Regression-Based RTL Power Modeling

ALESSANDRO BOGLIOLO
DI - University of Ferrara
LUCA BENINI
DEIS - University of Bologna
and
GIOVANNI DE MICHELI
CSL - Stanford University

Register-transfer level (RTL) power estimation is a key feature for synthesis-based design flows. The main challenge in establishing a sound RTL power estimation methodology is the construction of accurate, yet efficient, models of the power dissipation of functional macros. Such models should be automatically built, and should produce reliable average power estimates.

In this paper we propose a general methodology for building and tuning RTL power models. We address both hard macros (presynthesized functional blocks) and soft macros (functional units for which only a synthesizable HDL description is provided). We exploit linear regression and its nonparametric extensions to express the dependency of power dissipation on input and output activity. Bottom-up off-line characterization of regression-based power macromodels is discussed in detail. Moreover, we introduce a low overhead on-line characterization method for enhancing the accuracy of off-line characterization.

Categories and Subject Descriptors: B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids; B.6.3 [**Logic Design**]: Design Aids

General Terms: Design, Simulation, Verification

Additional Key Words and Phrases: Adaptive characterization, functional macros, RTL design, RTL power modeling, regression models

1. INTRODUCTION

Managing the rapidly increasing complexity of VLSI and ULSI design is probably the most severe challenge for today's digital designers. As technol-

This work was partially supported by NSF (under contract MIP-9421129).
Authors' addresses: A. Bogliolo, DI - University of Ferrara, Ferrara, 44100, Italy; L. Benini, DEIS - University of Bologna, Bologna, 40136, Italy; G. D. Micheli, CSL - Stanford University, Stanford, CA 94305-9030.

ogy improves to the point where it is now possible to pack millions of gates on a single chip, it becomes almost impossible to design every single functional component in a system at a low level of abstraction. The only practical way to cope with complexity is to design at high level of abstraction. Hardware description languages (HDLs) are a key component of the paradigm shift because they allow us to specify the behavior of a digital system in a compact and abstract fashion. However, state-of-the-art designs have HDL descriptions of sizes exceeding 100,000 lines, and it is rapidly becoming unthinkable to develop every design from scratch in the time imposed by market requirements.

Design *reuse* is key for rapid and correct development. This observation is well understood and fruitfully exploited by the software engineering community, where entire libraries of optimized and well-debugged routines are available to the software engineer. Hardware design would greatly benefit from the availability of a similar feature: productivity would improve and debugging time would be much reduced. Recognizing this need, and the consequent business opportunities, both EDA and hardware companies are focusing on the concept of *intellectual property* (IP) components, i.e., reusable functional blocks that can be instantiated within larger designs. The success of the business model based on IP components depends critically upon the availability of a well-defined mechanism for specifying how they can be embedded in a system and what levels of performance are to be expected.

The recent *virtual socket interface* (VSI) initiative [VSI Alliance 1997] is aimed at providing standards for the interface layer between IP providers and IP users. Similarly to software library routines, the functional interface of IP components must be formally specified, together with a detailed description of the component's behavior. Unlike software, however, detailed information about the *cost* of instantiating an IP component in hardware must be provided as well. Cost is related to three fundamental metrics: speed, power, and area. When instantiating an IP component, a designer must be able to estimate its delay, power consumption, and area.

We assume a design flow where a system is specified as an interconnection of digital components whose behavior is described at the register-transfer level (RTL). Specification and functional verification of RTL designs is much faster than design and verification at the gate level. On the other hand, a functionally correct RTL implementation gives the designer a higher degree of confidence because it matches the behavior of the final implementation at the cycle boundaries. Unfortunately, functional verification is not sufficient to validate a design. Accurate estimates of cost metrics are equally important. Ideally, a designer would like to obtain data on speed, power dissipation, and area *while running functional verification*, with minimal computational overhead. In this work we address the problem of enabling fast and accurate power simulation at the RT level.

RTL designs are usually described hierarchically. The main challenge in estimating the power dissipation of a hierarchical design is the construction of accurate black-box power models for the leaves of the hierarchy, for

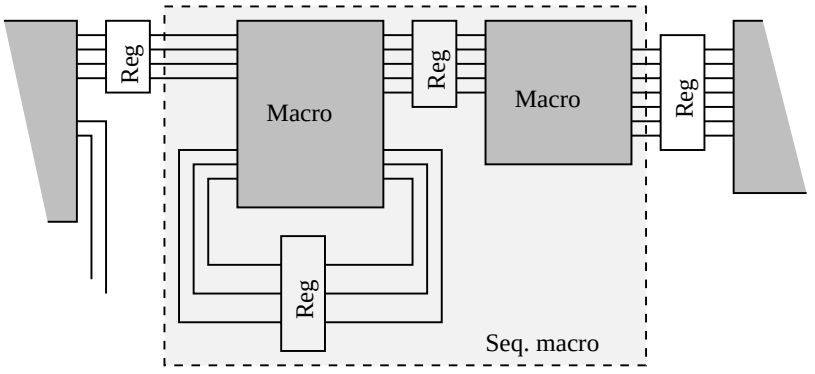


Fig. 1. Schematic representation of a design described at the register-transfer level. The design is described hierarchically. Leaf components are combinational macros and registers.

which only functional descriptions are available at the RT level. We restrict our scope to structural RTL representations whose leaf components are combinational logic blocks (hereafter called *macros*) and state-holding elements (*registers*), as shown in Figure 1. We do not target generic sequential macros with an unobservable internal state such as the one represented by the shaded area in Figure 1, which are inherently more abstract. Such sequential macros can sometimes be further decomposed in combinational logic blocks and registers. In other cases, however, complex sequential macros are directly described as black-box RTL primitives. Our modeling approach is not applicable in these cases because it is based on the assumption that there is a correlation between input-output activity and internal power dissipation. For generic sequential primitives this assumption does not hold.

Combinational macros and registers can be either IP components, which are externally provided, or application-specific blocks of HDL code, which are directly provided by the designer.[1] Similar to the VSI specification [VSI Alliance 1997], we distinguish two classes of macros: hard and soft.[2] *Hard macros* are components for which the full gate-level implementation is available, though it is not instantiated at the RT level for efficiency (for example, consider a multiplier: its functional specification consists of a few lines of HDL, while its gate-level description contains hundreds of gates). In contrast, *soft macros* are synthesizable HDL specifications of hardware blocks. The gate-level implementation of a soft macro is *not* available, and is usually synthesized on the fly whenever the RTL description is compiled to a gate-level netlist.

