High-Level Power Modeling, Estimation, and Optimization

Enrico Macii, Massoud Pedram, *Member, IEEE*, and Fabio Somenzi

***Abstract*—Silicon area, performance, and testability have been, so far, the major design constraints to be met during the development of digital very-large-scale-integration (VLSI) systems. In recent years, however, things have changed; increasingly, power has been given weight comparable to the other design parameters. This is primarily due to the remarkable success of personal computing devices and wireless communication systems, which demand high-speed computations with low power consumption. In addition, there exists a strong pressure for manufacturers of high-end products to keep power under control, due to the increased costs of packaging and cooling this type of devices. Last, the need of ensuring high circuit reliability has turned out to be more stringent. The availability of tools for the automatic design of low-power VLSI systems has thus become necessary. More specifically, following a natural trend, the interests of the researchers have lately shifted to the investigation of power modeling, estimation, synthesis, and optimization techniques that account for power dissipation during the early stages of the design flow.**

This paper surveys representative contributions to this area that have appeared in the recent literature.

***Index Terms*— Behavioral and logic synthesis, low power design, power management.**

I. Introduction

IN ORDER to shorten the overall time-to-market of new products, today's electronic systems are designed from specifications given at a very high level of abstraction. This novel design paradigm is made possible by the recent availability of electronic design automation (EDA) tools that can take, as input, the description of a system expressed in a hardware description language (HDL) like VHDL or Verilog, and that can automatically produce the corresponding gate-level implementation with very limited human intervention. From there, well-established technology can be exploited to generate transistor-level netlists and layout masks.

Fig. 1 summarizes the flow of operations that are required to go from a system-level specification to an architecture made of a processor, a memory, a few register-transfer level (RTL) macrocells, and some glue and steering logic (in the form of a gate or switch-level netlist). Depending on the application, different constraints (e.g., performance, area, power, testability) must be satisfied during the various phases of the flow.

Manuscript received September 4, 1997; revised June 20, 1998. This paper was recommended by Associate Editor M. Papaefthymiou.

E. Macii is with the Politecnico di Torino, Torino 10129 Italy.

M. Pedram is with the Department of Electrical Engineering Systems, University of Southern California, Los Angeles, CA 90089 USA.

F. Somenzi is with the University of Colorado, Boulder, CO 90309 USA.

Publisher Item Identifier S 0278-0070(98)08592-3.

When the target is a low-power application, the search for the optimal solution must include, at each level of abstraction, a "design improvement loop." In such a loop, a power analyzer/estimator (shown in gray in Fig. 1) ranks the various design, synthesis, and optimization options, and thus helps in selecting the one that is potentially more effective from the power standpoint. Obviously, collecting the feedback on the impact of the different choices on a level-by-level basis, instead of just at the very end of the flow (i.e., at the gate level), enables a shorter development time. On the other hand, this paradigm requires the availability of power estimators, as well as synthesis and optimization tools, that provide accurate and reliable results at various levels of abstraction.

In this paper, we review some of the techniques for high-level power modeling, estimation, and optimization that have appeared recently in the literature. In particular, we focus on the software, behavioral, and RT levels, since these are the areas where most of the research efforts have been concentrated in the last few years. On the other hand, we do not discuss traditional logic-level (and below) techniques, since this subject is out of the scope of this paper (the interested reader may refer to [1]–[4] for excellent surveys on this topic).

II. Modeling and Estimation

It has been pointed out in the introduction that the availability of level-by-level power analysis and estimation tools that are able to provide fast and accurate results are key for increasing the effectiveness of automatic design frameworks organized as shown in Fig. 1. We start this section with a concise description of techniques for software-level estimation (Section II-A). We then move to the behavioral level (Section II-B), where we discuss existing power-estimation approaches that rely on information-theoretic (Section II-B1), complexity-based (Section II-B2), and synthesis-based (Section II-B3) models. Last, we focus our attention to designs described at the RT level (Section II-C). This is the area where most of the research activity on power modeling and estimation has been concentrated in recent times; we cover two of the most investigated classes of methods, namely, those relying on regression-based models (Section II-C1) and on sampling-based models (Section II-C2).

As mentioned in the introduction, power-estimation techniques working below the RT level have reached a solid degree of maturity, since they have been studied for quite a long time now; therefore, we do not treat them in this paper.

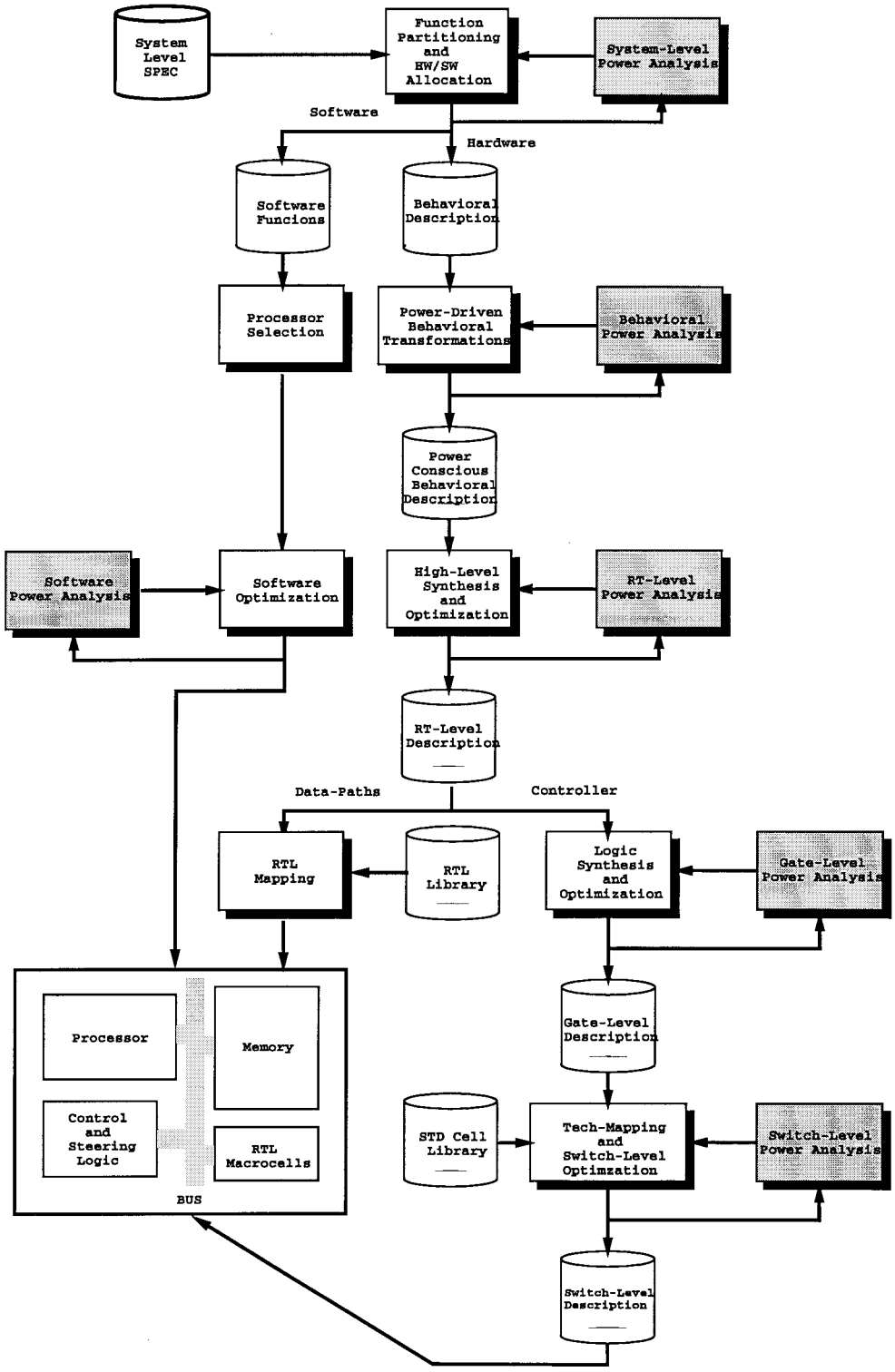


Fig. 1. Low-power design flow.

A. Software-Level Power Estimation

The first task in the estimation of power consumption of a digital system is to identify the typical application programs that will be executed on the system. A nontrivial application program consumes millions of machine cycles, making it nearly impossible to perform power estimation using the complete program at, say, the RT level. Most of the reported results are based on *power macro-modeling*, an estimation approach that is extensively used for behavioral and RT-level estimation (see Sections II-B and II-C).

In [5], the power cost of a CPU module is characterized by estimating the average capacitance that would switch when the given CPU module is activated. In [6], the switching activities on (address, instruction, and data) buses are used to estimate

the power consumption of the microprocessor. In [7], based on actual current measurements of some processors, Tiwari *et al.* present the following instruction-level power model:

$$\text{Energy}_p = \sum_i (BC_i N_i) + \sum_{i,j} (SC_{i,j} N_{i,j}) + \sum_k OC_k$$

where Energy_p is the total energy dissipation of the program, which is divided into three parts. The first part is the summation of the base energy cost of each instruction (BC_i is the base energy cost and N_i is the number of times instruction i is executed). The second part accounts for the circuit state ($SC_{i,j}$ is the energy cost when instruction i is followed by j during the program execution). The third part accounts for energy contribution OC_k of other instruction effects such as stalls and cache misses during the program execution.

In [8], Hsieh *et al.* present an approach, called *profile-driven program synthesis*, to perform RT-level power estimation for high-performance CPU's. Instead of using a macro-modeling equation to model the energy dissipation of a microprocessor, the authors use a synthesized program to exercise the microprocessor in such a way that the resulting instruction trace behaves (in terms of performance and power dissipation) much the same as the original trace. The new instruction trace is however much shorter than the original one and can hence be simulated on an RT-level description of the target microprocessor to provide the power-dissipation results quickly.

Specifically, this approach consists of the following steps.

1) Perform architectural simulation of the target microprocessor under the instruction trace of typical application programs.
2) Extract a *characteristic profile*, including parameters such as the instruction mix, instruction/data cache miss rates, branch prediction miss rate, pipeline stalls, etc., for the microprocessor.
3) Use mixed integer linear programming and heuristic rules to gradually transform a generic program template into a fully functional program.
4) Perform RT-level simulation of the target microprocessor under the instruction trace of the new synthesized program.

Notice that the performance of the architectural simulator in gate-vectors/second is roughly three to four orders of magnitude higher than that of an RT-level simulator.

This approach has been applied to the Intel Pentium processor (which is a superscalar pipelined CPU with 8-KB, two-way, set-associative data, instruction and data caches, branch prediction, and dual instruction pipeline) demonstrating three to five orders of magnitude reduction in the RT-level simulation time with negligible estimation error.

B. Behavioral-Level Power Estimation

Conversely, from some of the RT-level methods that will be discussed in Section II-C, estimation techniques at the behavioral level cannot rely on information about the gate-level structure of the design components and hence must resort to abstract notions of physical capacitance and switching activity to predict power dissipation in the design.

1) Information-Theoretic Models: Information-theoretic approaches for high-level power estimation [9], [10] depend on information-theoretic measures of activity (for example, entropy) to obtain quick power estimates.

Entropy characterizes the randomness or uncertainty of a sequence of applied vectors and thus is intuitively related to switching activity, that is, if the signal switching is high, it is likely that the bit sequence is random, resulting in high entropy. Suppose the sequence contains t distinct vectors and let p_i denote the occurrence probability of any vector v in the sequence. Obviously, $\sum_{i=1}^{t} p_i = 1$. The entropy of the sequence is given by

$$h = -\sum_{i=1}^{t} p_i \log p_i$$

where $\log x$ denotes the base 2 logarithm of x. The entropy achieves its maximum value of $\log t$ when $p_i = 1/t$. For an n-bit vector, $t \leq 2^n$. This makes the computation of the exact entropy very expensive. Assuming that the individual bits in the vector are independent, then we can write

$$h = -\sum_{i=1}^{n} (q_i \log q_i + (1 - q_i) \log(1 - q_i))$$

where q_i denotes the signal probability of bit i in the vector sequence. Note that this equation is only an upper bound on the exact entropy, since the bits may be dependent. This upper bound expression is, however, the one that is used for power-estimation purposes. Furthermore, in [9], it has been shown that, under the temporal independence assumption, the average switching activity of a bit is upper bounded by one-half of its entropy.

The power dissipation in the circuit can be approximated as

$$\text{Power} = 0.5 V^2 f C_{\text{tot}} E_{\text{avg}}$$

where C_{tot} is the total capacitance of the logic module (including gate and interconnect capacitances) and E_{avg} is the average activity of each line in the circuit, which is, in turn, approximated by one-half of its average entropy h_{avg}. The average line entropy is computed by abstracting information obtained from a gate-level implementation. In [10], it is assumed that the word-level entropy per logic level reduces quadratically from circuit inputs to circuit outputs, whereas in [9] it is assumed that the bit-level entropy from one logic level to the next decreases in an exponential manner. Based on these assumptions, two different computational models are obtained.

