of these techniques assume that the supply and ground voltages are fixed, and only the supply current waveform is estimated.

Even though these simulation-based techniques can be efficient, their utility, in practice, is limited because the estimate of the power which they provide corresponds directly to the input patterns that were used to drive the simulation. This points to the central problem in power estimation, namely, that the power dissipation is *input pattern-dependent.* Indeed, in CMOS and in most other modern logic styles, the chip components (gates, cells) draw power supply current only during a logic transition (if we ignore the small leakage current). Thus, the power-dissipation is highly dependent on the switching activity inside these circuits. Simply put, a more active circuit will consume more power. Since internal activity is determined by the input signals, then the circuit power is input pattern-dependent.

In practice, the pattern-dependence problem is a serious limitation. Often, the power dissipation of a circuit block may need to be estimated when the rest of the chip has not yet been designed, or even completely specified. In such a case, very little may be known about the inputs to this block, and exact information about its inputs may be impossible to obtain. Furthermore, for a microprocessor or a DSP chip, the exact data inputs cannot be determined a priori, because they depend on how the chip is used in the design of a larger board or system.

Recently, several techniques have been proposed to overcome this problem by using probabilities as a compact way to describe a large set of possible logic signals, and then studying the power resulting from the collective influence of all these signals. In order to use these techniques, the user only specifies *typical* behavior at the circuit inputs, in the form of transition probability, or average frequency. If typical input pattern sets *are* available, then the required input probability or frequency information can be easily obtained by a simple averaging procedure. Thus, the average switching frequency at a node is considered to be the mean or average of many possible switching behaviors at the node.

We will classify power estimation techniques as being either static or dynamic. An approach is called *static* when it is based on propagating a probability or activity measure directly through the logic, in order to estimate the average switching frequency. To perform this, special models for circuit blocks must be developed and stored in the cell library. In contrast, other techniques, referred to as *dynamic,* do not require specialized circuit models. Instead, they use traditional simulation models and simulate the circuit, using existing simulation capabilities, for a limited number of randomly generated input vectors while monitoring the power. These vectors are generated from user-specified probability information about the circuit inputs. Essentially, these techniques are based on statistical mean estimation resulting from a Monte Carlo procedure. Using statistical estimation techniques, one can determine when to stop the simulation in order to obtain certain user-specified accuracy and confidence.

Logic Level

At this level of abstraction, we usually exclude special and highly structured circuits such as memory arrays or PLAs, because these are not amenable to a gate level representation and are perhaps better represented at higher levels of abstraction. Instead, we focus on circuits that can be represented as an interconnection of logic gates (of any kind) and memory elements (flip-flops or registers). In describing the estimation techniques at this level, it is also helpful to restrict the discussion to synchronous circuits, and to circuits with just a single clock. Circuits with multiphase clocks can be handled by extensions of the techniques to be presented. Finally, another common simplification is to assume that the memory elements are edge-triggered flip-flops, rather than transparent latches. In a circuit with transparent latches, the power analysis should be identical to the edge-triggered case if no cycle-borrowing is employed.

The first issue to consider is the gate power model. A common simplification at this level of abstraction is to ignore the leakage power and to assume that a logic gate consumes power only when its output makes a logic transition. In reality, a gate does consume power due to incomplete output transitions and due to charging/discharging events at its internal

capacitances that do not lead to an output transition, but these are considered second-order effects and are generally ignored. If, for now, we also ignore the short-circuit current, it can be shown that the average energy consumed per logic transition is:

$$E = QV_{dd} = \frac{1}{2}CV_{dd}^2$$

where V_{dd} is the supply voltage, Q is the average charge delivered per transition, and where C is the gate output capacitance, which includes the gate intrinsic (diffusion) capacitance and the extrinsic capacitance due to interconnect capacitance and gate oxide capacitance of the fanout gates. This simple model is very desirable because it greatly simplifies the power estimation problem and affords reasonable accuracy. It is possible to include the short-circuit power in this model by choosing an appropriate value of C, which we will call C_{eff} (C-effective), such that $(1/2)C_{\text{eff}}V_{dd}^2$ represents the average energy per transition *including* the short-circuit power. Since the short-circuit power depends on the input signal slope during transition and the output loading, this requires one to assume a "nominal" slope for the gate input signals, based on knowledge of the technology, and a nominal output loading. The value of C_{eff} can be determined a priori during a characterization step that takes these factors into account. Thus, in summary, the average energy per output transition for a logic gate is:

$$E = \frac{1}{2}C_{\text{eff}}V_{dd}^2$$

Thus, all that is required in order to estimate the power is to compute the average number of transitions per second for every gate output node, a quantity which we will simply refer to as the *switching activity* at that node. Similar modeling can be performed for the flip-flops.