[1]The term *intellectual property* is often used to denote very large (possibly sequential) components, such as microprocessor cores. We deal with IP components at a finer level of granularity, namely registers and combinational functional units.

[2]The VSI specification introduces a third class: *firm macros*, conceptually a hybrid between hard and soft macros. For our purposes, firm macros can be seen as hard.

We propose a power macromodeling approach for RTL designs with the following salient features:

- Power models for the atomic RTL components (i.e., macros and registers) are automatically constructed.
- The models can be statically precharacterized once and for all, or they can be tuned automatically during the design process.

From the designer's viewpoint, the application of our technique makes it possible to obtain accurate average power estimates during functional RTL simulation at the price of a modest increase in simulation time. The only additional care required is that the simulation patterns provided by the designer match typical (or expected) usage traces for the target system. This is an obvious consequence of the fact that power dissipation is strongly input-dependent, and performing simulation of unlikely or impossible conditions will produce nonrepresentative estimates of the average power consumption.

Our approach is based on linear and nonparametric regression models that exploit the high correlation between power consumption and input-output switching activities. We first propose an *off-line* characterization procedure that is performed only once to compute power models for all combinational macros and registers in a given RTL library. off-line characterization can be augmented (or replaced) by an *on-line*, adaptive characterization procedure that either improves the accuracy of precharacterized macromodels (i.e., performs tuning) or builds the models from scratch on the fly. On-line characterization requires the availability of a multilevel simulation engine that supports gate-level as well as RTL simulation. An interesting feature of regression-based macromodels is that they enable direct (static) evaluation of the average power consumption, given signal and transition probabilities at the macros' boundaries.

Incidentally, we target only average power estimates. Evaluating instantaneous and peak power consumption is a different task, and is not addressed here. Nevertheless, the regression models we propose are all pattern-dependent in nature, in that they provide pattern-by-pattern power information during RTL simulation. As shown later, pattern dependence is a key feature that grants accuracy and flexibility to the average power estimates. Experimental evidence shows that accuracy is satisfactory (the error on average power estimates is less than 10% compared to accurate real-delay gate-level simulations) and the RTL simulation slowdown is very small.

The paper is organized as follows. In Section 2, the task of RTL power modeling is formally defined and related work on the topic is reviewed. The basic theory of models based on linear and non-linear regressions is presented in Section 3. In the same section, we describe an algorithm for obtaining average power estimates by providing only input and output average switching activities. Automatic model tuning during simulation (i.e., on-line) is addressed in Section 4. An adaptive characterization

strategy based on the *least mean square* (LMS) algorithm that greatly improves the model accuracy with small computational overhead is proposed. We also describe an implementation based on Verilog XL of the adaptive strategy. Experimental results are presented and evaluated in Section 5, while Section 6 concludes the work.

2. RTL POWER MODELING

As outlined in the previous section, our purpose is to provide an accurate and efficient power modeling methodology for combinational (hard and soft) macros and registers. Consider a macro with n inputs $\mathbf{x} = [x_1, \ldots, x_n]^{\mathcal{T}}$ and m outputs $\mathbf{y} = [y_1, \ldots y_m]^{\mathcal{T}}$. [3] Assume that the circuit is stable at time t^i and t^f ($t^f > t^i$), and that an input transition from $\mathbf{x}^i$ to $\mathbf{x}^f$ occurs in the time interval $T = [t^i, t^f]$. We call *input transition vector* the concatenation of two successive input patterns $[\mathbf{x}^{i\mathcal{T}}, \mathbf{x}^{f\mathcal{T}}]^{\mathcal{T}}$ (hereafter denoted $(\mathbf{x}^i, \mathbf{x}^f)$ for simplicity) and by $e(\mathbf{x}^i, \mathbf{x}^f)$ we denote the supply energy drawn by the unit in the time interval T.[4] The power modeling task consists of finding a simple but accurate *black-box model* of $e(\mathbf{x}^i, \mathbf{x}^f)$ using boundary information only (i.e., the knowledge of the inputs and outputs of the unit at time t^i and t^f). We call a function $e(\mathbf{x}^i, \mathbf{x}^f)$ that associates any input transition with the corresponding energy consumption provided by gate-level (or switch-level) power simulation of the implementation of the unit a *golden model*. Unfortunately, for real-size circuits, $e(\mathbf{x}^i, \mathbf{x}^f)$ is nothing but an ideal reference, impossible to construct and to be kept in memory.

2.1 Related Work

Two main classes of models have been proposed in the past: *pattern-independent* and *pattern-dependent* models. The distinctive characteristic of the first class is that the average power dissipation is estimated in one single model evaluation, and the input information required to perform evaluation ranges from none to a compact representation of input-output signal probabilities and switching activities. Pattern-independent models include the constant power model [Powell and Chau 1990], the *dual bit type* model [Landman and Rabaey 1995], and the *3d table* model [Gupta and Najm 1997]. The constant model is highly inaccurate because it does not account for input statistics. All other approaches take into account input statistics by assuming the dependency of power dissipation on average input transition activity and/or probability. However, they still belong to the pattern-independent class because they cannot be used for pattern-by-pattern power estimation during simulation. The main limitation of the dual-bit type method is that it requires direct human intervention for

[3]Vectors are hereafter denoted by boldface letters, apex $\mathcal{T}$ denotes the matrix transpose.
[4]Power consumption corresponding to transition vector $(\mathbf{x}^i, \mathbf{x}^f)$ is defined as $e(\mathbf{x}^i, \mathbf{x}^f)/T$. In the following we assume $T = 1$, thus e and p have the same value and can be used interchangeably.