In [9], Marculescu *et al.* derive a closed-form expression for the average line entropy for the case of a linear gate distribution, i.e., when the number of nodes scales linearly between the number of circuit inputs n and circuit outputs m.

The expression for h_{avg} is given by

$$h_{\text{avg}} = \frac{2nh_{\text{in}}}{(n+m)\ln\dfrac{h_{\text{in}}}{h_{\text{out}}}} \cdot \left(1 - \frac{m}{n}\frac{h_{\text{out}}}{h_{\text{in}}} - \frac{\left(1-\dfrac{m}{n}\right)\left(1-\dfrac{h_{\text{out}}}{h_{\text{in}}}\right)}{\ln\dfrac{h_{\text{in}}}{h_{\text{out}}}}\right)$$

where h_{in} and h_{out} denote the average bit-level entropies of circuit inputs and outputs, respectively. h_{in} is extracted from the given input sequence, whereas h_{out} is calculated from a quick functional simulation of the circuit under the given input sequence or by empirical entropy propagation techniques for precharacterized library modules. In [10], Nemani and Najm propose the following expression for h_{avg}:

$$h_{\text{avg}} = \frac{2}{3(n+m)}(H_{\text{in}} + H_{\text{out}})$$

where H_{in} and H_{out} denote the average sectional (word-level) entropies of circuit inputs and outputs, respectively. The sectional entropy measures H_{in} and H_{out} may be obtained by monitoring the input and output signal values during a high-level simulation of the circuit. In practice, however, they are approximated as the summation of individual bit-level entropies h_{in} and h_{out}.

If the circuit structure is given, the total module capacitance is calculated by traversing the circuit netlist and summing up the gate loadings. Wire capacitances are estimated using statistical wire-load models. Otherwise, C_{tot} is estimated by quick mapping (for example, mapping to three-input universal gates) or by information-theoretic models that relate the gate complexity of a design to the difference of its input and output entropies. One such model proposed by Cheng and Agrawal in [11], for example, estimates C_{tot} as

$$C_{\text{tot}} = \frac{m}{n}2^n h_{\text{out}}.$$

This estimate tends to be too pessimistic when n is large; hence, in [12], Ferrandi *et al.* present a new total capacitance estimate based on the number N of nodes (i.e., two–to–one multiplexors) in the ordered binary decision diagram representation of the logic circuit as follows:

$$C_{\text{tot}} = \alpha\frac{m}{n}Nh_{\text{out}} + \beta.$$

The coefficients of the model are obtained empirically by doing linear regression analysis on the total capacitance values for a large number of synthesized circuits.

Entropic models for the controller circuitry are proposed by Tyagi in [13], where three entropic lower bounds on the average Hamming distance (bit changes) with state set S and with T states are provided. The tightest lower bound derived in this paper for a sparse finite-state machine (FSM) (i.e., $t \leq 2.23T^{1.72}/\sqrt{\log T}$, where t is the total number of transitions with nonzero steady-state probability), is the following:

$$\sum_{s_i, s_j \in S} p_{i,j}H(s_i, s_j) \geq h(p_{i,j}) - 1.52\log T - 2.16 + 0.5\log(\log T)$$

where $p_{i,j}$ is the steady-state transition probability from s_i to s_j, $H(s_i, s_j)$ is the Hamming distance between the two states, and $h(p_{i,j})$ is the entropy of the probability distribution $p_{i,j}$. Notice that the lower bound is valid regardless of the state encoding used.

2) Complexity-Based Models: These models relate the circuit power dissipation to some notion of *circuit complexity.* Example parameters that influence the circuit complexity include the number and the type of arithmetic and/or Boolean operations in the behavioral description, the number of states and/or transitions in a controller description, and the number of cubes (literals) in a minimum sum-of-products (factored-form) expression of a Boolean function.

Most of the proposed complexity-based models rely on the assumption that the complexity of a circuit can be estimated by the number of "equivalent gates." This information may be generated on-the-fly using analytical predictor functions or retrieved from a precharacterized high-level design library. An example of this technique is the *chip estimation system* [14], which uses the following expression for the average power dissipation of a logic module:

$$\text{Power} = fN(\text{Energy}_{\text{gate}} + 0.5V^2C_{\text{load}})E_{\text{gate}}$$

where f is the clock frequency, N is the gate equivalent count for the component, $\text{Energy}_{\text{gate}}$ is the average internal consumption for an equivalent gate (it includes parasitic capacitance contributions as well as short-circuit currents) per logic transition, C_{load} is the average capacitive load for an equivalent gate (it includes fanout load capacitances and interconnect capacitances), and E_{gate} is the average output activity for an equivalent gate per cycle. C_{load} is estimated statistically based on the average fanout count in the circuit and custom wire-load models. E_{gate} is dependent on the functionality of the module. The data are precalculated and stored in the library and are independent of the implementation style (static versus dynamic logic, clocking strategy), library-specific parameters (gate inertia, glitch generation, and propagation), and the circuit context in which the module is instantiated. This is an example of an implementation-independent and data-independent power-estimation model.

In [15], Nemani and Najm present a high-level estimation model for predicting the area of an optimized single-output Boolean function. The model is based on the assumption that the area complexity of a Boolean function f is related to the distribution of the sizes of the on-set and off-set of the function. For example, using the "linear measure," the area complexity of the on-set of f is written as

$$\mathcal{C}_1(f) = \sum_{i=1}^{N} c_i p_i$$

where the set of integers $\{c_1, c_2, \cdots, c_N\}$ consists of the distinct sizes of the essential prime implicants of the on-set and weight p_i is the probability of the set of all minterms in the on-set of f, which are covered by essential primes of size c_i but not by essential primes of any larger size. The area complexity of the off-set of $f\mathcal{C}_0(f)$ is similarly calculated. Hence, the area complexity of function f is estimated as

$$\mathcal{C}(f) = \frac{\mathcal{C}_1(f) + \mathcal{C}_0(f)}{2}.$$

The authors next derive a family of regression curves (which happen to have exponential form) relating the actual area $\mathcal{A}(f)$ of random logic functions optimized by the SIS program (in terms of the number of gates) to the area complexity measure $\mathcal{C}(f)$ for different output probabilities of function f. These regression equations are subsequently used for total capacitance estimation and hence high-level power estimation. The work is extended in [16] to area estimation of multiple-output Boolean functions.

A similar technique would rely on predicting the quality of results produced by EDA flows and tools. The predictor function is obtained by performing regression analysis on a large number of circuits synthesized by the tools and relating circuit-specific parameters and/or design constraints to postsynthesis power-dissipation results. For example, one may be able to produce the power estimate for an unoptimized Boolean network by extracting certain structural properties of the underlying directed acyclic graph, average complexity of each node, and user-specified constraints and plugging these values in the predictor function.

Complexity-based power-prediction models for controller circuitry have been proposed by Landman and Rabaey in [17]. These techniques provide quick estimation of the power dissipation in a control circuit based on the knowledge of its target implementation style (that is, precharged pseudo-NMOS or dynamic programmable logic array), the number of inputs, outputs, states, and so on. The estimates will have a higher degree of accuracy by introducing empirical parameters that are determined by curve fitting and least squared fit error analysis on real data. For example, the power model for an FSM implemented in standard cells is given by

$$\text{Power} = 0.5\, V^2 f (N_I C_I E_I + N_O C_O E_O) N_M$$

where N_I and N_O denote the number of external input plus state lines and external output plus state lines for the FSM, C_I and C_O are regression coefficients that are empirically derived from low-level simulation of previously designed standard cell controllers, E_I and E_O denote the switching activities on the external input plus state lines and external output plus state lines, and N_M denotes the number of minterms in an optimized cover of the FSM. Dependence on N_M indicates that this model requires a partial (perhaps symbolic) implementation of the FSM.

3) Synthesis-Based Models: One approach for behavioral-level power prediction is to assume some RT-level template and produce estimates based on that assumption. This approach requires the development of a *quick synthesis* capability, which makes some behavioral choices (mimicking a full synthesis program). Important behavioral choices include type of I/O, memory organization, pipelining issues, synchronization scheme, bus architecture, and controller design. This is a difficult problem, especially in the presence of tight timing constraints. Fortunately, designers or the environment often provide hints on what choices should be made. After the RT-level structure is obtained, the power is estimated by using any of the RT-level techniques that will be described in Section II-C.

Relevant data statistics such as the number of operations of a given type, bus and memory accesses, and I/O operations are captured by *static profiling* based on stochastic analysis of the behavioral description and data streams [18], [19] or *dynamic profiling* based on direct simulation of the behavior under a typical input stream [20], [21]. Instruction-level or behavioral simulators are easily adapted to produce this information.

C. RT-Level Power Estimation

Most RT-level power-estimation techniques use regression-based, switched-capacitance models for circuit modules. Such techniques, which are commonly known as *power macro-modeling,* are reviewed next.

1) Regression-Based Models: A typical RT-level power-estimation flow consists of the following steps.

1) Characterize every component in the high-level design library by simulating it under pseudorandom data and fitting a multivariable regression curve (i.e., the power macro-model equation) to the power-dissipation results using a least mean square error fit [22].
2) Extract the variable values for the macro-model equation either from static analysis of the circuit structure and functionality or by performing a behavioral simulation of the circuit. In the latter case, a power cosimulator linked with a standard RT-level simulator can be used to collect input data statistics for various RT-level modules in the design.
3) Evaluate the power macro-model equations for high-level design components, which are found in the library by plugging the parameter values in the corresponding macro-model equations.
4) Estimate the power dissipation for random logic or interface circuitry by simulating the gate-level description of these components [25], [26] or by performing probabilistic power estimation [27]–[31]. The low-level simulation can be significantly sped up by the application of statistical sampling techniques [32]–[35] or automata-based compaction techniques [36]–[38].

The macro-model for the components may be parameterized in terms of the input bit width, the internal organization/architecture of the component, and the supply voltage level. Notice that there are cases where the construction of the macro-model of step 1) can be done analytically using the information about the structure of the gate-level description of the modules, without resorting to simulation as proposed by Benini *et al.* in [23]. On the other hand, if the low-level

netlist of the library components is not known (which may be the case for soft macros), step 1) can be replaced by data collection from past designs of the component followed by appropriate process technology scaling [24]. In addition, the macro-model equation in step 2) may be replaced by a table lookup with necessary interpolation equations.

In the following paragraphs, we review various power macro-model equations, which exhibit different levels of accuracy versus computation/information usage tradeoff.

The simplest power macro-model, known as the *power factor approximation* technique [39], is a *constant type model* which uses an experimentally determined weighting factor to model the average power consumed by a given module per input change. For example, the power dissipation of an $n \times n$-bit integer multiplier can be written as

$$\text{Power} = 0.5\,V^2 n^2 C f_{\text{activ}}$$

where V is the supply voltage level, C is the capacitive regression coefficient, and f_{activ} is the activation frequency of the module (this should not be confused with the average, bit-level switching activity of multiplier inputs).

The weakness of this technique is that it does not account for the data dependency of the power dissipation. For example, if one of the inputs to the multiplier is always one, we would expect the power dissipation to be less than the case when both inputs are changing randomly. In contrast, the *stochastic power analysis* technique proposed by Landman and Rabaey in [40] is based on an activity-sensitive macro-model, called the *dual bit type model*, which maintains that switching activities of high-order bits depend on the temporal correlation of data, whereas lower order bits behave randomly. The module is thus completely characterized by its capacitance models in the sign and white noise bit regions. The macro-model equation form is then given by

$$\text{Power} = 0.5\,V^2 f\left(n_u C_u E_u + n_s \sum_{xy=++}^{--} C_{xy} E_{xy}\right)$$

where C_u and E_u represent the capacitance coefficient and the mean activity of the unsigned bits of the input sequence, while C_{xy} and E_{xy} denote the capacitance coefficient and the transition probability for the sign change xy in the input stream. n_u and n_s represent the number of unsigned and sign bits in the input patterns, respectively. Note that E_u, E_{xy}, and the boundary between sign and noise bits are determined based on the applied signal statistics collected from simulation runs. Expanding this direction, one can use a *bitwise data model* as follows:

$$\text{Power} = 0.5\,V^2 f \sum_{i=1}^{n} C_i E_i$$

where n is the number of inputs for the module in question, C_i is the (regression) capacitance for input pin i, and E_i is the switching activity for the ith pin of the module. This equation can produce more accurate results by including, for example, spatio-temporal correlation coefficients among the circuit inputs. This will, however, significantly increase the number of variables in the macro-model equation, and thus the equation evaluation overhead.