In a synchronous, edge-triggered, single-clock, sequential circuit, it is clear that when a flip-flop output makes a logic transition, it does so simultaneously with the clock and makes at most one transition per clock cycle. The same is not true for all logic gates. Many gates may experience multiple transitions per cycle, due to the possibly unequal delays from the flip-flop outputs to the inputs of the logic gate. If an even number of transitions occur in a certain cycle, then the gate output was not intended to make a transition at all, and all the transitions that did occur were artificial and not needed. If the number of transitions is odd, then only one of them was required and all the rest were not needed. In any case, these additional unneeded transitions have been called *glitches* in the power literature, and the power due to them is called the *glitch power*. The glitch power for a circuit can be small (20% of total power) or can be large (70% of total power), depending on the circuit structure. The glitch power is hard to handle because it depends on the relative delays inside the circuit.

It is generally the case that all feedback paths in a circuit go through flip-flops. Thus, if somehow the flip-flops are removed, we would be left with (perhaps disconnected) circuit blocks that are all combinational, that is, they contain no feedback paths. Given a general sequential circuit, it is convenient to think of it as partitioned into blocks of combinational logic separated by boundaries consisting of flip-flops. Most of the proposed power estimation techniques assume this partitioning is provided, and take the following two-step approach to power estimation:

1. From an analysis of the overall (sequential) circuit, compute certain switching statistics for the flip-flop outputs.
2. From a knowledge of the flip-flop switching statistics, compute the switching activity for every logic gate in the combinational blocks.

This decoupling of the problem is purely a simplification. It leads to some loss of accuracy because the relationships between various flip-flop output values are lost when applying the second step. The error is felt to be generally acceptable, and this is an ongoing research topic. The two steps mentioned above will be discussed in detail in the following. It is simpler to start with a discussion of the second step, the combinational circuit analysis, and then present the sequential circuit step.

Combinational Circuit Power. Several power estimation methods have been proposed for isolated combinational circuits. All assume that some information is provided about the circuit inputs, mainly in the form of switching statistics. Two statistics are deemed important, the signal probability and the switching activity. The signal probability is the fraction of time that a node is in the high state. The switching activity is the average number of logic transitions per second. Two styles of techniques have been proposed: static and dynamic.

In the static methods, one directly propagates the supplied input statistics into the circuit to compute the corresponding switching activity at all the nodes. We can give a flavor of these methods by considering one of the earliest proposed techniques, the signal probability propagation method of (10). In this method, the signal probability at the inputs to a logic gate are propagated to its output by making the simplifying assumption that the gate inputs are (statistically) independent. As a result if $z = \text{AND}(x, y)$ is an AND gate, then the probability of z is computed simply as $P_z = P_xP_y$. Once the signal probability is available for every gate output node, the node transition probabilities are computed as $2P(1 - P)$, which assumes that the node values before and after the clock edge are independent. Finally, the switching activity is obtained as the transition probability divided by the clock period.

This algorithm is very efficient, but the first independence assumption (called a *spatial independence assumption*) may be unacceptable if the circuit has many reconvergent fanout paths, and the second (called a *temporal independence assumption*) also may not be acceptable. Furthermore, the method does not take delay into account so that glitches are ignored. This and other improved methods are reviewed in more detail in (2). Some of the improved methods are probabilistic simulation (11), transition density propagation (12,13), symbolic propagation (14), correlation coefficients (15), and spatio-temporal correlation (16,17).

The above three issues of temporal independence, spatial independence, and delay sensitivity are major failings of the static methods, because no one method completely deals with these issues in an efficient and practical way. Some methods which succeed in doing so are computationally too expensive.

Particularly annoying is that the error cannot be predicted or bounded up-front. Typically, if one is interested only in the total circuit power, then static methods will on average have an error of around 10% for large circuits. But if one is interested in the power consumption of every gate or of small circuits, then the error is potentially much larger and unacceptable. Nevertheless, these methods are the fastest available and are the only way that we may be able to estimate the power at the gate level for extremely large circuits. Because of this, at least one of these methods (12,13), has been incorporated into commercial products in spite of the above weaknesses. This method directly propagates the switching activity from the circuit inputs to provide the switching activity at all nodes, and is based on a spatial independence assumption. The switching activity $D(x_i)$ at the inputs to a logic gate (or, more generally, a Boolean logic block) are used to compute the switching activity $D(y)$ at its output, according to:

$$D(y) = \sum_{i=1}^{n} P\left(\frac{\partial y}{\partial x_i}\right) D(x_i)$$

where $\partial y/\partial x$ is the *Boolean difference* of y with respect to x, defined as $\partial y/\partial x = y|_{x=1} \oplus y|_{x=0}$ where $\oplus$ denotes the exclusive-or operation.

Another class of methods, called *dynamic,* is based on the use of traditional simulators and simulation models. The key idea is that, even though the power depends on the specific vectors, if one were to simulate the circuit for randomly chosen typical vectors, then the average cumulative power measured from the simulation will converge to the true average power. It turns out that the number of vectors required to achieve convergence can be quite small, under 100 vectors for circuits with a few hundred gates or more. Often, circuits with thousands of gates will converge with under 50 vectors. Also, it turns out that using statistical mean estimation techniques, it is possible to tell, during the simulation, when to stop the simulation in order to achieve some user-specified accuracy and confidence in the result, leading to a stopping criterion. Being able to specify the accuracy up-front is a major advantage of these methods. Glitches are taken into account and only the primary inputs are assumed spatially independent, although this is not a limitation of these methods (they can be extended to not require this).