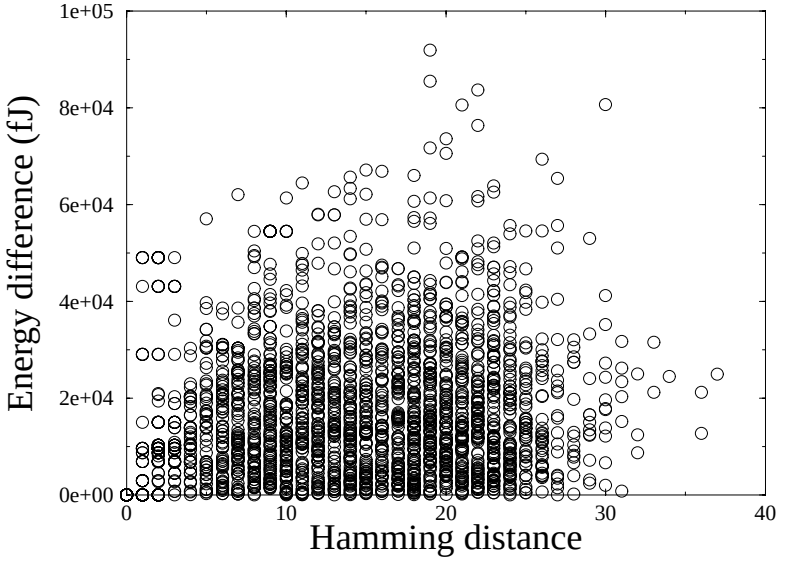


Fig. 2. Scatter plot of the difference between the energy dissipation associated with different input transitions, as a function of the Hamming distance, between the corresponding transition vectors. Data refers to an 8-bit carry-lookahead adder..

model construction. The *3d table* method is completely automated and quite accurate.

Pattern-dependent models attempt to directly approximate the golden model. In other words, they provide a function that associates power dissipation with transition vectors. In order for a model to be practical, the computation of the function value, given a transition vector, must be fast (orders of magnitude faster than gate-level simulation). Pattern-dependent models are represented by the *clustering* approach proposed by Mehta et al. [1996] and by the *regression-based power macromodeling* approach by Hsieh et al. [1996].

Clustering relies on the assumption that closely related input transition vectors have similar power dissipations. In our experience, for many circuits, the assumption is not valid. Consider, for instance, the effect of the carry-in signal on the power consumption of a full adder. Two transition vectors that differ only for the value taken by the carry-in bit at time t^f give rise to completely different power consumptions, even if their Hamming distance is 1. This is shown in the scatter plot of Figure 2 for an 8-bit carry-lookahead adder. Points in the graph represent the difference between the energy consumption associated with different input transitions as a function of the Hamming distance between the corresponding transition vectors. The correlation between the two quantities is weak. In particular, the leftmost points in the graph show that the difference between the energy consumption associated with similar input transitions may be of the same order as the average energy drawn by the circuit. Although the authors obtained an average error within 10%–15% on the

same sample used for characterization, they do not discuss the accuracy's dependence on the input statistics. Finally, the model proposed in Mehta et al. [1996] is strongly pattern-dependent: for each input pattern, a table lookup has to be performed to obtain the power estimate.

The regression-based approach proposed in Hsieh [1996] is related to ours, and has been independently (and concurrently) developed. The authors postulate the existence of power macromodels for the primitive components in an RTL design and propose two techniques for accurately estimating the average power consumption during RTL simulation with small overhead, compared to baseline functional simulation. The first technique, called *sampler macromodeling*, reduces the runtime overhead by reducing the number of times the macromodels are evaluated. The second technique, called *adaptive macromodeling*, exploits a multilevel simulation engine for enhancing the accuracy of the macromodels at the expense of a relatively small runtime overhead. In this work, we first consider the issue of creating the macromodels whose existence is postulated by Hsieh [1996], then we propose a model-tuning technique to improve the runtime accuracy of the models similar to adaptive macromodeling. The differences between the two techniques will be discussed later in more detail.

Both pattern-dependent and pattern-independent models discussed above rely on *characterization*. Model characterization is a process that improves the accuracy of a power model by exploiting accurate simulations of the gate-level (or switch-level) implementation of the unit to be modeled, repeated for a significant set of input transitions (i.e., a *sample*). Characterization-based approaches are not applicable when a low-level implementation (the *golden model*) is not available. Characterization-free *information-theoretical models* have been developed as well [Marculescu 1996; Nemani and Najm 1996; Lioy 1997]. Such models are based on the input-output functionality of the macro only, and cannot be used to distinguish among alternative implementations of the same functionality. Although information-theoretical models may be used in the early phases of design exploration, they are quite inaccurate.

2.2 Average Power Estimation

The final goal of our technique is to provide an accurate estimate for a complex RTL design. The designer starts with an RTL *specification* (in an HDL such as Verilog or VHDL), a *library* of RTL macros, a *technology library* of elementary gates (such as NAND, NOR, etc.) and a set of *patterns*. The RTL specification is a hierarchical structure whose atomic components (i.e., the leaves of the hierarchy) are combinational macros and registers that belong to the RTL library. Such components can be provided either by IP vendors, from previous in-house designs, or be specifically designed for the application at hand.

The library of RTL components is precharacterized and initial power macromodels are obtained for each component (the structure of such macromodels is described in the following sections). For hard macros, the

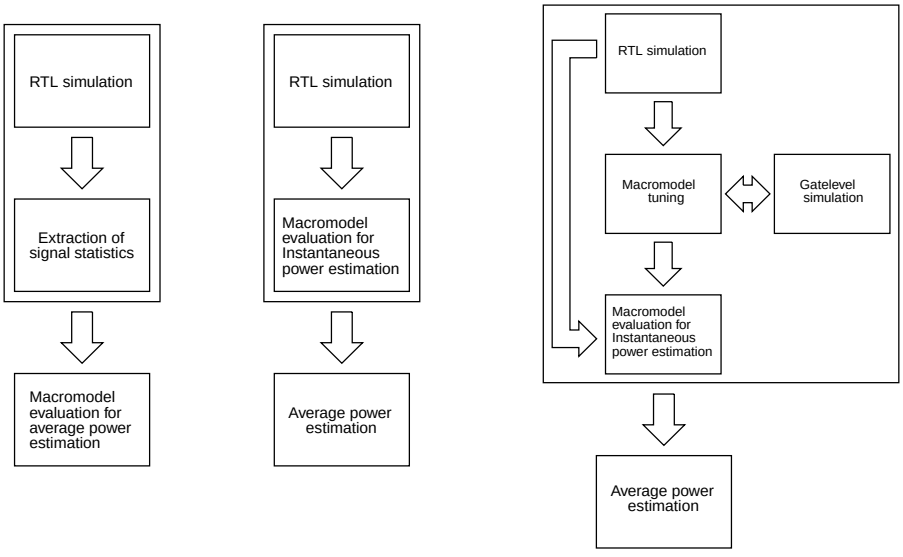


Fig. 3. (a) Static power estimation; (b) dynamic power estimation; (c) dynamic power estimation with *in-situ* tuning.

gate-level (or transistor-level) implementation serves as golden model for macromodel characterization. For soft macros, an implementation in the target technology library is first synthesized then used as golden model for characterization.