Accuracy may be improved (especially for components with deep logic nesting, such as multipliers) by power macro-modeling with respect to both the average input and output activities (the *input–output data model*, that is

$$\text{Power} = 0.5\,V^2 f(C_I E_I + C_O E_O)$$

where C_I and C_O represent the capacitance coefficients for the mean activities of the input and output bits, respectively. The dual bit type model or the bitwise data model may be combined with the input–output data model to create a more accurate, but more expensive, macro-model form. Recently, in [41], the authors presented a three-dimensional-table, power macro-modeling technique that captures the dependence of power dissipation in a combinational logic circuit on the average input signal probability, the average switching activity of the input lines, and the average (zero-delay) switching activity of the output lines. The latter parameter is obtained from a fast functional simulation of the circuit. The paper also presents an automatic macro-model construction procedure based on random sampling principles. Note that the equation form and variables used for every module are the same (although the regression coefficients are different).

A parametric power model is described by Liu and Svensson in [42], where the power dissipation of the various components of a typical processor architecture, including on-chip memory, busses, local and global interconnect lines, H-tree clock net, off-chip drivers, random logic, and data path, are expressed as a function of a set of parameters related to the implementation style and internal architecture of these components. For example, consider a typical on-chip memory (a storage array of six-transistor memory cells), which consists of four parts: the memory cells, the row decoder, the column selection, and the read/write circuits. The power model for a cell array of 2^{n-k} rows and 2^k columns in turn consists of expressions for:

1) the power consumed by 2^k memory cells on a row during one precharge or one evaluation;
2) the power consumed by the row decoder;
3) the power needed for driving the selected row;
4) the power consumed by the column select part;
5) the power dissipated in the sense amplifier and the readout inverter.

For instance, the memory cell power [1) above] is given by

$$\text{Power}_{\text{memcell}} = 0.5\,V\,V_{\text{swing}} 2^k (C_{\text{int}} + 2^{n-k} C_{\text{tr}})$$

where V_{swing} is the voltage swing on the bit/$\overline{bit}$ line (which may be different for read versus write), C_{int} gives the wiring-related row capacitance per memory cell, and $2^{n-k}C_{\text{tr}}$ gives the total drain capacitances on the bit/$\overline{bit}$ line. Notice that during the read time, every memory cell on the selected row drives exactly bit or $\overline{bit}$.

A salient feature of the above macro-model techniques is that they only provide information about average power consumption over a relatively large number of clock cycles. The above techniques, which are suitable for estimating the

average-power dissipation, are referred to as *cumulative* power macro-models. In some applications, however, estimation of average power only is not sufficient. Examples are circuit reliability analysis (maximum current limits, heat dissipation and temperature gradient calculation, latchup conditions), noise analysis (resistive voltage drop and inductive bounce on power and ground lines), and design optimization (power/ground net topology design, number and placement of decoupling capacitors, buffer insertion, etc.). In these cases, *cycle-accurate* (*pattern-accurate*) power estimates are required.

Mehta *et al.* propose a clustering approach for pattern-accurate power estimation in [43]. This approach relies on the assumption that closely related input transitions have similar power dissipation. Hence, each input pattern is first mapped into a cluster, and then a table lookup is performed to obtain the corresponding power estimates from precalculated and stored power characterization data for the cluster. The weakness of this approach is that, for efficiency reasons, the number of clusters has to be relatively small, which would introduce errors into the estimation result. In addition, the assumption that closely related patterns (e.g., patterns with short Hamming distance) result in similar power distribution may be quite inaccurate, especially when the *mode-changing bits* are involved, i.e., when a bit change may cause a dramatic change in the module behavior.

Addressing these problems, Wu *et al.* describe in [44] an automatic procedure for cycle-accurate macro-model generation based on statistical sampling for the *training set* design and regression analysis combined with appropriate statistical tests (i.e., the F^* test) for macro-model variable selection and coefficient calculation. The test identifies the most (least) power-critical variable to add to (delete from) the set of selected variables. The statistical framework enables prediction of the power value and the confidence level for the predicted power value. This work is extended by Qiu *et al.* in [45] to capture "important" first-order temporal correlations and spatial correlations of up to order three at the circuit inputs. Note that here the equation form and variables used for each module are unique to that module type. Experimental results show that power macro-models with a relatively small number of input variables (i.e., eight) predict the module power with a typical error of 5–10% for the average power and 10–20% for the cycle power.

2) Sampling-Based Models: RT-level power evaluation can be implemented in the form of a *power cosimulator* for standard RT-level simulators. The cosimulator is responsible for collecting input statistics from the output of the behavioral simulator and producing the power value at the end. If the cosimulator is invoked by the RT-level simulator every simulation cycle to collect activity information in the circuit, it is called *census macro-modeling*.

Evaluating the macro-model equation at each cycle during the simulation is actually a census survey. The overhead of data collection and macro-model evaluation can be high. To reduce the run-time overhead, Hsieh *et al.* use *simple random sampling* to select a sample and calculate the macro-model equation for the vector pairs in the sample only [46]. The sample size is determined before simulation. The *sampler macro-modeling* randomly selects n cycles and marks those cycles. When the behavioral simulator reaches the marked cycle, the macro-modeling invokes the behavioral simulator for the current input vectors and previous input vectors for each module. The input statistics are only collected in these marked cycles. Instead of selecting only one sample of large size, we can select several samples of at least 30 units (to insure normality of sample distribution) before the simulation. Then the average value of sample means is the estimate of population mean. In this manner, the overhead of collecting input statistics at every cycle, which is required by census macro-modeling, is substantially reduced. Experimental results show that sampler macro-modeling results in an average efficiency improvement of $50\times$ over the census macro-modeling, with an average error of 1%.

The macro-model equation is developed by using a training set of input vectors. The training set satisfies certain assumptions such as being pseudorandom data, speech data, etc. Hence, the macro-model may become biased, meaning that it produces very good results for the class of data that behave similarly to the training set; otherwise, it produces poor results. One way to reduce the gap between the power macro-model equation and the gate-level power estimation is to use a *regression estimator* as follows [46]. It can be shown that the plot of the gate-level power value versus a well-designed macro-model equation estimate for many functional units reveals an approximately linear relationship. Hence, the macro-model equation can be used as a predictor for the gate-level power value. In other words, the sample variance of the ratio of gate-level power to macro-model equation power tends to be much smaller than that of the gate-level power by itself. It is thus more efficient to estimate the mean value of this ratio and then use a linear regression equation to calculate the mean value of the circuit-level power. The *adaptive macro-modeling* thus invokes a gate-level simulator on a small number of cycles to improve the macro-model equation estimation accuracy. In this manner, the "bias" of the static macro-models is reduced or even eliminated. Experimental results show that the census macro-modeling incurs large error (an average of 30% for the benchmark circuits) compared to gate-level simulation. The adaptive macro-modeling, however, exhibits an average error of only 5%, which demonstrates the superiority of the adaptive macro-modeling technique.

III. Synthesis and Optimization

Power constraints must be taken into account during various phases of the design flow. In this section, we first focus on software optimization techniques (Section III-A), followed by system-level power-management strategies (Section III-B). In Section III-C, we illustrate transformations that are applicable to behavioral descriptions and that improve the potential savings achievable during the subsequent high-level synthesis phase. Algorithms for low-power operation scheduling and resource allocation (which are at the core of high-level synthesis tools) are discussed in Sections III-D and III-E, respectively, while a procedure for multiple supply-voltage scheduling is presented in Section III-F. The output of the

high-level synthesis phase is an RT-level description consisting of a (possibly partitioned) control unit and some computing (i.e., data-path) units, on which the bus encoding schemes summarized in Section III-G can be applied to reduce the overall power budget. A methodology to be used for translating the specification of the system's controller (as generated by the high-level synthesis phase) into a gate-level netlist is briefly outlined in Section III-H. Last, we go over a few RT and gate-level logic shutdown techniques (Section III-I), as well as retiming transformations (Section III-J) that can be exploited to further reduce the total power requirements. We would like to point out that not all the various design, synthesis, and optimization steps indicated in the flow of Fig. 1 will be discussed in detail in the sequel, but only those for which innovative, as well as sufficiently reliable solutions have been proposed in recent years. For example, we do not deal with techniques for low-power hardware-software partitioning, since only a few, preliminary contributions have appeared in the literature [47]–[49]. Also, as stated in the introduction, we do not consider traditional logic and transistor-level techniques.

A. Software Optimization

The software domain offers a large variety of opportunities for optimizing the power dissipation of a processor-based digital system. Software design for low power has thus become an active area of research in the last few years. In this section we summarize a few promising approaches. Specifically, we discuss techniques targeting power minimization through a) instruction scheduling and code generation and b) minimization of memory access costs.

The methods developed to properly select and order the instructions of a program to reduce the instruction bus activity are based on the simple observation that a given high-level operation (e.g., a C statement) can be compiled into different machine instruction sequences. Since the same observation is at the basis of code optimization for speed and size, the most straightforward way to proceed is to modify the objective function used by existing code optimizers to obtain low-power versions of a given software program. More specifically, the basic power cost of each instruction (determined *a priori* through a characterization process) must be considered during code optimization.

Though this approach has proved to be effective, more substantial power savings can be obtained by resorting to optimizations specifically addressing power minimization [7]. *Cold scheduling* is an instruction scheduling procedure proposed by Su *et al.* in [6] that attempts to reduce the number of instruction bus transitions occurring when the processor experiences a state change due to the execution of instructions of different types. In essence, the algorithm acts as a list scheduler that determines the priority of execution of the instructions according to their power cost. The method, though innovative, has been shown to work well only on processors with specific reduced instruction set computer architectures. A more articulated methodology for code generation and optimization, whose practical applicability has been demonstrated in the case of a digital signal processor (DSP), has been proposed by Lee *et al.* in [50]. In this solution, techniques such as instruction packing, minimization of circuit state effects, and operand swapping are exploited [51].

```
for (i=0;i< n;i++)                for (i=0;i< n;i++) {
   b[i] = f(a[i]);                   b[i] = f(a[i]);
for (i=0;i< n;i++)                   c[i] = g(b[i]);
   c[i] = g(b[i]);                }
```

Fig. 2. Code optimization to reduce the number of memory accesses.

Regarding the reduction of the costs of memory accesses, the most effective and straightforward way of obtaining it is through the minimization of the number of read/write operations required by an algorithm. Consider, as an example, the fragment of source code taken from [52] and shown on the left-hand side of Fig. 2. If we assume the size of array `b` to be too large to fit in the registers of the CPU, a total of `2n` read/write accesses to the memory are needed for the intermediate array `b` during the execution of the program. By transforming the code as indicated on the right-hand side of Fig. 2, the required element of array `b` can be kept into a register of the processor; therefore, only register accesses are necessary to store and load the intermediate data.

Since minimization of the number of memory accesses is one of the main objectives pursued by compilers that optimize programs for speed, existing techniques developed in the context of high-performance code generation can be easily adapted to reduce the power requirements of the software component of processor-based digital systems. However, further improvements in the power budget can be achieved by applying techniques (discussed next) that explicitly target the minimization of the switching activity on the address bus and that best exploit the hierarchy in the memory system.

The work by Panda and Dutt [53], [54] focuses on the reduction of the power dissipated by off-chip drivers and memory decoding logic by reducing the number of address bus transitions. The goal is reached through a memory mapping scheme that allows one to properly place in the main memory large arrays of data for which the access patterns can be extracted from the program source code at compilation time.

Additional contributions to the problem of finding data allocations that minimize the power in the memory-processor interface are available in the literature [52], [55]. These techniques have the same objective as some of the bus encoding strategies discussed in Section III-G. Therefore, in order to be effective, power minimization strategies should leverage their combination.

The basic assumption behind the exploitation of the memory hierarchy to reduce power is that, usually, the higher levels of the hierarchy can be accessed at a low power cost, but they have limited storage capacity (for example, cache versus RAM access). Power can then be reduced by organizing the data in such a way that the higher levels of the hierarchy are optimally utilized. Relevant work on this subject has been published in recent years by Catthoor *et al.* [52], [56], [57]. Their emphasis is on systems for DSP and video applications, where the power dissipated at the memory interface usually dominates, and the type of data to be manipulated is usually much simpler to

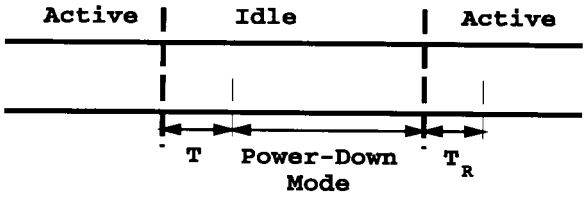


Fig. 3. Static shutdown strategy.

predict. They present a formalized methodology for the choice of the proper memory hierarchy to be adopted in the design of data-intensive systems.