Such methods were proposed in (18,19) for finding the total average power of a circuit. They are fast but do not provide estimates of the individual node power values. An extension, given in (20), does so, but becomes somewhat slower. Further improvements have been proposed in (21) and (22).

Sequential Circuit Power. In this case, the objective is to compute the switching statistics at the flip-flop outputs. These would then be used for the combinational circuit analysis. Given the above discussion of combinational circuit techniques, it is clear that we need two statistics, the signal probability and the switching activity. To simplify the analysis, we will only discuss the estimation of the signal probabilities. Estimation of the switching activity is similar. Since flip-flop outputs constitute the circuit *state,* they are usually referred to as the *state bits.*

Once more, two styles of techniques have emerged, static and dynamic. In the static methods (23,24), initial values of probability at the state bits are assumed, and they are then propagated through the circuit (around the feedback loops) in order to provide updated values of the assumed probabilities. This process is repeated until convergence of the probability values is obtained. One problem with this approach is that the state bits are assumed independent during the propagation, but the error due to this can be reduced by unrolling the feedback loop several times.

In the dynamic method (25), several copies of the whole circuit are simulated in parallel using randomly chosen input vectors until convergence of the monitored node statistics is obtained. This method can solve very large sequential circuits very efficiently, and has the advantage that the desired accuracy and confidence can be specified up-front. It has been used to estimate the power for circuits with up to 1,500 flip-flops and 20,000 gates.

Architectural Level

At this level of abstraction, also called the *register-transfer level* (RTL), the circuit is described as an interconnection of clocked memory elements (flip-flops or registers) and combinational logic blocks whose gate-level structure is not specified. Typically, the combinational logic blocks may be specified only as Boolean functions, so that the design description consists of flip-flops and Boolean black boxes. Since the flip-flops are specified, then it is possible to simulate the circuit, as described above, in order to estimate the switching statistics at the flip-flop outputs. After this essential first step, it remains to compute the power consumed by the Boolean blocks, given their input/output switching statistics. Thus, the problem reduces to developing a power model for these blocks that can be used to compute their power from their I/O statistics.

In some cases, the Boolean blocks may correspond to circuit blocks that were used in previous designs. In this case, the detailed implementation of the combinational black boxes is completely known. This is, for example, the case in DSP designs, where the circuit blocks come from a library of well-characterized adders, multipliers, and so forth, and where the design task may be to determine which type of adder or multiplier to use in a given chip design. In this case, it is still advantageous to carry out the analysis at a high level of abstraction, because the analysis can be done much faster. We refer to techniques of this kind as being *bottom-up* approaches—the low-level details are known, but we choose to ignore them and use instead a simplified high-level model of the block behavior. This is essentially a macro-modeling for power approach. Bottom-up techniques have been proposed in (26), where black-box models (macro-models) are built for circuit blocks by a process of characterization that models the block power as a function of the input/output signal statistics (probabilities) of the block. Other details are also included, such as the bus width, average capacitance, etc.

In other cases, the low-level details of the circuit blocks may be truly yet unknown, because such a circuit block my never have been designed before. This presents a harder problem to solve of extracting power out of pure (Boolean) functionality. We refer to such techniques as being *top-down.* Recently, some top-down techniques have been proposed (27,28) that make use of entropy of a logic signal as a measure of the amount of information that can be carried by that signal. The rationale for this is that the power requirements of a circuit

is probably related to the amount of computational work that the circuit performs, which has traditionally been modeled with the entropy measure.

All these techniques are fairly recent, and it is not clear yet how useful they will be in practice, or how their performances compare/contrast in a practical setting.

Behavioral Level

In this case, the circuit description is at an even higher level of abstraction, where it is not clear exactly where the flip-flops are, and the design is an interconnection of blocks for which only their behavior is known. These blocks will eventually be implemented using flip-flops and/or combinational logic, but the specific architecture to be chosen for the implementation is not known. For instance, all we may know about a given block is that it performs n additions, but we may not know whether it performs them sequentially (on a single adder) or concurrently (on n adders). Or, a block may be a small microprocessor that will be embedded in a larger chip design, and which may be only specified in terms of its instruction set and its I/O ports.

Power estimation at this level of abstraction is, understandably, very difficult, but also very appealing because of the potential gains of knowing the power so early in the design process. Estimation techniques in this area are still in their infancy, and much needs to be done to develop practical and accurate solutions. Notable is the method of (29) in which the design is described with a behavioral flow-chart that shows several computational resources (behavioral blocks, or modules) and the way that they interact. From a simulation of the behavior, the frequency with which a resource is accessed is measured (in accesses per second); call this f. The total capacitance inside the module, call this C, is either known from its low-level description (bottom-up approach), or is a rough estimate from prior design knowledge or using techniques borrowed from high-level synthesis (top-down approach). With this, the average power for a behavioral module is given by:

$$P_{avg} = \alpha C V_{dd}^2 f$$

where α is the average node switching activity *per access* inside the module. The estimation of α is difficult. One way is to just use a fixed number, obtained from experimental studies of prior designs of the same class as this. For instance, minicomputers have been found experimentally to have α in the range 0.01 to 0.005, while microcomputers have α in the range 0.05 to 0.01. Another way of estimating α is to simulate a low-level description of the design, if available (bottom-up approach), under some arbitrary input switching statistics, and to use the resulting α value as a fixed number, and not bother to account for its dependence on the input switching statistics once it is embedded in the system being considered (to do otherwise would be computationally too expensive).