Our power estimation tool can provide three types of average power estimates. We describe them in order of increasing accuracy.

- *Static power estimation* (Figure 3(a)). For each atomic component, signal probabilities and switching activity values for its inputs and outputs are collected. This can be done by running RTL functional simulation and counting the number of times signals have a constant value or a transition (as suggested in Landman and Rabaey [1995]). Power is computed in a postprocessing step. For each atomic component, its macromodel is evaluated only once, providing signal probabilities and transition activities as inputs. It returns an estimate of the average power of the atomic component. The average power estimates of all components are summed to obtain the total average power.

- *Dynamic power estimation* (Figure 3(b)). Power in a complex design is obtained by simply running an RTL simulation: the macromodel is re-evaluated for each transition at the input-outputs of an atomic component. It returns an instantaneous power estimate. The instantaneous power estimates of each component are summed together during simulation to obtain the total power. Finally, the average power is obtained by computing the ratio between total power and the duration of the simulation.

- *Dynamic power estimation with in-situ tuning* (Figure 3(c)). For this simulation mode, a multilevel (RTL and gate-level or transistor-level) simulation engine is required. RTL simulation is run. The macromodels of some (or all) of the atomic components in the design are *tuned* during RTL simulation by running gate-level (or transistor-level) accurate power simulation for the input patterns provided by the RTL simulator. Tuning improves accuracy of the macromodels but increases simulation time, hence it should complete as soon as possible. Once the macromodels are tuned, gate-level simulation is stopped and only RTL simulation continues. The final estimate of the average power is obtained as for dynamic power estimation.

Clearly, static power estimation is the fastest. Its overhead over purely functional RTL simulation reduces to computing signal probabilities and transition activities. Accuracy losses are mainly caused by (i) inaccuracies of the macromodels; (ii) loss of information on actual input pattern distribution (signal probabilities and transition activities are a "lossy" description of the input pattern distribution). Dynamic power estimation eliminates the second source of inaccuracy, but requires the evaluation of power macromodels for each component and each pattern. Hence, it decreases the speed of RTL simulation. Finally, in-situ tuning reduces the first source of inaccuracy, but further decreases simulation speed because gate-level simulation is required to tune the macromodels. Notice that the runtime overhead of the three techniques could be reduced by applying the sampler macromodeling technique presented in Hsieh [1996].

In the next sections we focus on the details of power macromodels, characterization, and tuning. Regarding the choice of the macromodels, our primary requirements are (i) generality and suitability for automatic construction; (ii) efficient automatic characterization; and (iii) good accuracy. We focus on *linear regression models*, which are attractive for several reasons: (i) they are simple to store and evaluate; (ii) they are completely general; (iii) they do not require any human knowledge to be constructed and characterized; (iv) they have well-known statistical properties; and (v) they are suitable for both dynamic and static evaluation. Moreover, they can be tuned easily by means of efficient adaptive algorithms [Widrow and Stearns 1985]. Linear regression models (and their non-linear extensions) are presented in the next section, while on-line tuning is addressed in Section 4.

3. LINEAR REGRESSION MODELS

In general, the dependence of e on $\mathbf{x}^i$ and $\mathbf{x}^f$ is not linear. Using a linear function to approximate the behavior of $e(\mathbf{x}^i, \mathbf{x}^f)$ may lead to more severe approximations than those provided by clustering [Mehta et al. 1996]. In both cases, however, the main source of error is use of meaningless independent variables. Although any transition vector $(\mathbf{x}^i, \mathbf{x}^f)$ is associated with a unique value of e, the dependence of e on a single input variable (say

x_k) is almost unpredictable and heavily sensitive to the value taken by other variables.

More significant independent variables to approximate the pattern-dependence of e are suggested by the physical understanding of the main power consuming phenomena for the reference logic family. Referring to static CMOS logics, we observe that (i) in a combinational circuit some input has to switch in order to dissipate power; and (ii) the presence of switching outputs always corresponds to some internal activity. Based on these observations in Benini et al. [1996], we propose to represent e as a linear function of input/output switching activities. Symbolically:

$$e = c_0 + c_1 a_1 + \ldots + c_n a_n + c_{n+1} a_{n+1} + \ldots + c_{n+m} a_{n+m} \tag{1}$$

where $\mathbf{c} = [c_0, c_1, \ldots, c_{n+m}]^{\mathcal{T}}$ is the vector of fitting coefficients to be determined during characterization and $\mathbf{a} = [a_0, a_1, \ldots, a_{n+m}]^{\mathcal{T}}$ is the vector of independent variables.[5] Each variable is a Boolean flag taking a value of 1 if and only if there is a transition on the corresponding signal: $a_1 = x_1^i \oplus x_1^f, \ldots, a_n = x_n^i \oplus x_n^f, a_{n+1} = y_1^i \oplus y_1^f, \ldots, a_{n+m} = y_m^i \oplus y_m^f$.

Notice that several transition vectors $(\mathbf{x}^i, \mathbf{x}^f)$ are associated with the same configuration of $\mathbf{a}$ (i.e., , with the same approximated value of e). Thus, in principle, the linear model can be viewed as a clustering technique that associates a unique energy value with a cluster of input transitions. However, using activity flags as independent variables gives rise to a more significant partition than that based on the Hamming distance between transition vectors [Mehta et al. 1996]. This is a key advantage that ultimately affects accuracy. Moreover, using output transition flags as additional independent variables actually reduces the cluster size, further improving accuracy. The correlation between I/O switching activity (expressed as the total number of input and output signals switching during the same transition) and power consumption is shown in Figure 4(a) for an 8-bit carry-lookahead adder. Notice that the proposed regression model provides a deeper insight than the model used in Figure 4(a), in that it accounts for the activity of single inputs and outputs.