B. System-Level Power Management

The activity of several components in a computing system is *event driven;* for example, the activity of display servers, communication interfaces, and user interface functions is triggered by external events and it is often interleaved with long periods of quiescence. An intuitive way of reducing the average power dissipated by the whole system consists of shutting down the resources during their periods of inactivity. In other words, one can adopt a system-level power-management policy that dictates how and when the various components should be shut down.

In [58], Srivastava *et al.* review conventional power-management approaches, such as those already in use in current portable computers, and propose some innovative schemes.

An event-driven computing device can be thought of as a finite-state system that can be in two states: *Active* and *Idle*. When the device is idle, it is desirable to shut it down by lowering its power supply or by turning off its clock; in this way, its power dissipation can be drastically reduced. If we call T_A and T_I the average time spent by the device in the *Active* and in the *Idle* states, respectively, we have that the maximum power improvement achievable through shutdown is given by $1 + T_I/T_A$. Improvement figures closer to the upper bound are, however, rarely obtained by existing shutdown strategies (called *static*). In fact, normally a device is put in its power-down mode only T time units after it has entered the *Idle* state (see Fig. 3). This is because it is assumed that there is a high chance for the system to be idle for a much longer time if it has been in the *Idle* state for at least T time units.

Obviously, this simple policy is not efficient for three reasons. First, the assumption that if the system is idle for more than T time units, it will be so for much longer may not be true in many cases. Second, even if the above assumption is valid in the majority of the cases, whenever the system enters the *Idle* state, it stays powered for at least T time units, wasting a considerable amount of power in that period. Third, speed and power degradations due to shutdowns performed at inappropriate times are not taken into account. In fact, it should be kept in mind that the transition from power-down to fully functional mode has an overhead: it takes some time T_R to bring the system up to speed, and it may also take more power than the average, steady-state power.

To overcome the limitations of the static shutdown policy discussed above, Srivastava *et al.* have proposed a *predictive* power-management strategy, whose main feature exploits the past history of the active and idle intervals to predict the length of an idle interval as soon as the system enters the *Idle* state. In practice, two approaches are suggested: one is based on obtaining a regression equation that predicts the value of T_I based on a quadratic function of the previous values of both T_A and T_I. The other is based on the simple observation that if the T_A immediately preceding a T_I is shorter than the minimum value of T_A ever experienced, it is highly probable that the next T_I will be longer than the minimum time for which it is convenient to shut down the system.

Obviously, the power-management mechanism is constrained by two factors: 1) the time overhead needed to restart the system and 2) the power overhead paid in restarting the system. The higher these two factors, the more conservative the shutdown strategy must be.

An experimental investigation performed on a SunSPARC station running an X server has shown power improvements achievable through the predictive shutdown policies to be as high as 38×, with a very limited decrease in performance (around 3%).

In [59], Hwang and Wu have introduced a more complex predictive shutdown strategy that performs better than the methods of Srivastava *et al.* The use of a technique for correcting possible idle period mispredictions, along with a prewakeup mechanism, account for the higher efficiency and the decreased delay penalty provided by the new approach.

It is important to point out that the applicability of power-optimization techniques based on resource shutdown is not limited to system-level descriptions. We will show later in the paper how the concept of power management can be successfully exploited during high-level synthesis and RTL optimization.

C. Behavioral Transformations

Given a *control-data-flow graph* (CDFG) describing the behavior of the hardware part of the system being designed, some transformations can be applied to it in order to improve the potential power savings achievable in the subsequent phases of high-level synthesis and RTL optimization. To be applicable in practice, such transformations must only modify the computational structure of the selected algorithm, while they must preserve its original input/output behavior and, to some extent, its latency.

According to Chandrakasan *et al.* [18], there are two distinct ways of optimizing power using behavioral transformations. The first one consists of enabling the reduction of the supply voltage through application of speedup transformations, such as retiming, pipelining, algebraic manipulations, and loop restructuring. Since these transformations have been extensively used in the context of performance optimization, we do not discuss them here. Instead, we focus on the second behavioral-level optimization approach. Here, the target is the minimization of the effective capacitance through transformations that increase the utilization of the system resources; this is because fewer and smaller computing elements usually provide better power performance of the design being developed. As an

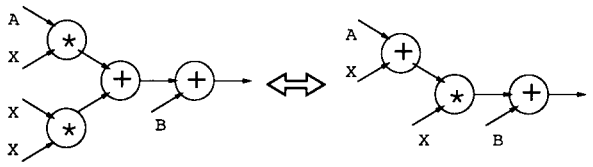


Fig. 4. Evaluation of a second-order polynomial.

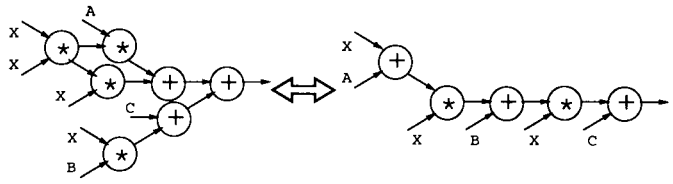


Fig. 5. Evaluation of a third-order polynomial.

example, we illustrate how a reduction of the total number of operations in the CDFG or the substitution of some operations with more convenient ones can yield more power-efficient descriptions.

The easiest way to reduce the total switched capacitance consists of reducing the number of operations in the CDFG. Unfortunately, reducing the number of operations may adversely affect system performance. In the following, we present two examples, taken from [18], which illustrate the contradictory effects that this transformation may have on the design under optimization.

Fig. 4 shows two possible implementations of a system that evaluates a second-order polynomial. The one on the left is the most straightforward: it requires a total of two adders and two multipliers, and it has a critical path of length three. The realization on the right, on the other hand, is obtained through simple algebraic transformations. It only consists of two adders and one multiplier, and it still has a critical path of length three. Obviously, in this case, the transformed structure is advantageous.

Consider now two different implementations, depicted in Fig. 5 and taken from [18], of a system that evaluates a third-order polynomial. The straightforward implementation on the left requires a total of three adders and four multipliers, and it has a critical path of length four. Algebraic transformations yield the implementation on the right, which contains only three adders and two multipliers, but which has a critical path of length five. In this case, a decrease in the number of operations corresponds to a decrease in speed, which, in turn, causes a reduction in the potential power optimization achievable through supply voltage downscaling.

It is well known that there exist operations whose corresponding hardware implementations require less energy per computation than others. For example, multiplications usually require more energy than additions. Therefore, *strength reduction* transformations are used to substitute multipliers with adders/subtractors, whenever possible. Unfortunately, this technique has a serious drawback: it usually produces an increase in the original critical path length. The conversion of multiplications with constants into the combination of shift and add operations is another powerful transformation belonging to this category. Its applicability is mainly found in DSP circuits, where constant multiplications are quite common.

TABLE I
CAPACITANCE STATISTICS FOR A TAP FIR FILTER

Component	Before constant mult.		After constant mult.	
	Switched cap. (pF)	% of total cap.	Switched cap. (pF)	% of total cap.
Execution units	739.65	64.80	93.07	21.63
Registers/clock	179.57	15.73	161.40	37.50
Control logic	65.45	5.73	83.79	19.47
Interconnect	156.69	13.74	92.10	21.40
Total	1141.36	100.00	430.36	100.00

As an example of the usefulness of this transformation, in Table I we report the capacitance statistics, taken from [18], for a Tap finite-duration impulse response (FIR) filter before and after application of the conversion of multiplications with constants into shift-add operations.

The capacitance switched by the control units is reduced by approximately a factor of eight; reductions are also achieved for the registers, the clock distribution network, and the interconnect network, mainly due to the reduced area of the final implementation. On the contrary, a small capacitance penalty is paid for the control logic.

The impact of various transformations on the characteristics of the design, depending on the specific situations in which they are applied, is such that a fully automatic procedure that drives the optimization process does not seem to be of practical interest. On the contrary, tools that help the designer in selecting the most useful transformations by quickly proposing the possible alternatives are highly desirable.

D. Operation Scheduling

The goal of a scheduling algorithm is to associate each primitive operation appearing in the CDFG with the time interval (also called control step) in which the operation is to be executed so as to satisfy some design constraints.

Several attempts have been made to modify traditional scheduling algorithms to take into account power consumption. For example, in [60], Musoll and Cortadella have proposed to include in the cost function that drives the scheduling procedure a measure of the switching activity occurring at the inputs of the functional units. By selecting from the CDFG the nodes for which no change of values in the input operands occurs between consecutive operations of the same functional unit, and by placing such nodes as close as possible in the scheduling, a substantial minimization of the total switched capacitance can be achieved. In [61], the same authors have also proposed a set of CDFG transformations that may help in minimizing the activity at the inputs of a functional unit. In particular, the use of *loop interchange*, *operand reordering*, and *operand sharing* has been suggested.

Clearly, the CDFG transformations mentioned above perform best in the cases where common input operands can be identified. Unfortunately, these situations are not encountered very frequently in real designs. However, it is still possible to target a switching activity reduction at the inputs of the functional units by resorting to the *power-conscious*

loop folding technique presented by Kim and Choi in [62]. Such technique, derived from a well-known transformation traditionally applied for throughput optimization and resource minimization, enables the detection of common input operands that are hidden inside the loops of the CDFG. The method has proven to have significant power-reducing effects on several applications taken from the DSP domain (e.g., filters).

A substantially different approach to the problem of determining a low-power scheduling of the CDFG has been introduced by Monteiro *et al.* in [63]. This work is based on the idea of enabling, at a lower level of abstraction, a power-management strategy similar to those discussed in Section III-B for system-level design descriptions.

The proposed scheduling algorithm attempts to assign the operations involved in determining and controlling the flow of the data within the system to the earliest possible time intervals. This allows one to establish which computational units are strictly required for a specific computation. The unused resources can be disabled during the system execution. They are identified by detecting mutually exclusive operations in the CDFG and by scheduling them for execution in time frames occurring after the decision on which unit must be activated has been made. In this way, all mutually exclusive units but one are guaranteed to be shut down during the current computation. In addition, if mutually exclusive operations are scheduled in the same time interval, it may be possible to share the corresponding resource, thus possibly achieving further power savings.

A scheduling that enables dynamic power management can be computed as follows. The multiplexors in the CDFG are considered individually, one at a time, starting with the ones that are closer to the bottom of the graph. Clearly, this is an arbitrary choice, and it is made in view of the fact that applying power management to such multiplexors may enable the shutdown of a larger number of units. The set of nodes N_0, N_1, and N_C of the CDFG, which belong to the transitive fanin of the zero, one, and control inputs of the currently selected multiplexor, are identified. Nodes that are simultaneously in N_0 and N_1 are obviously not suitable for power management, since the corresponding operation is needed no matter what the value of the multiplexor control input will be; therefore, they can be removed from the sets they belong to. The as-soon-as-possible (ASAP) scheduling algorithm is then run on the remaining nodes of N_0 and N_1, assuming that such nodes are assigned to time intervals that follow the one assigned to the last node in N_C. Similarly, the nodes in N_C are scheduled using the as-late-as-possible (ALAP) strategy, assuming that they are all associated with control steps preceding the one of the first nodes in either N_0 or N_1. If there exists at least one node in N_0, N_1, and N_C for which the time interval assigned by the ASAP algorithm is greater than the one assigned by the ALAP procedure, such a node cannot be scheduled under the required assumptions. Therefore, the multiplexor under consideration is not power manageable. On the other hand, if no conflict happens on the values of the time intervals assigned by the ASAP/ALAP scheduling procedures, all the nodes in the three sets are assigned the newly computed ASAP and ALAP control steps. The process just outlined is iterated over all the multiplexors. Upon completion (i.e., after having selected the multiplexor nodes for which the power management is possible), new precedence edges are created in the CDFG between the last node belonging to set N_C and the top nodes belonging to sets N_0 and N_1 for each of the selected multiplexors. (A precedence edge entering a node controls the activation of such a node.) The control step assignment phase is finally completed using an existing scheduling algorithm.

E. Resource Allocation

Once the scheduling is complete, a resource-allocation procedure must be run to assign registers and functional units to variables and operations in the scheduled CDFG, respectively, and to specify the interconnection of the various resources in terms of buses and multiplexors.

There are three classes of resources to be considered, namely, registers, functional units, and interconnections. Traditionally, the allocation has been carried out separately, one class of resources at a time (*serial* allocation). Usually, the power consumed by a resource mainly depends on the input switching activity induced by the data being stored or processed. Since, in reality, the patterns flowing through a circuit may have specific probability distributions, the way registers and functional units are allocated in the CDFG may heavily impact the switching activities at the interfaces of the resources. Graph-based algorithms for register allocation for nonpipelined designs [64] and module allocation for functionally pipelined designs [19] proposed by Chang and Pedram rely on an accurate computation of the probability density functions at the inputs of the various resources, given the probability distributions for the system primary inputs.