Interconnect Issues

Missing in all of the above discussion has been the issue of interconnect length, which, of course determines the physical capacitance at a node. In the past, when gate delays (capacitance) were significantly larger than wire delays (capacitance), the interconnect could be ignored, or somehow its length and capacitance would have been estimated by an intelligent guess. Today and in the future, due to the fine dimensions of sub-micron CMOS technology, this is no longer the case. Interconnect delays are now dominant, significantly larger than gate delays (52). So any accurate power estimation must factor in the interconnect capacitance. If the design description has been extracted from a layout, then we may have good estimates of the wire capacitance. But this is rarely the case, because it would be too late to wait until the layout has been done before doing a power estimation. More frequently, power estimation (and optimization) would be attempted long before layout.

This problem impacts estimation and optimization at all levels of abstraction. The research literature (30,31) includes several references to approaches that attempt to estimate the average wire length from knowledge of the circuit structure and function. Application and validation of these techniques in the realm of power analysis is an ongoing and future research problem.

POWER OPTIMIZATION

Reducing the average power dissipation of a VLSI chip is a fairly recent concern of the industry, which became prevalent in the early 1990s. It is now a major concern, and all design groups are faced with power management problems. A variety of techniques have been developed, both in industry and academia, to address this problem. As pointed out previously, three courses of action are open: (1) reduce the power supply voltage, (2) reduce the node capacitance through technology scaling, and (3) reduce the node activity-capacitance product by design. The first two of these have to do with the manufacturing process, while the third has to do with design and CAD, and is the one option with which we are concerned in this article.

If the circuit contains n nodes (logic gate outputs) and if each node has capacitance C_i (this can be C_{eff} in order to include the short-circuit current as explained previously) and has a switching activity D_i (average number of transitions per second), then we can write:

$$P_{avg} \propto \sum_{i=1}^{n} C_i D_i$$

Thus the power is proportional to the summation of a large number of cross-products of activity and capacitance. If the values of the product terms are reduced, then the power will be reduced as well. To be sure, one further way of reducing the value of the summation is to reduce the value of n, the number of gates, which is related to circuit area. Thus, to some extent, area optimization can lead to power optimization. In some cases, though, such as when one considers area reduction through gate sizing, the minimum-power circuit is not necessarily the minimum-area one (51), mainly due to the effect of gate sizing on the overall short-circuit current of the circuit. Nevertheless, it is fortunate that, at least as far as the capacitive power is concerned, the two objectives of area and power do not conflict. Unfortunately, this is not the case with the delay objective. In order to reduce circuit delay often requires one to use larger gates, which will dissipate larger capacitive power.

The complex interactions between the area, delay, and power objectives lead to a situation where the objective function to be reduced during optimization can be hard to specify, and even harder to evaluate. Traditionally, optimization has been done with a compound objective function that takes into account both delay and area. Nowadays, it also has to take into account the power. The power component of this objective function is supposed to reflect the change in circuit power due to some candidate transformation that is being considered during optimization. We will refer to this as the *power cost function.*

In principle, the power cost function has only to perform a power estimation. However, since the cost function has to be evaluated several times during an optimization process, dynamic (simulation-based) estimation methods cannot be used, because they can take too long, especially since we often need to know the power consumption (or switching activity) of every gate in order to decide what optimizations to apply. On the other hand, static estimation methods are notoriously inaccurate for estimating individual gate power or switching activity. This is a serious problem that confronts power optimization. The few existing solutions include application of a mix of static and dynamic estimation in a way that offers an acceptable trade-off between execution speed and accuracy.

In the following, we will look at optimizations that can be applied at the three levels of abstraction previously considered, namely, the logic, architecture, and behavioral levels. At all levels, certain transformations will be applied to modify the design, and the cost function is the central difficulty. Given the loss of design detail at higher levels of abstraction, it is clear that estimation, and therefore the cost function evaluation, will not be entirely accurate. Given this, we are interested in cost functions with so-called *relative accuracy* rather than *absolute accuracy.* This means that we need a cost function that correctly predicts that we are on the right track, that transformation A would reduce the power, and would do so more than transformation B, but the exact value of the cost function may not be equal to the true power consumption of circuit. To date, there is no simple cost function that has been rigorously demonstrated to have this property, and this is an ongoing research topic.

In the sections below, we will discuss briefly the different optimizations that have been applied at the different levels. Most of these will be part of CAD, that is, will be part of *synthesis* methods [for a general reference on synthesis, see (32)], but some will be design decisions or styles that can be used irrespective of automatic CAD optimization. It is generally agreed that optimizations at the highest level have the potential of producing the most gains, because there is more freedom to make design changes at that the higher levels. But optimizations at all levels are useful and possible, as we discuss below.