To determine the coefficients of Eq. (1) we need a *sample* of input-output activity vectors $[\mathbf{a}^{(0)}, \mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(s)}]^{\mathcal{T}}$ and corresponding reference values of the energy $[e^{(0)}, e^{(1)}, \ldots, e^{(s)}]^{\mathcal{T}}$. The sample of data collected during the characterization phase can be represented by a pair $(\mathbf{A}, \mathbf{e})$. If s is the sample size, $\mathbf{A}$ is an $s \times (n + m + 1)$ Boolean matrix containing the values taken by the independent variables during characterization (its kth row is $\mathbf{a}^{(k)\mathcal{T}} = [1, a_1^{(k)}, a_2^{(k)}, \ldots, a_{n+m}^{(k)}]$), while $\mathbf{e}$ is a vector of size s contain-

[5]The choice of independent variables depends on the target technology and logic family. For instance, the power consumption of precharged CMOS circuits is mainly correlated to the initial values of the input signals $\mathbf{x}^i$. Though we implicitly refer to static CMOS realizations, the modeling approach we propose can be extended to different technologies and logic families once a meaningful set of independent variables ($\mathbf{a}$) is chosen.

ing the corresponding values of the dependent variable (the kth element is $e^{(k)}$) obtained from accurate gate-level power simulation.

Given a sample (**A, e**), coefficients **c** are the unknowns of the following system of linear equations:

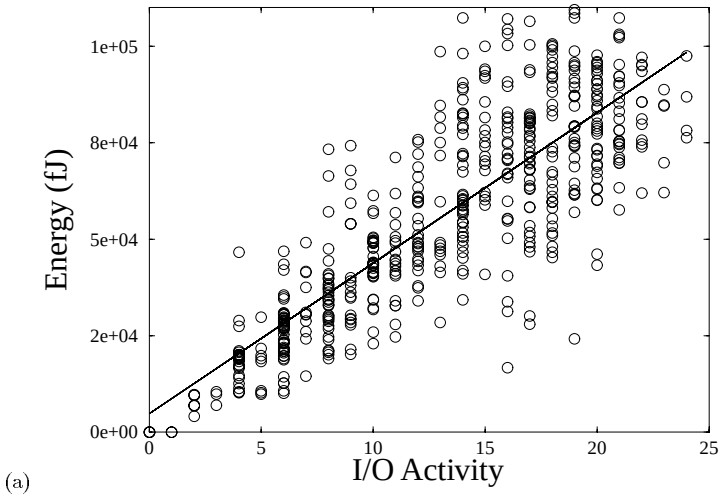
$$\mathbf{e} = \mathbf{Ac}. \tag{2}$$

Due to the statistic nature of the characterization process, the sample size must be significantly larger than the number of parameters to be characterized. Hence, matrix **A** has many more rows than columns and the linear system is over-defined. The vector **c** giving the minimum mean square error among all possible linear estimators of **e** can be obtained from Eq. (2) using well-known least squares fitting techniques [Bowerman and O'Connell 1990]. An important property of the least squares linear model is that it always produces an estimate of e with the same average value as the average value of e in the sample used for fitting. Hence, it is guaranteed to perform at least as well as an average-value constant approximation.

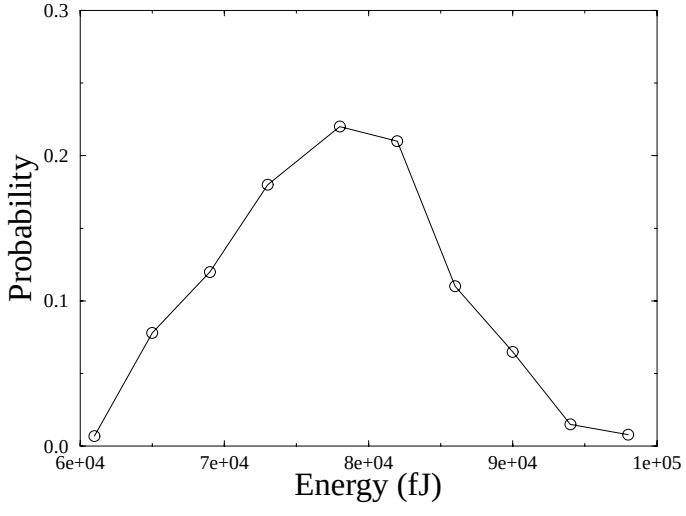
Moreover, the least squares solution is robust in presence of the noise made by the dependence of e on parameters that do not take part in the model (such as the initial input values). If the dependent variable can be seen as the superposition of a deterministic variable (function of the independent variables) and a random noise with Gaussian distribution, it can be shown that the least squares fit maximizes the probability that, for a given value of the independent variables, the dependent noisy variable has the value predicted by the least squares solution [Bowerman and O'Connell 1990]. We checked the Gaussian hypothesis by plotting the distribution of energy consumption obtained for all input transitions corresponding to the same configurations of **a**. An example probability distribution for the same adder mentioned above is shown in Figure 4(b): the bell-shaped curve closely resembles a Gaussian distribution. Extensive tests performed on different benchmarks for different values of **a** provided similar results.

The experimental evidence we collected suggests that linear models are accurate for a large class of macros. In practice, as long as linear models are characterized by means of least squares fitting on a significant sample of reference data, their power estimates are as accurate as a linear approximation of a non-linear function can be. Custom-designed macromodels can be more accurate than linear ones, but they are not always available. Linear macromodels represent a generic default solution with well-known and desirable statistical properties.

However, if the golden model is strongly non-linear, a linear approximation may produce unacceptably large errors. To improve the accuracy of linear models, in Benini et al. [1996] we proposed an advanced regression technique closely related to *nonparametric* statistical models, known as *regression trees* [Breiman et al. 1993]. We called our procedure *tree regres-*



(a)



(b)

Fig. 4. Statistical analysis of the power consumption of an 8-bit carry-lookahead adder. (a) Correlation between the cycle-by-cycle energy and the I/O activity, expressed as the total number of switching input/output signals; (b) bell-shaped distribution of the energy consumption due to different input transitions with the same activity vector (namely, $\mathbf{a}$ = (0011001100011011,00101001)).

sion because it recursively builds a tree structure with linear regression equations on the leaves.