Unfortunately, in some cases, serial allocation may result in suboptimal solutions, i.e., designs using more interconnections than required. It may then be convenient to perform the three operations concurrently (*simultaneous* allocation). The technique of [65], proposed by Raghunathan and Jha and described next, considers data-dominated designs and targets a combined minimization of the total circuit capacitance and the switching activities at the inputs of the registers and the functional modules.

The first objective is reached by limiting the total number of resources in the final design implementation and by keeping under control the required amount of steering logic and interconnect. The minimization of the input switching activities, on the other hand, is obtained through exploitation of the correlations that may exist between the data words traveling and being stored within the circuit.

The allocation procedure is based on the concept of compatibility graph (CG) [66]. The CG is an undirected weighted graph that has as many nodes as there are variables and operations in the CDFG. Edges in the CG connect pairs of compatible nodes, that is, nodes that can be mapped onto the same resources (registers, in the case of variables, and functional modules, in the case of operations). Edge weights reflect the potential savings that could be achieved in the archi-

tectural implementation of the system if the pairs of variables or operations connected by the edges were assigned to the same hardware resources. Let us indicate such edge weights as W_c (capacitance weights). When power consumption is the target of the optimization, the switching activities at the inputs of the various resources must be taken into account while building the CG. To do that, another set of edge weights, the W_s's (switching activity weights), is determined through high-level simulation of the CDFG. The W_s's represent the average number of bits that switch between pairs of compatible variables or operations, and they are used in conjunction with the W_c's to form the global edge weights, W's, of the compatibility graph

$$W = W_c \cdot (1 - W_s).$$

Notice that $(1 - W_s)$ is used instead of W_s, since the target is the minimization of the switching activity.

After the construction of the compatibility graph is terminated, the allocation algorithm iteratively merges pairs of compatible nodes, starting with the ones having higher global weights. Obviously, this merging operation corresponds to the mapping of the two variables or operations connected by the edges to the same resources.

The allocation and binding algorithm summarized above does not guarantee the minimum number of registers and functional modules in the final architecture; however, the result is usually very close to the optimum, as shown by a number of experiments, and power savings are between 5 and 33%.

To further reduce the overall power budget, power management of the available hardware resources can be enabled through a careful design of the control circuitry. In fact, not all the resources of a system are simultaneously active at all times. In particular, a component is idle when none of the variables or operations mapped to it is active. The inputs to idle registers and modules do not affect the behavior of the overall system. Therefore, it may be possible to specify some *don't care* conditions in the controller; this information may then be exploited to decrease the overall switching activity within the design.

Other low-power allocation strategies have been proposed in the recent literature. Some can be found in [60], [61], and [67]–[69].

As a concluding remark, we would like to point out that in the design flow of Fig. 1, operation scheduling is assumed to precede resource allocation. This may not always be the case in existing high-level synthesis tools, where the order of execution of the two phases may be reversed [70], [71]; also, it may happen that scheduling and allocation are performed *simultaneously*. If the latter is the case, the optimization problem must be formulated in a more global way, and the various aspects related to low-power design that we have separately discussed for scheduling and allocation must be combined. An algorithm that simultaneously performs low-power operation scheduling, clock selection, and resource allocation is described in [72].

F. Multiple Supply-Voltage Scheduling

Supplying different voltages to different parts of a chip may reduce the global energy requirements of a design at a very limited cost in terms of algorithmic and/or architectural modifications. This is because the modules of the chip that are part of the critical paths are powered at the maximum allowed voltage, thus avoiding any delay increase; the power consumed by the modules that are not on the critical paths, on the other hand, is minimized through proper voltage scaling.

The presence on the same chip of circuitry powered at different voltages imposes the use of level shifters at the boundaries of the various modules. Obviously, the area and power costs due to such shifters must be considered while evaluating the quality of the optimized circuit.

An important phase in the design flow of multipowered systems is that of assigning the most convenient supply voltage, selected from a fixed number of values, to each operation in the CDFG. The problem to be solved is then that of scheduling the supply voltages so as to minimize the power dissipation under throughput/resource constraints.

An effective solution has been proposed by Chang and Pedram in [73]. The technique is based on dynamic programming, and it requires the availability of accurate timing and power models for the macro-modules in the RTL library. A preliminary characterization procedure must then be run to determine an energy-delay curve for each module in the library and for all possible supply-voltage assignments. The points on the curve represent various voltage assignment solutions with different tradeoffs between the performance and the energy consumption of the cell. Each set of curves is stored in the RTL library, ready to be used by the cost function that controls the multiple supply-voltage scheduling algorithm, outlined next for the simple case of CDFG's with tree structure. It consists of two phases: first, a set of possible power-delay tradeoffs at the root of the tree is calculated; then, a specific macro-module is selected for each node in such a way that the scheduled CDFG meets the required timing constraints.

To compute the set of possible solutions, a power-delay curve at each node of the tree (proceeding from the inputs to the output of the CDFG) is computed; such a curve represents the power-delay tradeoffs that can be obtained by selecting different instances of the macro-modules, and the necessary level shifters, within the subtree rooted at each specific node. The computation of the power-delay curves is carried out recursively, until the root of the CDFG is reached.

Given the power-delay curve at the root node, that is, the set of tradeoffs the user can choose from, a recursive preorder traversal of the tree is performed, starting from the root node, with the purpose of selecting which module alternative should be used at each node of the CDFG.

Upon completion, all the operations are fully scheduled; therefore, the CDFG is ready for the resource-allocation step for which the techniques presented in Section III-E can be used. Multipowered scheduling for high-throughput, functionally pipelined designs is also addressed in [73].

Alternatives to the multiple supply-voltage scheduling approach discussed above do exist in the literature. The interested reader may find them in [74]–[76].

G. Bus Encoding

It is known that bus capacitances are usually several orders of magnitude higher than those of the internal nodes of a circuit. Consequently, a considerable amount of power can be saved by reducing the number of transitions at the circuit input/output interfaces. This task can be accomplished by encoding the information transmitted over the buses.

The *Bus-Invert code* of [77] is a simple, yet effective, low-power encoding scheme. It works as follows: the Hamming distance between two successive patterns is computed; if it is larger than $N/2$, where N is the bus width, the current address is transmitted with inverted polarity; otherwise, it is transmitted as is. Obviously, a redundant bus line *INV* is needed to signal to the receiving end of the bus which polarity is used for the transmission of the incoming pattern. The method guarantees a maximum of $N/2$ transitions per clock cycle, and it performs well when the patterns to be transmitted are randomly distributed in time and no information about their correlation is available. For this reason, it is appropriate for data-bus encoding.

Concerning address buses, other techniques have also been explored. Since the addresses generated by processors in ordinary computing systems are often consecutive, Su *et al.* have suggested the adoption of the *Gray code* [78] as encoding strategy. This code achieves its asymptotic best performance of a single transition per emitted address when infinite streams of consecutive addresses are considered [79], and it is optimum only in the class of irredundant codes. If some redundancy is allowed, as for the Bus-Invert approach, better performance can be achieved by resorting to the T0 *code* [80], which requires an extra line *INC* to signal when a pair of consecutive addresses is written to the bus. When *INC* is high, the current bus value is frozen to avoid unnecessary switchings, and the new address is computed directly by the receiver. On the other hand, when two addresses are not consecutive, the *INC* line is low, and the bus operates normally. Several variants of the T0 code are possible, some of which may incorporate the Bus-Invert principle to exploit distinctive spectral characteristics of the streams being transmitted [81].

The high frequency of consecutive patterns in the address streams is at the basis of the effectiveness of encoding mechanisms such as Gray and T0. Clearly, if the percentage of in-sequence addresses decreases, their effectiveness diminishes as well. Two recently proposed solutions tackle some of the limitations of Gray and T0.

The *working zone* code [82] is based on the observation that many programs access multiple data arrays. The accesses to each array are mainly in sequence, but unfortunately they are often interleaved; then, the sequentiality on the bus is destroyed. The working-zone scheme restores sequentiality by storing the reference addresses of each working zone on the receiver side and by sending only the highly sequential offsets. Whenever the data access moves to a new working zone, this information is communicated to the receiver with a special code word. The receiver changes the default reference address, and offset transmission can resume. Although this scheme is more flexible than Gray and T0, it still relies on strong assumptions on the patterns in the stream. If the data-access policy is not array based, or if the number of working zones is too large, this encoding scheme loses effectiveness. Moreover, similar to the case of the T0 code, it requires one extra bus wire for communicating a working-zone change. This requirement might not be acceptable because it changes standard bus widths and chip pinouts.

The *Beach code* [83] relies on the fact that other types of temporal correlations than arithmetic sequentiality exist between the patterns that are being transmitted over the address bus. Since it has been experimentally noted that time-adjacent addresses normally show remarkably high block correlations, the idea is that of determining an encoding strategy that depends on the particular stream being transmitted. Given a typical execution trace of the address bus to be encoded, some statistical information identifying possible block correlations is collected. The bus lines are then grouped into clusters according to their correlations, that is, lines belonging to the same cluster are highly correlated. An encoding function is automatically generated for each cluster, and each configuration of bits in the original cluster is translated into a new bit configuration. The algorithm that finds the various encoding functions targets the minimization of the switching activity; thus, the technology developed for low-power finite state machine encoding (see Section III-H) can be successfully exploited. The output of the transformation is an encoded stream for which the average number of bus line transitions between two successive patterns is minimized. Clearly, since the computation of the encoding functions is strictly dependent on the selected execution trace, the Beach code performs best on special-purpose systems, where a dedicated processor (e.g., core, DSP, microcontroller) repeatedly executes the same portion of embedded code.

The motivation for adopting a bus-encoding scheme is a reduction of the global power budget; then, the savings achieved through a bus switching activity reduction must not be offset by the power dissipated by the encoding and decoding circuitry at the bus terminals. In addition, bus latency is usually a critical design constraint. Simultaneous optimization of power and timing must then be targeted while synthesizing the logic for bus encoding/decoding [81].

H. Control Logic Synthesis and Optimization

High-level synthesis produces a combined description of data-path and control logic. The latter is normally in the form of a transition structure, whose most familiar representation is an FSM or a collection of FSM's. The translation of such FSM's into a structural description presents opportunities for reducing power consumption and poses corresponding challenges, especially when the control is complex and contains a large number of latches. In this section, we outline how a gate-level netlist, suitable as input to logic-level optimization techniques, can be synthesized from a state transition graph (STG).

The synthesis process starts with the extraction of the STG from the RTL description of the FSM. For controllers with more than a handful of latches, the explicit representation of the STG is infeasible. Though decomposition of the controller before synthesis may alleviate the problem, optimization opportunities may be lost in the process. For this reason, symbolic techniques based on *binary decision diagrams* (BDD's) [84] are often applied to the manipulation of large graphs. These techniques represent sets by their characteristic functions and use BDD's to represent characteristic functions. To be effective, symbolic algorithms must avoid explicit enumeration of the elements of the sets (e.g., the edges of a graph).

Since BDD's are used to represent the transition relation of the graph, a preliminary *encoding* of the states is required. This is often derived heuristically from the behavioral description. The graph is then subjected to various transformations intended to improve energy efficiency as well as other metrics. Last, a detailed structural description must be produced from the graph.

Given the STG of the circuit controller, the optimization task consists of modifying and encoding the graph in preparation for logic synthesis. We review these techniques with particular emphasis on those algorithms that can be applied to large circuits. (Those that dissipate nonnegligible amounts of energy.) Among the modifications are *decomposition* and *restructuring*. Decomposition techniques produce interconnected FSM's from one large FSM, and they fall broadly into two categories: those based on the algebraic theory of [85] and those based on the identification in the STG of subroutines or coroutines [86]. A subroutine/coroutine corresponds to a fragment of the STG augmented with a wait state. Shutdown techniques can be applied to the individual machines because only one is active at any point in time [87]. Both approaches to decomposition try to minimize the activity along the lines connecting the submachines, which tend to drive heavier loads. Decomposition naturally helps tackling the complexity issue; however, no decomposition algorithms are currently available that are applicable to STG's with millions of states.

Restructuring of the STG is a generic term that encompasses those graph transformations that preserve equivalence of behavior (or compatibility in the presence of don't care conditions). The best known of such transformations is state minimization. Algorithms are available for the minimization of very large, completely specified FSM's [88]. However, state minimization by itself may have a deleterious effect on both area and energy efficiency, especially for large circuits. It is more advantageous to use the knowledge of the equivalence classes to identify don't care conditions and then use such conditions in conjunction with a cost function that accounts for the desired cost metrics [89].

The problem of encoding a state transition graph for low power consumption has received considerable attention. Among the earliest works is [90]. The idea common to this and other encoding methods (see, for example, [91]–[94]) is to use the transition probability of a given arc as a (partial) measure of its cost. The problem is thus translated into the embedding of the state transition graph into a hypercube of suitable dimension so that arcs of high cost connect states at low Hamming distance. Standard search techniques can be applied to this combinatorial optimization problem.