Logic Level

Logic level optimization is usually performed as part of the process of *logic synthesis,* which is the automatic translation from an RTL description to a gate-level description that refers to a given gate library. In this process of translation, the circuit representation goes through different intermediate forms, and several kinds of optimizations or transformations are performed at intermediate steps in order to improve the value of the objective function. These transformations range from retiming the provided RTL, to looking for common Boolean expressions to form a multi-level logic network, to choosing the right size gates from the library, etc. We will consider some important transformations below and briefly give a feel for how they work and how they may be adapted to the new power objective.

State Assignment. If the circuit implements a finite-state machine (FSM), then the circuit states are assigned specific codes (Boolean vectors) that uniquely identify each state. Choosing the best state codes (to provide the smallest objective function value) is called the state assignment problem. This is a very difficult classical synthesis problem that can only be solved efficiently when the objective function is quite simple and the number of states is small. In case of classical area optimization, the cost function may be simply the number of literals in a two-level Boolean expression for the Boolean function representing the combinational part of the circuit. For power, one can aim to minimize the switching activity on the state lines, or to minimize the Hamming distance between neighboring states. The Hamming distance between two states is the number of bits that are not the same in their two codes, i.e., it is the number of bits that have to switch if a transition occurs from one of these states to the other. The Hamming distance measure does not reflect the effect of the code assignment on the size and functionality (and therefore, power) of the combinational logic. A recent technique that does include these effects is given in (33).

Clock Control. A common-sense technique to reduce power dissipation is to shut off circuit sections that are not required to perform a useful function for a certain time period. Inside an IC, this does not mean that the supply voltage is turned off to parts of the circuit, because that creates many problems, such as having to recover the correct circuit state and the turn-off/turn-on transients and the resulting noise. Instead, this only means turning the clock off (holding it fixed at either logic 0 or 1) so that the registers or flip-flops do not keep switching when the data stored in them is not needed. This general solution can be applied at all levels of abstraction and is common practice in the industry by now. However, it is not always easy to automate.

At the logic level, there have been some attempts to automate this procedure. The method in (34) uses so-called *precomputation architectures,* which are essentially generalizations of the idea of look-ahead circuitry, in order to determine inexpensively whether or not some Boolean function needs to be computed in the next clock cycle. This is implemented by using additional circuitry to monitor the flip-flop inputs and, if the transition there is a "don't care" (will not cause a useful transition at the circuit outputs), then the clock is disabled, and that transition is not applied at the combinational circuit inputs. Other related techniques include guarded evaluation (35) and activity-driven clock network design (50).

Multi-Level Logic Optimization. At this point in the synthesis flow, one is dealing with a combinational logic circuit, represented as a multi-level Boolean network. A Boolean network is an interconnection of artificial Boolean gates that may not correspond to any real gates in the library. For this reason, this representation is said to be *technology indepen-*

dent and constitutes an *intermediate* form of the circuit being designed. Previously, several kinds of optimizations have been applied to the Boolean network in order to reduce its area, including making use of "don't cares" and extraction of common sub-expressions to do Boolean factoring. For area minimization, the cost function used was related to the literal count (the number of Boolean variables). In order to adapt these techniques to the power minimization objective, the methods in (36,37) include switching activity in the cost function and predict the effect of the change in the function on its fanout cone. The method in (38) modifies the Boolean factoring procedure by using a cost function that looks at the amount of loading and logic sharing. One needs to keep in mind though, that the circuit is at this point technology independent, so that notions of loading and switching activity are approximate, because no capacitance or delay information is available yet.

Technology Decomposition. Once the processing of the technology independent Boolean network is complete, it is time to start selecting gates from the library to implement the artificial gates of the Boolean network. As a prerequisite to that step, one performs a so-called *technology decomposition,* by which an artificial gate is decomposed into a tree of smaller gates, whose sizes are such that it is possible to find a library gate to implement each of them. For example, an artificial gate in the Boolean network may be a 10-input NAND gate. For performance reasons, such a large gate is never implemented as a single gate and will not be found in the library. Instead, this is decomposed into a simple tree interconnection of 2-input NANDs, which are sure to be found in the library. Traditionally, this decomposition is implemented as a balanced binary tree, in order to keep the delay of the tree structure at a minimum. For low-power, considerations of switching activity at the tree inputs are used to restructure the tree to reduce its overall switching activity. Typically, the result is a tree which is not balanced, and examples of these techniques may be found in (39,40).

Technology Mapping. After technology decomposition, the circuit is mapped to the library by selecting library gates that can be used to implement either single gates or groups of gates in the decomposed Boolean network. The cost function used in the traditional mapping algorithm has been extended to take switching activity and power into account in (39,41).