3.1 Tree Regression

The inputs of large logic units can often be grouped into two classes: *control inputs* and *data inputs*. Control inputs have a very strong influence on the behavior of the units because they select either among different modes of

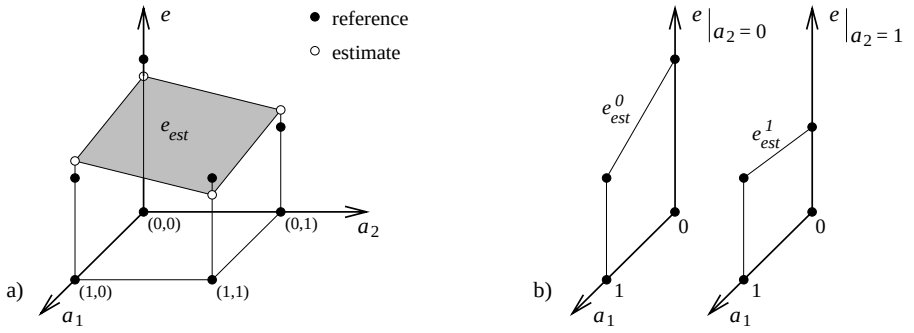


Fig. 5. (a) Least squares linear approximation of a non-linear function of two Boolean variables; (b) exact fit of the same function using two linear equations of variable a_1. The value of a_2 is used to switch between the two models.

operation (as in ALUs) or among different input-output paths (as in multiplexers). On the other hand, while high activity on data inputs usually correlates well with high power dissipation, such behavior is not observed for control inputs. Following this observation, control inputs can be used to *select among different regression equations*. Given a control variable and a sample, we split the sample in two subsets, one for each value of the control variable. We then compute two new linear regression models on the two subsamples.

The advantage of this procedure is intuitively clear. If the behavior of the logic unit changes radically for different values of the control variable, a single regression model will attempt to find a linear fit between two widely spaced clusters of data. As a result, the fitting will not be satisfactory for either of the two clusters. If we split the data and we separately fit the two clusters, much better results are obtained. The effectiveness of model splitting is illustrated in Figure 5 for a two variable function.

In principle, model splitting also addresses nonlinearities. A function e of Boolean variables a_1, a_2,..., a_n is non-linear if and only if it cannot be decomposed in a sum of terms, where each term depends on a single variable. In other words, e is nonlinear if and only if there exists an independent variable (a_i) that affects its dependence on some other variables. In this case, a_i plays the role of a control variable. Accuracy can then be improved by using two different regression models for the two values of a_i.

Though we observed that I/O signal values are not significant independent variables for regression, they can be natural decision variables for model splitting. The different operational modes of a functional unit are usually associated with different configurations of a few control bits. Hence, both signal transition flags ($\mathbf{a}$) and static signal values ($\mathbf{x}^i$ and $\mathbf{x}^f$) are candidate variables for splitting. Intuitively, input signal values will be used to select among the different power models, possibly associated with the different operational modes of the same unit, while I/O transition flags (which are the same variables used for regression) will possibly be used as decision variables to address nonlinearities.

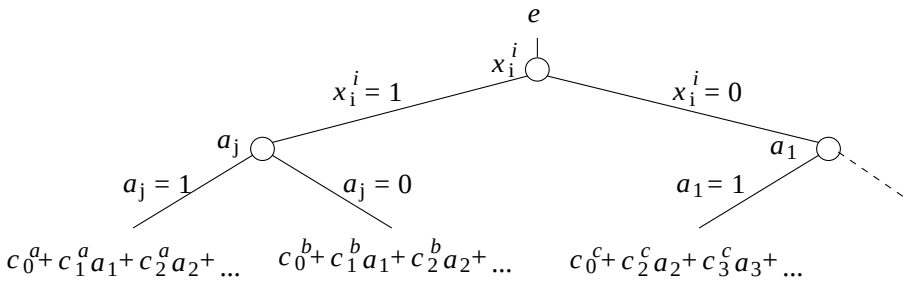


Fig. 6. A generic *regression tree* of depth 2.

This reasoning can be extended to multiple control variables in a recursive fashion. Once we have split the data in two clusters, we can split further if other control variables can be found in the partial models. The structure generated by the recursive splitting is called a *regression tree* (see Figure 6). The internal nodes of the tree are labeled with the control variables on which we split the model. The leaves correspond to regression equations with $n + m - d$ independent variables, where d is the number of transition flags already used as control variables along the path leading to each leaf. Consider the regression tree of Figure 6. The initial value of the ith I/O signal is the first control variable. For $x_i^i = 1$, the model is further split on the value of the jth transition flag, while for $x_i^i = 0$, the second-level splitting variable is a_1. At each leaf, a regression equation is used to represent the dependence of e on those transition flags that have not yet been fixed along the descending path. For instance, variable a_1 does not appear on the rightmost equation, since its value is always 1 when the equation is used.

The number of leaves is exponential in the depth of the tree. Consequently, the splitting procedure must be limited to a small number of control variables, both to keep the model small and to guarantee the statistical significance of linear regressions. In the limiting situation where all independent variables are used for splitting, the leaves of the tree become constants. This never happens in practice, since both the memory requirements for storing the complete tree and the sample size to grant statistical significance would be excessive.

3.1.1 *Splitting Criterion*. Since our goal is a black-box modeling procedure, we need an automatic splitting criterion based on boundary information. For this purpose we use a statistical approach that can be outlined as follows: (i) the global regression model is computed; (ii) for each candidate variable v_i (hereafter, $\mathbf{v}$ denotes the vector of candidate control variables, composed of both I/O signal values and activities), the proportion of variance σ_i^2 of the dependent variable e due to v_i is computed [Bowerman and O'Connell 1990]; (iii) the variable with the largest σ_i^2 is chosen for splitting. The rationale behind this procedure is quite simple. The variance σ_i^2 is high if a change in the value of v_i is associated to a wide variation of

e (on average). In other words, if the independent variable v_i selects between two radically different behaviors of the unit, the variance of e due to v_i will be significant.

The advantage of using a statistical method to select the splitting variables is twofold. No human knowledge is required to steer the characterization process, and the method may also be applied to units with no evident control signals in order to isolate behaviors with good linearity characteristics. The automatic splitting process makes our regression *nonparametric*. In nonparametric regression, different functional relationships are approximated with regression models for which not only the fitting parameters but also the structure of the model itself may change. The regression tree retains the soundness of linear regression, joined with the flexibility of nonparametric methods.

The last issue is the choice of the terminating conditions. If the number of samples is sufficient and the distribution of the samples is uniform, the user simply specifies the depth value and the procedure automatically builds a complete regression tree. This is not always the case. Some of the independent variables are *outputs* of the module, and the user has no control on their distribution. Moreover, the input vectors may not be distributed uniformly. As a consequence, some of the branches may find singular or statistically nonsignificant least-square matrices. In this case, the least-square equation of the level immediately above in the tree is used, with the independent variable used for the last splitting stuck at the value corresponding to the branch of the tree.