When the STG is large, it is normally given in an already encoded form. The problem is then the one of reencoding. The initial encoding may come from a manual design, and therefore it may provide a useful starting point. In general, however, it is not optimal from the power viewpoint. The main difference between algorithms for reencoding [95] and those for encoding is in the size of the problems they try to solve (millions of states versus thousands). To cope with very large graphs, BDD-based techniques are used to manipulate the graphs and sets of states; and the usual algorithms must be reformulated so as to avoid any explicit iteration over states or edges. The computation of the state probabilities can be carried out exactly [96] or by resorting to approximate techniques [31].

A direct translation of the optimized STG into gates should produce a structure that is relatively close to a good final solution. Otherwise, the successive synthesis algorithms are likely to produce suboptimal results. The problem when the transition relation is represented by a BDD is that the obvious mapping of each BDD node to a multiplexor results in networks that are large, deep, and slow. Among the approaches that overcome this problem, one builds a circuit in which transitions for a given input vector propagate along a single path, which corresponds to the selected path in the BDD; several optimizations are then applied to control the cost of the circuit [97].

Another approach is based on the work of Minato [98]. Zero-suppressed BDD's can represent very large function covers efficiently. Powerful factorization algorithms exist that work on these symbolic covers. It is therefore possible to first flatten the multilevel representation provided by the transition relation BDD and extract from the two-level cover a multilevel network. Factoring can be guided by low-power concerns, but the objective of the symbolic techniques is to provide a link to existing logic-level optimization tools, not to supplant them.

I. RT and Gate-Level Power Management

Dynamic power-management strategies such as those discussed in Sections III-B and III-E can be extended, with a finer degree of granularity, to the case of RT and gate-level descriptions. In fact, digital circuits usually contain portions that are not performing useful computations at each clock cycle. Power reductions can then be achieved by shutting down the circuitry when it is idle. In this section, we briefly outline three techniques for automatically inserting dynamic power-management mechanisms into RT and gate-level designs.

Precomputation [99], [100] relies on the idea of duplicating part of the logic with the purpose of precomputing the circuit output values one clock cycle before they are required, and then uses these values to reduce the total amount of switching in the circuit during the next clock cycle. In fact, knowing the output values one clock cycle in advance allows the original logic to be turned off during the next time frame, thus eliminating any charging and discharging of the internal capacitances. Obviously, the size of the logic that precalcu-

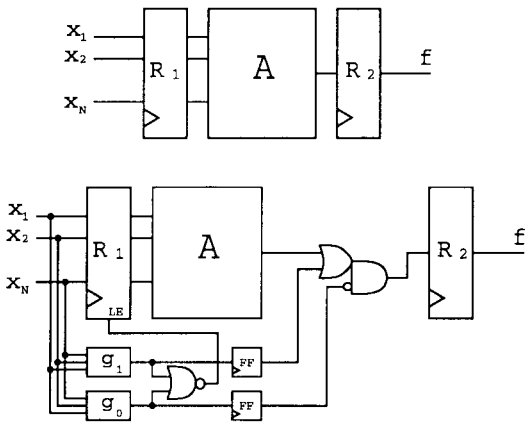


Fig. 6. Example of precomputation architecture.

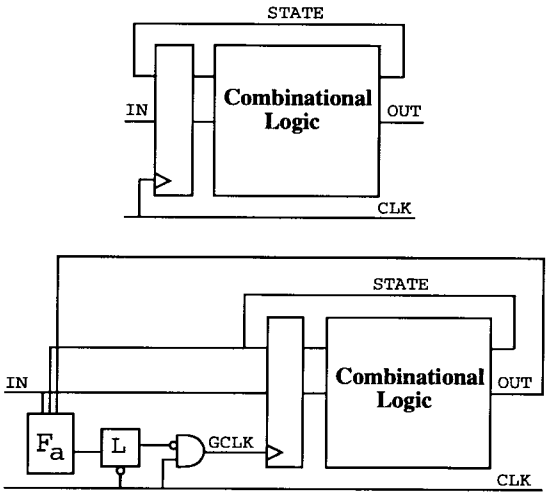


Fig. 7. Example of gated clock architecture.

lates the output values must be kept under control, since its contribution to the total power balance may offset the savings achieved by blocking the switching inside the original circuit. Several variants to the basic architecture can then be adopted to take care of this problem; in particular, sometimes it may be convenient to resort to partial, rather than global, shutdown, i.e., to select for power management only a (possibly small) subset of the circuit inputs.

As an example, consider the left part of Fig. 6; the combinational block A implements an N-input, single-output Boolean function f, and it has the I/O pins connected to registers R_1 and R_2. A possible precomputation architecture is depicted at the bottom of Fig. 6.

The key elements of the architecture are the two N-input, single-output predictor functions g_1 and g_0, whose behavior is required to satisfy the following constraints:

$$g_1 = 1 \Rightarrow f = 1$$
$$g_0 = 1 \Rightarrow f = 0.$$

The consequence is that, if at the present clock cycle either g_1 or g_0 evaluates to one, the load enable signal *LE* goes to zero, and the inputs to block A at the next clock cycle are forced to retain the current values. Hence, no gate output transitions inside block A occur, while the correct output value for the next time frame is provided by the two registers located on the outputs of g_1 and g_0.

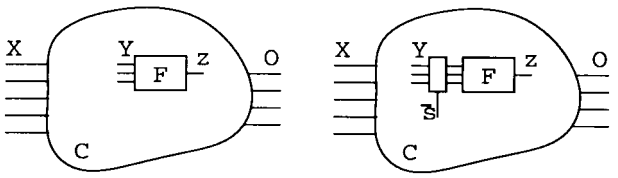
As mentioned earlier, the choice of the predictor functions is a difficult task. Perfect prediction requires $g_1 \equiv f$ and $g_0 \equiv f'$. However, this solution would not give any advantage in terms of power consumption over the original circuit, since it would entail the duplication of block A, and thus it would cause the same number of switchings as before but with an area twice as large as the original network. Consequently, the objective to be reached is the realization of two functions for which the probability of their logical sum (i.e., $g_1 + g_0$) to be one is as high as possible, but for which the area penalty due to their implementations is very limited. Also, the delay of the implementation of g_1 and g_0 should be given some attention, since the prediction circuitry may be on the critical path and, therefore, it may impact the performance of the optimized design.

Another approach to RT and gate-level dynamic power management, known as *gated clocks* [101]–[103], provides a way to selectively stop the clock, and thus force the original circuit to make no transition, whenever the computation to be carried out at the next clock cycle is useless. In other words, the clock signal is disabled in accordance to the idle conditions of the logic network. For reactive circuits, the number of clock cycles in which the design is idle in some wait states is usually large. Therefore, avoiding the power waste corresponding to such states may be significant.

As an example of use of the clock-gating strategy, consider the traditional block diagram of a sequential circuit, shown on the upper part of Fig. 7. It consists of a combinational logic block and an array of state registers that are fed by the next-state logic and that provide some feedback information to the combinational block itself through the present-state input signals. The corresponding gated-clock architecture is shown in the lower part of the picture.

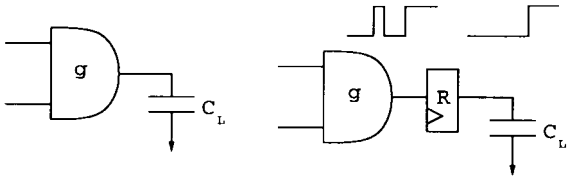
The circuit is assumed to have a single clock, and the registers are assumed to be edge-triggered flip-flops. The combinational block F_a is controlled by the primary inputs, the present-state inputs, and the primary outputs of the circuit, and it implements the activation function of the clock-gating mechanism. Its purpose is to selectively stop the local clock of the circuit anytime no state or output transition takes place. The block named L is a latch, transparent when the global clock signal CLK is inactive. Its presence is essential for a correct operation of the system, since it takes care of filtering glitches that may occur at the output of block F_a. It should be noted that the logic for the activation function is on the critical path of the circuit; therefore, timing violations may occur if the synthesis of F_a is not carried out properly.

The logic for the clock management is automatically synthesized from the Boolean function that represents the idle conditions of the circuit. It may well be the case that considering all such conditions results in additional circuitry that is too large and power consuming. It may then be necessary to synthesize a simplified function, which dissipates the minimum possible power and stops the clock with maximum efficiency.



(a) (b)

Fig. 8. Example of guard logic insertion.



(a) (b)

Fig. 9. Reducing the switching activity by inserting registers.

The use of gated clocks has the drawback that the logic implementing the clock-gating mechanism is functionally redundant, and this may create major difficulties in testing and verification. The design of highly testable gated clock circuits is discussed in [104].

Guarded evaluation [105] is the third RT and gate-level shutdown technique we review in this section. The distinctive feature of this solution is that, unlike precomputation and gated clocks, it does not require one to synthesize additional logic to implement the shutdown mechanism; rather, it exploits existing signals in the original circuit. The approach is based on placing some *guard logic*, consisting of transparent latches with an enable signal, at the inputs of each block of the circuit that needs to be power managed. When the block must execute some useful computation in a clock cycle, the enable signal makes the latches transparent. Otherwise, the latches retain their previous states, thus blocking any transition within the logic block.

The use of transparent latches as devices to eliminate useless node transitions is not new, since it has been proposed by Lemonds and Shetti in [106] for the handcrafted optimization of multipliers and other arithmetic circuits. However, the work by Tiwari *et al*. on guarded evaluation provides a systematic approach, described next, to identify where transparent latches must be placed within the circuit and by which signals they must be controlled.

Let C be a combinational logic block [shown in Fig. 8(a)], X be the set of primary inputs to C, and z be a signal in C. Also, let F be the portion of logic that drives z and Y be the set of inputs to F. Last, let $D_z(X)$ be the observability don't care set for z (that is, the set of primary input assignments for which the value of z does not influence the outputs of C). Consider a signal s in C that logically implies $D_z(X)$, that is, $\overline{s} + D_z(X) \equiv 1$. Then, if $s = 1$, the value of z is not required to compute the outputs of C. If we call $t_e(Y)$ the earliest time at which any input to F can switch when $s = 1$, and $t_l(s)$ the latest time at which s settles to one, we have that signal s can be used as the guard signal for F [shown in Fig. 8(b)] if $t_l(s) < t_e(Y)$. This is because z is not required to compute the outputs of C when $s = 1$, and thus block F can be shut down. Notice that the condition $t_l(s) < t_e(Y)$ guarantees that the transparent latches in the guard logic are shut down before any of the inputs to F makes a transition.

The technique described above, referred to as *pure guarded evaluation* in [105], has the desirable property that, when applied, no changes in the original combinational circuitry are needed. On the other hand, if some resynthesis and restructuring of the original logic is allowed, a larger number of logic shutdown opportunities may become available.

Other RTL power-management approaches exist in the literature. For example, in [107] and [108], Raghunathan *et al.* propose a technique that reduces switching activity through respecification of some of the control signals in such a way that both the multiplexor networks and the functional units get conveniently reconfigured. The strength of the proposed methodology, thought specifically for control-flow-intensive designs, can be augmented by applying RT-level transformations for glitch minimization [109]; in fact, in such kind of systems, the power dissipated by the control and steering circuitry is usually predominant with respect to the power required by the functional units.

J. RT and Gate-Level Retiming

The position of the registers within a design may greatly affect the area and performance of the circuit implementation. The transformation that repositions the registers of a design without modifying its external behavior is called retiming. The technique, initially proposed by Leiserson and Saxe [110], has found wide applications in the context of area and timing optimization. In [111], Monteiro *et al.* have pointed out that register positions can also affect power dissipation. Consider the simple example of a logic gate g belonging to a synchronous circuit [see Fig. 9(a)], and call C_L the capacitive load driven by the output node of g. In the case of CMOS technology, the power dissipated by gate g is proportional to the product of the switching activity of the output node of the gate E_g and the output load C_L. Now consider the case in which a register R is connected to the output of g. Let C_R be the input capacitance of the register, and let E_R be the switching activity of the register output [see Fig. 9(b)].

The total power dissipated by the new circuit is proportional to $E_g C_R + E_R C_L$. Since the output of the register can make, at most, one transition per clock cycle, we have that $E_R \leq E_g$. In fact, at the output of gate g some spurious transitions (i.e., glitches) may occur, but they are filtered by the register; hence, they do not propagate to the output of R. Consequently, it may happen that $E_g C_R + E_R C_L < E_g C_L$ if both E_g and C_L are sufficiently high. If this is the case, the presence of the register at the output of the gate has beneficial effects to the power behavior of the circuit.

Though sometimes it may be advantageous (for instance, in the case of pipelining, when registers are added to speed up a design), inserting registers into a design is not always feasible. On the other hand, when registers are already present in the circuit, it may be possible to move them across RTL blocks or logic gates so as to modify the circuit's timing—and, in view

of the discussion above, also its power dissipation—without affecting the behavioral characteristics.