Post-Mapping Transformations. After the circuit has been mapped, there is a much better chance of doing a good job of low-power optimization because one is working with a technology-dependent representation. The gate delays and capacitances are known throughout the circuit. It is only at this point, for instance, that one can accurately estimate the glitching power, because one can use dynamic power estimation to compute a cost function that includes the glitches. However, given the relatively higher computational cost of dynamic estimation, when the individual gate power values or switching activity are desired, it cannot be applied for every cost function evaluation. Instead, it is applied every now and then as a corrective measure, and in the meantime static estimation techniques can be used, usually in a small neighborhood around the optimization site, in order to update the cost function value. We briefly consider the following optimizations:

1. *Rewiring and Transduction.* The transduction method (42) is a logic synthesis approach that applies transformations to the mapped network. One of these transformations rewires the network in a way that does not change its overall function (making use of internal don't cares) if the rewired network has a lower objective function value. This has been applied with a power cost function, leading to power reductions of 5 to 20%.
2. *Path Balancing.* If the delays along the paths leading to the inputs of a logic gate are approximately equal, then there will be few or no glitches at the gate output. Thus path balancing has been applied to equalize the path delays, through either delay insertion or gate sizing. In either case, there are possible disadvantages that one should watch out for, such as increased area or delay, or even increased power in some other part of the circuit.
3. *Retiming.* If a node has high glitching activity, then if a flip-flop is inserted right after it, only the one meaningful transition at that node would be allowed to propagate through. This is the idea behind using retiming for power reduction. The flip-flops are moved around, while preserving the circuit functionality and delay, in order to throttle glitches and stop them from propagating downstream, thus reducing power.

As a final word, it should be stated that, in spite of all the above proposed techniques, some of which are quite complex and involved, power reduction from gate-level optimization is limited. In the industry, if an industrial-strength logic synthesis is applied for minimum area, it is typically found that the resulting circuit has been optimized to such an extent that only about 5% reduction in its power is possible through further power-specific gate-level optimization. This is in-line with the comments made earlier that optimizations at higher levels can have more impact.

Architectural Level

Higher gains in power reduction are possible at this level. Indeed, order of magnitude reduction in power has been reported by choosing the right architecture for an application. Unfortunately, there are no good automatic algorithms for making these choices. The automatic translation from a behavioral description to an architectural (RTL) description is called *high-level synthesis.* This topic has been studied for over 10 years (43), but no commercial tools exist yet that efficiently provide a good RTL solution given a general behavioral level specification for a large design. As a result, there are no commercial architectural level power optimization tools available today. In this section, we will consider the types of decisions/choices that such a tool would have to make in order to reduce power, and consider some attempts that have been made recently to solve this problem.

One type of transformation that has been applied at this level is to consider that the voltage supply is variable and to consider decreasing the voltage as much as possible in order to reduce the power. Since voltage reduction leads to longer delays, and in order to compensate for this, the logic is repli-

cated and concurrent computation and multiplexing are used to maintain the computational throughput. This has been applied mainly to DSP designs (44).

Other approaches aim to construct the architecture in a way that maintains a *locality of reference.* This means that, whenever possible, communication between computational resources should take place over small distances, in a local neighborhood. One should minimize the accesses to central resources such as memories, ALUs, or buses. This approach helps reduce the power because long interconnect lines can be a big source of power dissipation. This technique, even though it sounds simple, is hard to automate, because of the complexity of the interactions that have to be considered. Architectural choices can impact the amount of concurrency, multiplexing, and frequency of a design, so the search problem for the best architecture is a very difficult one.

Existing high-level synthesis approaches include a 3-step process by which the behavioral specification is translated to the RTL level. These are *allocation* (how many instances of each hardware resource are needed?), *assignment* (on what hardware resource will a behavioral level computational operation be performed?), and *scheduling* (when will the operation be executed?). Traditionally, high-level synthesis tries to minimize the number of hardware resources needed (area) and/or the total time of the schedule (delay).

A common traditional approach in high-level synthesis is to schedule the operations so as to keep as many hardware resources busy as possible. This helps reduce the number of required resources (area) but is not always a good approach for power reduction. In fact, in some cases, one may have to add additional resources in order to reduce the overall switched capacitance.

For low-power, and in order to maintain the locality of reference, the assignment algorithm should aim for regularity and locality. This reduces the interconnect overhead and power. Scheduling can also affect the signal correlations at the inputs of resources, which can affect the power consumed in these resources. In general, the difficulty lies in developing a cost function that takes the switching activity into account and in deciding what transformations to apply and in what sequence. For further reading, consult (4,29,45).

Behavioral Level

At this highest level of design specification, the IC may be represented by an algorithm, or a combination of algorithms or functional modules. Due to the difficulty of this problem, there are almost no CAD optimization techniques at this level, but there are many proven design decisions and styles that have been shown to work in certain situations.

An obvious approach is to choose the right algorithm for a given function. In practice, this means rewriting the algorithm in order to reduce the number of times that certain functional operations are performed. The operations chosen should be those that are known to require a lot of energy per computation.

High-level design decisions may be made at this point, such as choosing to recycle the energy back to the power supply, so-called *adiabatic computing* (46). These very interesting techniques, however, are not yet practical enough. Memory-optimizing transformations can also be used to maintain the locality of reference. For instance, one can replace expensive accesses to background (secondary) memory by accessing foreground memory or by using distributed memory (47).