Notice that, in some cases, splitting may not produce any benefit even if the residual sample size guarantees statistical significance. This happens, in particular, if the behavior of e is almost linear on the entire sample and there are no evident control variables. Since we do not want to split when splitting is not necessary, we take a user-specified threshold value of σ_i^2 as additional stopping criterion.

The pseudocode of the tree regression procedure is shown in Figure 7. The procedure returns the pointer to the top node of the regression tree. The parameters are the extended sample matrix $\mathbf{V}$ containing the values of all candidate variables (matrix $\mathbf{A}$ of independent regression variables is a submatrix of $\mathbf{V}$); the sample vector $\mathbf{e}$, containing the measured values of power dissipation; the regression vector computed in the upper level of the recursion (initialized to NULL when the procedure is first called); the current level of recursion (initialized to 0); the required depth of the tree and the threshold value of σ_i^2 (defined by the user). Observe the two base cases for the recursion: least-square matrix too small (or singular) and leaf reached. If no stopping criteria are satisfied, the procedure `find max variance` selects the candidate variable v with the maximum σ^2. Procedure `split` selects the rows of $\mathbf{V}$ with fixed value $v = 0$ or $v = 1$ and the corresponding elements in $\mathbf{e}$.

```
tree RegTree(V, e, c_up, level, depth, threshold) {
  A = extract_transitions(V);          /* Extract the matrix of indep.  vars.  */
  if ( insufficient_sample(A,e) ) {              /* Base case:  singular matrix */
    Node→leaf = c_up;
    return(Node);
  } else {
    c = compute_least_square(A,e);
  }
  if ( (level == depth) || (max_variance(V,e) < threshold) ) {
    Node→leaf = c;                                  /* Base case:  leaf reached */
  } else {              /* Recursion:  split on the var.  with max.  e-variance */
    v = find_max_variance(V, e);
    Node→splitvar = v;
    (V_then,e_then) = split(V, e, v = 1);
    Res = RegTree(V_then, e_then, c, level+1, depth, threshold);
    Node→then = Res;
    (V_else,e_else) = split(V, e, v = 0);
    Res = RegTree(V_else, e_else, c, level+1, depth, threshold);
    Node→else = Res;
  }
  return(Node);
}
```

Fig. 7. Pseudocode of the tree regression algorithm.

We have described the advantages of our method for the case of units with control inputs. In the discussion of the results, we will see that the regression tree is useful in general. The choice of the splitting variables is exclusively based on statistical criteria, thus even units with no evident control signals may benefit from our technique, which dynamically develops a model by trying to isolate behaviors with good linearity characteristics.

3.2 Static Power Analysis

Both linear models and regression trees can be easily incorporated in any RTL simulator to provide pattern-by-pattern power estimates. At each clock cycle, the power consumption of the functional units is obtained from the switching activity at their I/O signals (directly available during simulation). The complexity of power evaluation is linear in the size of the model (i.e., in the number of I/O signals of each unit).

Even if power evaluation does not impair the efficiency of RTL simulation, numerous simulations are required to obtain significant estimates of the average power. Moreover, power estimation has to be repeated each time the designer decides to explore a different design choice by replacing one or more macros with functionally equivalent ones. As a consequence, pattern-dependent power simulation is not practical for exploring the design space. Faster (and usually less accurate) static analysis techniques are often preferred: the entire design is simulated once and for all to collect

statistical information about switching activity at signals connecting functional units; signal statistics are then used to estimate power consumption.

In this section we show how the regression models we propose can be used to perform static power analysis at the RT level, without loosing accuracy with respect to dynamic simulations, based on the same models. Consider the linear model of Eq. (1) applied to a n-input, m-output macro. To estimate the power consumption of an instance of the macro used in the context of a larger design, we simulate the design for a large number of input patterns (say, N), we evaluate Eq. (1) at each clock cycle and compute the average of the N values we obtain. In symbols:

$$\begin{aligned} e_{avg} &= \frac{1}{N}\sum_{k=1}^{N} e^{(k)} \\ &= \frac{1}{N}\sum_{k=1}^{N} (c_0 + c_1 a_1^{(k)} + \ldots + c_{n+m} a_{n+m}^{(k)}) \end{aligned} \tag{3}$$

where apex (k) is used to denote the kth clock cycle. For the sake of simplicity, we use $\mathcal{E}[\cdot]$ to denote the average operator applied to the sample:

$$e_{avg} = \mathcal{E}[c_0 + c_1 a_1 + \ldots + c_{n+m} a_{n+m}] \tag{4}$$

By linearity, Eq. (4) can be rewritten as

$$\begin{aligned} e_{avg} &= c_0 + c_1 \mathcal{E}[a_1] + \ldots + c_{n+m} \mathcal{E}[a_{n+m}] \\ &= c_0 + c_1 \tau_1 + \ldots + c_{n+m} \tau_{n+m} \end{aligned} \tag{5}$$

where the τ's are the transition probabilities at the inputs and outputs of the unit.

Equation (5) actually provides an abstract model for *static power analysis*. Transition probabilities at the interconnections between the functional units can be computed once and for all and then used to evaluate the power consumption of each element. If different macros are available to implement each functional unit, different solutions can be compared without resimulating the circuit.

Notice that Eqs. (3) and (5) represent exactly the same model. There is no loss of accuracy in using the static approach instead of the dynamic one. If the same set of N test patterns is used both to perform pattern-dependent power simulation and to compute transition probabilities, the two equations return exactly the same value. In both cases, accuracy depends on the modeling assumptions discussed in Section 3.1 and on the number of test vectors applied (N).

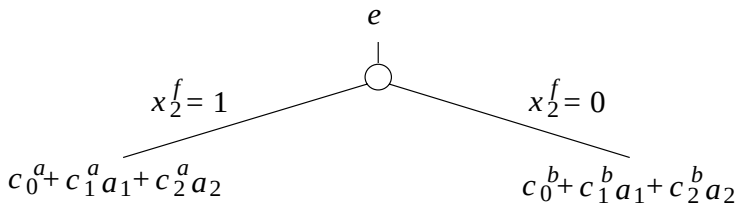


Fig. 8. A one-level *regression tree* used to express the dependence of e from two independent variables a_1 and a_2. The initial value of signal x_2 is also used as an additional control variable.