The heuristic retiming technique of [111] applies to a synchronous network with pipeline structure. The basic idea is to select a set of *candidate* gates in the circuit such that if registers are placed at their outputs, the total switching activity of the network gets minimized. The selection of the gates is driven by two factors: the amount of glitching that occurs at the output of each gate and the probability that such glitching propagates to the gates located in the transitive fanout.

Registers are initially placed at the primary inputs of the circuit, and backward retiming (which consists of moving one register from all gate inputs to the output) is applied until all the candidate gates have received a register on their outputs. Then, registers that belong to paths not containing any of the candidate gates are repositioned, with the objective of minimizing both the delay and the total number of registers in the circuit. This last retiming phase does not affect the registers that have been already placed at the outputs of the previously selected gates.

IV. Conclusions

The increased degree of automation of industrial design frameworks has produced a substantial change in the way digital IC's are developed. The design of modern systems usually starts from specifications given at a very high level of abstraction. This is because existing EDA tools are able to automatically produce low-level design implementations directly from descriptions of this type.

It is widely recognized that power consumption has become a critical issue in the development of digital systems; then, electronic designers need tools that allow them to explicitly control the power budget during the various phases of the design process. This is because the power savings obtainable through automatic optimization are usually more significant than those achievable by means of technological choices (e.g., process and supply-voltage scaling).

In this paper, we have provided a nonexhaustive review of existing methodologies and tools for high-level power modeling and estimation, as well as for power-constrained synthesis and optimization. Such methodologies and tools are younger and, therefore, less developed than those available at the gate and circuit level. A wealth of research results and a few pioneering commercial tools have appeared nonetheless in the last couple of years. We expect this field to remain quite active in the foreseeable future. New trends and techniques will emerge, and some approaches described in this review will consolidate, while others will become obsolete; this is in view of technological and strategic changes in the world of microelectronics.

References

[1] F. N. Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Trans. VLSI Syst.*, vol. 2, no. 4, pp. 446–455, 1994.

[2] M. Pedram, "Power minimization in IC design: Principles and applications," *ACM Trans. Design Automat. Electron. Syst.*, vol. 1, no. 1, pp. 3–56, 1996.

[3] J. M. Rabaey and M. Pedram, Eds., *Low Power Design Methodologies.* Norwell, MA: Kluwer Academic, 1996.

[4] J. Mermet and W. Nebel, Eds., *Low Power Design in Deep Submicron Electronics.* Norwell, MA: Kluwer Academic, 1997.

[5] T. Sato, Y. Ootaguro, M. Nagamatsu, and H. Tago, "Evaluation of architectural-level power estimation for CMOS RISC processors," in *Proc. ISLPE-95: IEEE Int. Symp. Low Power Electronics*, San Jose, CA, Oct. 1995, pp. 44–45.

[6] C.-L. Su, C.-Y. Tsui, and A. M. Despain, "Low power architecture design and compilation techniques for high-performance processors," in *Proc. IEEE CompCon'94*, Feb. 1994, pp. 489–498.

[7] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step toward software power minimization," *IEEE Trans. VLSI Syst.*, vol. 2, no. 4, pp. 437–445, 1994.

[8] C.-T. Hsieh, M. Pedram, H. Mehta, and F. Rastgar, "Profile-driven program synthesis for evaluation of system power dissipation," in *Proc. DAC-34: ACM/IEEE Design Automation Conf.*, Anaheim, CA, June 1997, pp. 576–581.

[9] D. Marculescu, R. Marculescu, and M. Pedram, "Information theoretic measures for power analysis," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 6, pp. 599–610, 1996.

[10] M. Nemani and F. Najm, "Toward a high-level power estimation capability," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 6, pp. 588–598, 1996.

[11] K. T. Cheng and V. D. Agrawal, "An entropy measure for the complexity of multi-output Boolean functions," in *Proc. DAC-27: ACM/IEEE Design Automation Conf.*, Orlando, FL, June 1990, pp. 302–305.

[12] F. Ferrandi, F. Fummi. E. Macii, M. Poncino, and D. Sciuto, "Power estimation of behavioral descriptions," in *Proc. DATE-98: IEEE Design Automation and Test in Europe*, Paris, France, Feb. 1998, pp. 762–766.

[13] A. Tyagi, "Entropic bounds on FSM switching," *IEEE Trans. VLSI Syst.*, vol. 5, no. 4, pp. 456–464, 1997.

[14] K. Muller-Glaser, K. Kirsch, and K. Neusinger, "Estimating essential design characteristics to support project planning for ASIC design management," in *Proc. ICCAD-91: IEEE/ACM Int. Conf. Computer Aided Design*, Santa Clara, CA, Nov. 1991, pp. 148–151.

[15] M. Nemani and F. Najm, "High-level area prediction for power estimation," in *Proc. CICC-97: Custom Integrated Circuits Conf.*, Santa Clara, CA, May 1997, pp. 483-486.

[16] ——, "High-level area and power estimation for VLSI circuits," in *Proc. ICCAD-97: IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1997, pp. 114–119.

[17] P. Landman and J. Rabaey, "Activity-sensitive architectural power analysis for the control path," in *Proc. ISLPD-95: ACM/IEEE Int. Symp. Low Power Design*, Dana Point, CA, Apr. 1995, pp. 93–98.

[18] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. W. Brodersen, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design*, vol. 14, no. 1, pp. 12–31, 1995.

[19] J. M. Chang and M. Pedram, "Module assignment for low power," in *Proc. EuroDAC-96: IEEE Eur. Design Automation Conf.*, Geneva, Switzerland, Sept. 1996, pp. 376–381.

[20] N. Kumar, S. Katkoori, L. Rader, and R. Vemuri, "Profile-driven behavioral synthesis for low power VLSI systems," *IEEE Design Test Comput. Mag.*, vol. 12, no. 3, pp. 70–84, 1995.

[21] R. San Martin and J. Knight, "Optimizing power in ASIC behavioral synthesis," *IEEE Design Test Comput. Mag.*, vol. 13, no. 2, pp. 58–70, 1996.

[22] L. Benini, A. Bogliolo, M. Favalli, and G. De Micheli, "Regression models for behavioral power estimation," in *Proc. PATMOS-96: Int. Workshop on Power and Timing Modeling, Optimization and Simulation*, Bologna, Italy, Sept. 1996, pp. 179–186.

[23] L. Benini, A. Bogliolo, and G. De Micheli, "Characterization-free behavioral power modeling," in *Proc. DATE-98: IEEE Design Automation and Test in Europe*, Paris, France, Feb. 1998, pp. 767–773.

[24] ——, "Adaptive least mean square behavioral power modeling," in *Proc. EDTC-97: IEEE Eur. Design and Test Conf.*, Paris, France, Mar. 1997, pp. 404–410.

[25] C. M. Huizer, "Power dissipation analysis of CMOS VLSI circuits by means of switch-level simulation," in *Proc. IEEE Eur. Solid State Circuits Conf.*, 1990, pp. 61–64.

[26] C. X. Huang, B. Zhang, A.-C. Deng, and B. Swirski, "The design and implementation of powermill," in *Proc. ISLPD-95: ACM/IEEE Int. Symp. Low Power Design*, Dana Point, CA, Apr. 1995, pp. 105–110.

[27] F. Najm, R. Burch, P. Yang, and I. Hajj, "Probabilistic simulation for reliability analysis of CMOS VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 4, pp. 439–450, 1990.

[28] C.-Y. Tsui, M. Pedram, and A. M. Despain, "Efficient estimation of dynamic power dissipation under a real delay model," in *Proc. ICCAD-93: IEEE/ACM Int. Conf. Computer Aided Design*, Santa Clara, CA, Nov. 1993, pp. 224–228.

[29] F. Najm, "Transition density: A new measure of activity in digital circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, no. 4, pp. 310–323, 1993.
[30] R. Marculescu, D. Marculescu, and M. Pedram, "Efficient power estimation for highly correlated input streams," in *Proc. DAC-32: ACM/IEEE Design Automation Conf.*, San Francisco, CA, June 1995, pp. 628–634.
[31] C.-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. M. Despain, and B. Lin, "Power estimation in sequential logic circuits," *IEEE Trans. VLSI Syst.*, vol. 3, no. 3, pp. 404–416, 1995.
[32] R. Burch, F. Najm, P. Yang, and T. Trick, "A Monte Carlo approach for power estimation," *IEEE Trans. VLSI Syst.*, vol. 1, no. 1, pp. 63–71, 1993.
[33] C.-S. Ding, C.-T. Hsieh, Q. Wu, and M. Pedram, "Stratified random sampling for power estimation," in *Proc. ICCAD-96: IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1996, pp. 577–582.
[34] L.-P. Yuan, C.-C. Teng, and S.-M. Kang, "Statistical estimation of average power dissipation in CMOS VLSI circuits using nonparametric technique," in *Proc. ISLPED-96: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1996, pp. 73–78.
[35] T.-L. Chou and K. Roy, "Statistical estimation of sequential circuit activity," in *Proc. ICCAD-95: IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1995, pp. 34–37.
[36] D. Marculescu, R. Marculescu, and M. Pedram, "Stochastic sequential machine synthesis targeting constrained sequence generation," in *Proc. DAC-33: ACM/IEEE Design Automation Conf.*, Las Vegas, NV, June 1996, pp. 696–701.
[37] R. Marculescu, D. Marculescu, and M. Pedram, "Adaptive models for input data compaction for power simulators," in *Proc. ASPDAC-2: ACM/IEEE Asia South Pacific Design Automation Conf.*, Chiba, Japan, Jan. 1997, pp. 391–396.
[38] D. Marculescu, R. Marculescu, and M. Pedram, "Sequence compaction for probabilistic analysis of finite state machines," in *Proc. DAC-34: ACM/IEEE Design Automation Conf.*, Anaheim, CA, June 1997, pp. 12–15.
[39] S. Powell and P. Chau, "Estimating power dissipation of VLSI signal processing chips: The FA Techniques," in *Proc. IEEE Workshop on VLSI Signal Processing*, 1990, vol. IV, pp. 250–259.
[40] P. Landman and J. Rabaey, "Power estimation for high-level synthesis," in *Proc. EDAC-93: IEEE Eur. Conf. Design Automation*, Paris, France, Feb. 1993, pp. 361–366.
[41] S. Gupta and F. N. Najm, "Power macromodeling for high-level power estimation," in *Proc. DAC-34: ACM/IEEE Design Automation Conf.*, Anaheim, CA, June 1997, pp. 365–370.
[42] D. Liu and C. Svensson, "Power consumption estimation in CMOS VLSI chips," *IEEE J. Solid State Circuits*, vol. 29, no. 6, pp. 663–670, 1994.
[43] H. Mehta, R. Owens, and M. J. Irwin, "Energy characterization based on clustering," in *Proc. DAC-33: ACM/IEEE Design Automation Conf.*, Las Vegas, NV, June 1996, pp. 702-707.
[44] Q. Wu, C.-S. Ding, C.-T. Hsieh, and M. Pedram, "Statistical design of macro-models for RT-level power evaluation," in *Proc. ASPDAC-2: ACM/IEEE Asia South Pacific Design Automation Conf.*, Chiba, Japan, Jan. 1997, pp. 523–528.
[45] Q. Qiu, Q. Wu, M. Pedram, and C.-S. Ding, "Cycle-accurate macro-models for RT-level power analysis," in *Proc. ISLPED-97: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1997, pp. 125–130.
[46] C.-T. Hsieh, C.-S. Ding, Q. Wu, and M. Pedram, "Statistical sampling and regression estimation in power macro-modeling," in *Proc. ICCAD-96: IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1996, pp. 583–588.
[47] D. Kirovski and M. Potkonjak, "System-level synthesis of low-power hard real-time systems," in *Proc. DAC-34: ACM/IEEE Design Automation Conf.*, Anaheim, CA, June 1997, pp. 697–702.
[48] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software cosynthesis of embedded systems," in *Proc. DAC-34: ACM/IEEE Design Automation Conf.*, Anaheim, CA, June 1997, pp. 703–708.
[49] R. P. Dick and N. K. Jha, "MOGAC: A multiobjective genetic algorithm for the cosynthesis of hardware-software embedded systems," in *Proc. ICCAD-97: IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1997, pp. 522–529.
[50] M. T.-C. Lee, V. Tiwari, S. Malik, and M. Fujita, "Power analysis and minimization techniques for embedded DSP software," *IEEE Trans. VLSI Syst.*, vol. 5, no. 1, pp. 123–135, 1997.
[51] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee, "Instruction level power analysis and optimization of software," *J. VLSI Signal Process.*, pp. 1–18, 1996.
[52] S. Wuytack, F. Catthoor, L. Nachtergaele, and H. De Man, "Global communication and memory optimizing transformations for low power design," in *Proc. IWLPD-94: ACM/IEEE Int. Workshop on Low Power Design*, Napa Valley, CA, Apr. 1994, pp. 203–208.
[53] P. R. Panda and N. D. Dutt, "Reducing address bus transitions for low power memory mapping," in *Proc. EDTC-96: IEEE Eur. Design and Test Conf.*, Paris, France, Mar. 1996, pp. 63–67.
[54] ———, "Low power mapping of behavioral array to multiple memories," in *Proc. ISLPED-96: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1996, pp. 289–292.
[55] M. T.-C. Lee and V. Tiwari, "A memory allocation technique for low-energy embedded DSP software," in *Proc. ISLPE-95: IEEE Int. Symp. Low Power Electronics*, San Diego, CA, Oct. 1995, pp. 44–45.
[56] S. Wuytack, F. Catthoor, L. Nachtergaele, and H. De Man, "Power exploration for data dominated video applications," in *Proc. ISLPED-96: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1996, pp. 359–364.
[57] J. P. Diguet, S. Wuytack, F. Catthoor, and H. De Man, "Formalized methodology for data reuse exploration in hierarchical memory mappings," in *Proc. ISLPED-97: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1997, pp. 30–35.
[58] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. VLSI Syst.*, vol. 4, no. 1, pp. 42–55, Mar. 1996.
[59] C.-H. Hwang and A. C.-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. ICCAD-97: IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1997, pp. 28-32.
[60] E. Musoll and J. Cortadella, "Scheduling and resource binding for low power," in *Proc. ISSS-95: IEEE Int. Symp. System Synthesis*, Cannes, France, Apr. 1995, pp. 104–109.
[61] ———, "High-level synthesis techniques for reducing the activity of functional units," in *Proc. ISLPD-95: ACM/IEEE Int. Symp. Low Power Design*, Dana Point, CA, Apr. 1995, pp. 99–104.
[62] D. Kim and K. Choi, "Power-conscious high-level synthesis using loop folding," in *Proc. DAC-34: ACM/IEEE Design Automation Conf.*, Anaheim, CA, June 1997, pp. 441–445.
[63] J. Monteiro, S. Devadas, P. Ashar, and A. Mauskar, "Scheduling techniques to enable power management," in *Proc. DAC-33: ACM/IEEE Design Automation Conf.*, Las Vegas, NV, June 1996, pp. 349–352.
[64] J. M. Chang and M. Pedram, "Low power register allocation and binding," in *Proc. DAC-32: ACM/IEEE Design Automation Conf.*, San Francisco, CA, June 1995, pp. 29–35.
[65] A. Raghunathan and N. K. Jha, "Behavioral synthesis for low power," in *Proc. ICCD-94: IEEE Int. Conf. Computer Design*, Cambridge, MA, Oct. 1994, pp. 318–322.
[66] S. Bhatia and N. K. Jha, "Genesis: A behavioral synthesis system for hierarchical testability," in *Proc. EDTC-94: IEEE Eur. Design and Test Conf.*, Paris, France, Feb. 1994, pp. 272–276.
[67] L. Goodby, A. Orailoglu, and P. M. Chau, "Microarchitectural synthesis of performance-constrained, low-power VLSI designs," in *Proc. ICCD-94: IEEE Int. Conf. Computer Design*, Cambridge, MA, Oct. 1994, pp. 323–326.
[68] R. Mehra and J. Rabaey, "Exploiting regularity for low-power design," in *Proc. ICCAD-96: IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1996, pp. 166–172.
[69] C. Gebotys, "Low energy memory and register allocation using network flow," in *Proc. DAC-34: ACM/IEEE Design Automation Conf.*, Anaheim, CA, June 1997, pp. 435–440.
[70] D. D. Gajski and L. Ramachandran, "Introduction to high-level synthesis," *IEEE Design Test Comput. Mag.*, vol. 11, no. 4, pp. 44–54, Dec. 1994.
[71] G. De Micheli, *Synthesis and Optimization of Digital Circuits.* New York: McGraw-Hill, 1994.
[72] A. Raghunathan and N. K. Jha, "An iterative improvement algorithm for low power data-path synthesis," in *Proc. ICCAD-95: IEEE Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1995, pp. 597–602.
[73] J. M. Chang and M. Pedram, "Energy minimization using multiple supply voltages," *IEEE Trans. VLSI Syst.*, vol. 5, no. 4, pp. 436–443, 1997.
[74] S. Raje and M. Sarrafzadeh, "Variable voltage scheduling," in *Proc. ISLPD-95: ACM/IEEE Int. Symp. Low Power Design*, Dana Point, CA, Apr. 1995, pp. 9–14.
[75] M. Johnson and K. Roy, "Optimal selection of supply voltages and level conversions during data-path scheduling under resource constraints," in *Proc. ICCD-96: IEEE Int. Conf. Computer Design*, Austin, TX, Oct. 1996, pp. 72–77.