Another option is to choose the right data representation or encoding so as to reduce switching activity. For instance, the commonly used 2's complement notation has the disadvantage that all data bits switch together when the data value changes from 0 to -1, which occurs quite often. In contrast, the sign magnitude representation produces a single bit change for this occurrence. And finally, for embedded processors, one can consider choosing the right instruction set so as to reduce power, and compiling software to produce low-power programs. This issue has been explored in (48).

Other power optimizations have been applied at the behavioral level. The reader is referred to (49) for a more detailed discussion.

BIBLIOGRAPHY

1. D. Singh et al., Power conscious CAD tools and methodologies: A perspective. *Proc. IEEE,* **83**: 570–593, 1995.
2. F. N. Najm, Power estimation techniques for integrated circuits. *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design,* **13**: 492–499, 1995.
3. P. Landman, High-level power estimation. *Proc. Int. Symp. Low Power Electron. Design,* **1**: 29–35, 1996.
4. M. Pedram, Power minimization in IC design: principles and applications. *ACM Trans. Design Autom. Electronic Syst.,* **1** (1): 3–56, 1996.
5. S. Devadas and S. Malik, A survey of optimization techniques targeting low power VLSI circuits. *Proc. Design Autom. Conf.,* **32**: 242–247, 1995.
6. S. Chowdhury and J. S. Barkatullah, Estimation of maximum currents in MOS IC logic circuits. *IEEE Trans. Comput.-Aided Des.,* **9**: 642–654, 1990.
7. S. Devadas, K. Keutzer, and J. White, Estimation of power dissipation in CMOS combinational circuits using Boolean function manipulation. *IEEE Trans. Comput.-Aided Des.,* **11**: 373–383, 1992.
8. H. Kriplani, F. N. Najm, and I. Hajj, Pattern independent maximum current estimation in power and ground buses of CMOS VLSI circuits: Algorithms, signal correlations, and their resolution. *IEEE Trans. Comput.-Aided Des.,* **14**: 998–1012, 1995.
9. S. Manne et al., Computing the maximum power cycles of a sequential circuit. *Proc. Design Automation Conf.,* **32**: 23–28, 1995.
10. M. A. Cirit, Estimating dynamic power consumption of CMOS circuits, *Proc. Int. Conf. on Computer-Aided Design,* **5**: 534–537, 1987.
11. F. Najm, R. Burch, P. Yang, and I. Hajj, Probabilistic simulation for reliability analysis of CMOS VLSI circuits, *IEEE Trans. Comput.-Aided Des.,* **9**: 439–450, 1990 (Errata in July 1990).
12. F. Najm, Transition density: A new measure of activity in digital circuits, *IEEE Trans. Comput.-Aided Des.,* **12**: 310–323, 1993.
13. F. Najm, Low-pass filter for computing the transition density in digital circuits, *IEEE Trans. Comput.-Aided Des.,* **13**: 1123–1131, 1994.
14. A. Ghosh et al., Estimation of average switching activity in combinational and sequential circuits, *Proc. Design Automation Conf.,* **29**: 253–259, 1992.
15. C-Y. Tsui, M. Pedram, and A. M. Despain, Efficient estimation of dynamic power consumption under a real delay model, *Proc. Int. Conf. on Computer-Aided Design,* **11**: 224–228, 1993.

16. R. Marculescu, D. Marculescu, and M. Pedram, Switching activity analysis considering spatiotemporal correlations, *Proc. Int. Conf. on Computer-Aided Design,* **12**: 294–299, 1994.
17. R. Marculescu, D. Marculescu, and M. Pedram, Efficient power estimation for highly correlated input streams, *Proc. Design Automation Conf.,* **32**: 628–634, 1995.
18. C. M. Huizer, Power dissipation analysis of CMOS VLSI circuits by means of switch-level simulation, *IEEE European Solid State Circuits Conf.,* pp. 61–64, 1990.
19. R. Burch et al., A Monte Carlo approach for power estimation, *IEEE Trans. VLSI Syst.,* **1**: 63–71, 1993.
20. M. Xakellis and F. Najm, Statistical Estimation of the Switching Activity in Digital Circuits, *31st ACM / IEEE Design Autom. Conf.,* **31**: 728–733, 1994.
21. L-P. Yuan, C-C Teng, and S-M Kang, Statistical estimation of average power dissipation in CMOS VLSI circuits using nonparametric techniques, *Proc. Int. Symp. on Low Power Electronics and Design,* **1**: 73–78, 1996.
22. C-S. Ding et al., Stratified random sampling for power estimation. *Proc. Int. Conf. on Computer-Aided Design,* **14**: 576–582, 1996.
23. J. Monteiro and S. Devadas, A methodology for efficient estimation of switching activity in sequential logic circuits, *ACM / IEEE 31st Design Autom. Conf.,* **31**: 12–17, 1994.
24. C-Y Tsui, M. Pedram, and A. M. Despain, Exact and approximate methods for calculating signal and transition probabilities in FSMs, *ACM / IEEE 31st Design Autom. Conf.,* **31**: 18–23, 1994.
25. F. N. Najm, S. Goel, and I. N. Hajj, Power estimation in sequential circuits, *ACM / IEEE 32nd Design Autom. Conf.,* **32**: 635–640, 1995.
26. P. E. Landman and J. M. Rabaey, Architectural power analysis: The dual bit type method, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,* **3**: 173–187, 1995.
27. M. Nemani and F. N. Najm, Towards a high-level power estimation capability. *IEEE Trans. Comput.-Aided Des.,* **15**: 588–598, 1996.
28. D. Marculescu, R. Marculescu, and M. Pedram, Information theoretic measures for power analysis. *IEEE Trans. Comput.-Aided Des.,* **15**: 599–610, 1996.
29. R. Mehra and J. Rabaey, Behavioral level power estimation and exploration. *Proc. Int. Workshop on Low Power Design,* **1**: 197–202, 1994.
30. F. Kurdahi and A. C. Parker, Techniques for area estimation of VLSI layouts. *IEEE Trans. Comput.-Aided Des.,* **8**: 81–92, 1989.
31. H. T. Heineken and W. Maly, Standard cell interconnect length prediction from structural circuit attributes. *Proc. Custom Integrated Circuits Conf.,* pp. 167–170, 1996.
32. G. De Micheli, Synthesis and Optimization of Digital Circuits. New York: McGraw-Hill, 1994.
33. C-Y. Tsui et al., Low power state assignment targeting two- and multi-level logic implementations. *Proc. Int. Conf. on Computer-Aided Design,* **12**: 82–87, 1994.
34. M. Alidina et al., Precomputation-based sequential logic optimization for low power. *IEEE Trans. VLSI,* **2**: 426–436, 1994.
35. V. Tiwari, S. Malik, and P. Ashar, Guarded evaluation: pushing power management to logic synthesis/design. *Proc. Int. Symp. on Low Power Design,* **1**: 221–226, 1995.
36. A. Shen et al., On average power dissipation and random pattern testability of CMOS combinational logic networks. *Proc. Int. Conf. on Computer-Aided Design,* **10**: 402–407, 1992.
37. S. Iman and M. Pedram, Multi-level network optimization for low power. *Proc. Int. Conf. on Computer-Aided Design,* **12**: 372–377, 1994.
38. K. Roy and S. C. Prasad, Circuit activity based logic synthesis for low power reliable operations. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,* **1**: 503–513, 1993.
39. C-Y Tsui, M. Pedram, and A. M. Despain, Technology decomposition and mapping targeting low power dissipation. *ACM / IEEE 30th Design Autom. Conf.,* **30**: 68–73, 1993.
40. R. Panda and F. N. Najm, Technology decomposition for low-power synthesis. *Proc. Custom Int. Circuits Conf.,* pp. 627–630, 1995.
41. V. Tiwari, P. Ashar, and S. Malik, Technology mapping for low power. *Proc. Design Automation Conf.,* **30**: 74–79, 1993.
42. S. Muroga et al., The Transduction Method—Design of Logic Networks Based on Permissible Functions. *IEEE Trans. Comput.,* **38**: 1404–1424, 1989.
43. R. A. Walker and R. Camposano, *A Survey of High-Level Synthesis Systems.* Boston: Kluwer, 1992.
44. A. P. Chandrakasan et al., HYPER-LP: A system for power minimization using architectural transformations. *Proc. Int. Conf. on Computer-Aided Design,* **10**: 300–303, 1992.
45. S. Wuytack et al., Global communication and memory optimizing transformations for low power systems. *Int. Workshop for Low-Power Design,* **1**: 203–208, 1994.
46. W. C. Athas et al., Low-power digital systems based on adiabatic-switching principles. *IEEE Trans. VLSI Syst.,* **2**: 398–407, 1994.
47. J. Bunda, D. Fussel, and W. Athas, Evaluating power implications of CMOS microprocessor design decisions. *Proc. Int. Workshop on Low-Power Design,* **1**: 147–152, 1994.
48. V. Tiwari, S. Malik, and A. Wolfe, Power analysis of embedded software: a first step towards software power minimization. *IEEE Trans. VLSI,* **2**: 437–445, 1994.
49. R. Mehra et al., Algorithm and architectural level methodologies for low power. In *Low Power Design Methodologies.* J. Rabaey and M. Pedram, eds., New York: Kluwer, 1996, pp. 333–362.
50. G. E. Tellez, A. Farrahi, and M. Sarrafzadeh, Activity-driven clock design for low power circuits. *Proc. Int. Conf. on Computer-Aided Design,* **13**: 62–65, 1995.
51. S. S. Sapatnekar and W. Chuang, Power vs. delay in gate sizing: conflicting objectives? *Proc. Int. Conf. on Computer-Aided Design,* **13**: 463–466, 1995.
52. L. Pileggi, Coping with RC(L) interconnect design headaches. *Proc. Int. Conf. on Computer-Aided Design,* **13**: 246–253, 1995.

FARID N. NAJM
University of Illinois at Urbana-Champaign

POWER ESTIMATION OF SOFTWARE. See SOFTWARE PERFORMANCE EVALUATION.

POWER FACTOR. See REACTIVE POWER.