[76] M. Igarashi *et al.*, "A low-power design method using multiple supply voltages," in *Proc. ISLPED-97: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1997, pp. 36–41.
[77] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," *IEEE Trans. VLSI Syst.*, vol. 3, no. 1, pp. 49–58, 1995.
[78] C.-L. Su, C.-Y. Tsui, and A. M. Despain, "Saving power in the control path of embedded processors," *IEEE Design Test Comput. Mag.*, vol. 11, no. 4, pp. 24–30, 1994.
[79] H. Mehta, R. M. Owens, and M. J. Irwin, "Some issues in gray code addressing," in *Proc. GLS-VLSI-96: IEEE/ACM Great Lakes Symp. VLSI*, Ames, IA, Mar. 1996, pp. 178–180.
[80] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, "Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems," in *Proc. GLS-VLSI-97: IEEE/ACM Great Lakes Symp. VLSI*, Urbana, IL, Mar. 1997, pp. 77–82.
[81] ——, "Address bus encoding techniques for system-level power optimization," in *Proc. DATE-98: IEEE Design Automation and Test in Europe*, Paris, France, Feb. 1998, pp. 861–866.
[82] E. Musoll, T. Lang, and J. Cortadella, "Exploiting the locality of memory references to reduce the address bus energy," in *Proc. ISLPED-97: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1997, pp. 202–207.
[83] L. Benini, G. De Micheli, E. Macii, M. Poncino, and S. Quer, "System-level power optimization of special purpose applications: The beach solution," in *Proc. ISLPED-97: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1997, pp. 24–29.
[84] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, 1986.
[85] J. Hartmanis and R. E. Stearns, *Algebraic Structure Theory of Sequential Machines*. Englewood Cliffs, NJ: Prentice-Hall, 1966.
[86] S. Devadas and A. R. Newton, "Decomposition and factorization of sequential finite state machines," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 11, pp. 1206–1217, 1989.
[87] L. Benini, P. Vuillod, C. Coelho, and G. De Micheli, "Synthesis of low-power selectively-clocked systems from high-level specification," in *Proc. ISSS-96: IEEE Int. Symp. System Synthesis*, La Jolla, CA, Oct. 1996, pp. 57–62.
[88] B. Lin and A. R. Newton, "Implicit manipulation of equivalence classes using binary decision diagrams," in *Proc. ICCD-91: IEEE Int. Conf. Computer Design*, Cambridge, MA, Oct. 1991, pp. 81–85.
[89] B. Kumthekar, I. H. Moon, and F. Somenzi, "A symbolic algorithm for low-power sequential synthesis," in *Proc. ISLPED-97: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1997, pp. 56–61.
[90] K. Roy and S. C. Prasad, "Circuit activity based synthesis for low power reliable operations," *IEEE Trans. VLSI Syst.*, vol. 1, no. 4, pp. 503–513, 1993.
[91] E. Olson and S. M. Kang, "Low-power state assignment for finite state machines," in *Proc. IWLPD-94: Int. Workshop on Low Power Design*, Napa Valley, CA, Apr. 1994, pp. 63–68.
[92] C.-Y. Tsui, M. Pedram, and A. M. Despain, "Low power state assignment targeting two- and multilevel logic implementations," in *Proc. ICCAD-94: IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1994, pp. 82–87.
[93] L. Benini and G. De Micheli, "State assignment for low power dissipation," *IEEE J. Solid-State Circuits*, vol. 30, no. 3, pp. 258–268, 1995.
[94] P. Surti, L. F. Chao, and A. Tyagi, "Low power FSM design using Huffman-style encoding," in *Proc. EDTC-97: IEEE Eur. Design and Test Conf.*, Paris, France, Mar. 1997, pp. 521–525.
[95] G. D. Hachtel, M. Hermida, A. Pardo, M. Poncino, and F. Somenzi, "Reencoding sequential circuits to reduce power dissipation," in *Proc. ICCAD-94: IEEE/ACM Int. Conf. Computer Aided Design*, San Jose, CA, Nov. 1994, pp. 70–73.
[96] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Markovian analysis of large finite state machines," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 12, pp. 1479–1493, 1996.
[97] L. Lavagno, P. C. McGeer, A. Saldanha, and A. L. Sangiovanni-Vincentelli, "Timed Shannon circuits: A power-efficient design style and synthesis tool," in *Proc. DAC-32: ACM/IEEE Design Automation Conf.*, San Francisco, CA, June 1995, pp. 254–260.
[98] S.-I. Minato, "Zero-suppressed BDD's for set manipulation in combinatorial problems," in *Proc. DAC-30: ACM/IEEE Design Automation Conf.*, Dallas, TX, June 1993, pp. 272–277.
[99] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou, "Precomputation-based sequential logic optimization for low power," *IEEE Trans. VLSI Syst.*, vol. 2, no. 4, pp. 426–436, 1994.
[100] J. Monteiro, J. Rinderknecht, S. Devadas, and A. Ghosh, "Optimization of combinational and sequential circuits for low power using precomputation," in *Proc. 1995 Chapel Hill Conf. Advanced Research in VLSI*, Chapel Hill, NC, Mar. 1995, pp. 430–444.
[101] L. Benini, P. Siegel, and G. De Micheli, "Automatic synthesis of gated clocks for power reduction in sequential circuits," *IEEE Design Test Comput. Mag.*, vol. 11, no. 4, pp. 32–40, 1994.
[102] L. Benini and G. De Micheli, "Transformation and synthesis of FSM's for low power gated clock implementation," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 6, pp. 630–643, 1996.
[103] L. Benini, G. De Micheli, E. Macii, M. Poncino, and R. Scarsi, "Symbolic synthesis of clock-gating logic for power optimization of control-oriented synchronous networks," in *Proc. EDTC-97: IEEE Eur. Design and Test Conf.*, Paris, France, Mar. 1997, pp. 514–520.
[104] L. Benini, M. Favalli, and G. De Micheli, "Design for testability of gated-clock FSM's," in *Proc. EDTC-96: IEEE Eur. Design and Test Conf.*, Paris, France, Mar. 1996, pp. 589–596.
[105] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: Pushing power management to logic synthesis/design," in *Proc. ISLPD-95: ACM/IEEE Int. Symp. Low Power Design*, Dana Point, CA, Apr. 1995, pp. 221–226.
[106] C. Lemonds and S. S. Shetti, "A low power 16 by 16 multiplier using transition reduction circuitry," in *Proc. IWLPD-94: ACM/IEEE Int. Workshop on Low Power Design*, Napa Valley, CA, Apr. 1994, pp. 139–142.
[107] A. Raghunathan, S. Dey, N. K. Jha, and K. Wakabayashi, "Controller respecification to minimize switching activity in controller/data path circuits," in *Proc. ISLPED-96: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1996, pp. 301–304.
[108] ——, "Power management techniques for control-flow intensive designs," in *Proc. DAC-34: ACM/IEEE Design Automation Conf.*, Anaheim, CA, June 1997, pp. 429–434.
[109] A. Raghunathan, S. Dey, and N. K. Jha, "Glitch analysis and reduction in register transfer level power optimization," in *Proc. DAC-33: ACM/IEEE Design Automation Conf.*, Las Vegas, NV, June 1996, pp. 331–336.
[110] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," *J. VLSI Comput. Syst.*, vol. 1, no. 1, pp. 41–67, 1983.
[111] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proc. ICCAD-93: IEEE/ACM Int. Conf. Computer Aided Design*, Santa Clara, CA, Nov. 1993, pp. 398–402.

Enrico Macii (M'92), for a photograph and biography, see p. 232 of the March 1998 issue of this TRANSACTIONS.

Massoud Pedram (S'88–M'90), for a photograph and biography, see p. 83 of the February 1998 issue of this TRANSACTIONS.

Fabio Somenzi graduated from the Politecnico di Torino in 1980.

He was with SGS-Thomson until 1989, when he joined the electrical and computer engineering Faculty of the University of Colorado at Boulder. In 1987, he was a Visiting Industrial Fellow at the University of California, Berkeley. Since the fall of 1997, he has been on a sabbatical at Cadence Berkeley Laboratories, Berkeley. His research interests include synthesis and verification of digital systems.