Similar considerations can be applied to regression trees, but in this case some accuracy may be lost when using the static power estimation model. The loss of accuracy is due to the inherent non-linearity of the regression tree. This statement can be clarified through an example.

Example 1. Consider the simple regression tree in Figure 8. The average power estimate provided by the model is

$$e_{avg} = \frac{1}{N}\sum_{k=1}^{N}(x_2^{f(k)}(c_0^a + c_1^a a_1^{(k)} + c_2^a a_2^{(k)}) + (1 - x_2^{f(k)})(c_0^b + c_1^b a_1^{(k)} + c_2^b a_2^{(k)})) \quad (6)$$

By applying the same transformations used in Eq. (5), we re-express e_{avg} in terms of signal probabilities (s) and transition probabilities (τ):

$$e_{avg} = s_2^{(k)}(c_0^a + c_1^a \tau_1^{(k)} + c_3^a \tau_2^{(k)}) + (1 - s_2^{(k)})(c_0^b + c_1^b \tau_1^{(k)} + c_3^b \tau_2^{(k)})) \quad (7)$$

In this case, however, a further assumption is required to state the equivalence of the two expressions. In fact, Eq. (6) is not linear, therefore the static power estimate does not imply loss of accuracy with respect to the dynamic power estimate if and only if the splitting variable x_2^f is statistically independent from the regression variables. In general, the following relation holds for any pair of statistically independent variables v_i and v_j:

$$\mathcal{E}[v_i v_j] = \mathcal{E}[v_i]\mathcal{E}[v_j] = p_i p_j$$

where p is used to represent the probability of a generic Boolean variable (either a signal value or a transition flag) being 1. In our example, the independence of splitting variable x_2^f guarantees the term by term equivalence of Eqs. (6) and (7). If the independence does not hold, accuracy may be impaired by the static evaluation of Eq. (7).

The pseudocode of an algorithm for the static evaluation of a regression tree is shown in Figure 9. The initial inputs are a pointer to the root of the tree and the array $\mathcal{P}$ of input/output signal and transition probabilities. Power consumption is recursively computed at each node during a depth-first traversal of the tree. There are two main cases: at a leaf node the value taken by the corresponding linear equation is returned (Eq. (5)),

```
float EvalRegTree(Node, 𝒫) {
  if ( Node→splitvar == NULL ) {                    /* Base case:  leaf node */
    e = EvalLinEq(Node→leaf, 𝒫);
  } else {                                        /* Recursion:  internal node */
    e_then = EvalRegTree(Node→then, 𝒫);
    e_else = EvalRegTree(Node→else, 𝒫);
    v_i = Node→splitvar;
    e = p_i e_then + (1 − p_i) e_else;
  }
  return (e);
}
```

Fig. 9. Pseudocode of the algorithm for the static evaluation of a regression tree.

while at a nonleaf node the return value is the weighted sum of those of the two branches (the weight being the probability p_i of the splitting variable v_i).

4. ON-LINE TUNING

Power models based on linear (and nonparametric) regressions can be statically precharacterized for all combinational macros and registers in the library. However, we have seen that the characterization procedure optimally fits the macromodel to a characterization sample that may not be representative of the actual patterns fed to the macro instances in the design. If this is the case, the accuracy of the macromodel prediction decreases. In this section we propose a two-step solution to this problem, which consists of using an adaptive characterization algorithm (namely, the *least mean square* algorithm, LMS [Widrow and Stearns 1985]) and a concurrent simulation paradigm. The key idea is tuning the macromodel by performing adaptive characterization during RTL simulation.

When tuning the macromodel, accurate gate-level simulation of the macro is run concurrently with RTL simulation. Concurrent simulation proceeds until the tuning process converges. Once the macromodel has reached convergence, the gate-level description of the corresponding unit does not need to be simulated any longer. We then keep simulating only its RTL description with the just characterized back-annotated power model. Tuning can be concurrently performed for several units. Simulation progressively speeds up as more models reach convergence.

The main effect of model tuning is a sizable improvement in accuracy, since characterization is run *in situ* (i.e., within the functional simulation of the complete design). In this way, significant input patterns are automatically generated for the units, thus automatically taking into account the actual spatio-temporal correlations and nonuniformities in pattern distribution. This is a fundamental point: in general, the LMS algorithm converges to different models for different instances of the same macro.

Each model is the best linear fit to the power dissipation of the instance, given the input patterns that are actually provided by the environment.

4.1 The LMS Algorithm

LMS is a gradient-based technique that iteratively modifies the coefficients of Eq. (1) and adaptively tries to minimize the mean square error produced by the model, until it reaches convergence in a neighborhood of the theoretical minimum error solution (i.e., , the least squares solution). An accurate description of LMS is beyond the scope of this work (see Widrow and Stearns [1985] for an exhaustive treatment). We simply outline the algorithm and the reasons for its usefulness in our case.

The iterative formula used for updating the coefficient vector is the following:

$$\mathbf{c}^{(k+1)} = \mathbf{c}^{(k)} + 2\mu\epsilon^{(k)}\mathbf{a}^{(k)} \tag{8}$$

where k is the iteration step, $\mathbf{c}^{(k)}$ is the current assignment of the fitting coefficients, μ is a fixed constant (to be discussed later), $\mathbf{a}^{(k)}$ is the input-output activity vector (i.e., the kth row of $\mathbf{A}$) and $\epsilon^{(k)} = e^{(k)} - e^{(k)}_{est}$ is the difference between the energy actually dissipated corresponding to the kth transition and the energy estimated by the model for the same transition. Intuitively, at each iteration, the algorithm tries to modify the coefficients in order to reduce the error made by the model.

The initial value ($\mathbf{c}^{(0)}$) of the coefficient vector does not change the asymptotic properties of convergence. However, convergence will take fewer iterations if $\mathbf{c}^{(0)}$ is close to the optimum value. This property can be exploited to speed-up the recharacterization process when a macromodel has been precharacterized and just needs to be slightly tuned.

The convergence of the LMS algorithm is controlled by parameter μ. It can be shown that for convergence the following condition must hold:

$$0 < \mu < \frac{1}{tr[\mathbf{R}]} \tag{9}$$

where $tr[\mathbf{R}]$ is the trace of the *correlation matrix* $\mathbf{R}$ [Widrow and Stearns 1985]. Element r_{ij} of $\mathbf{R}$ represents the correlation between the ith and the jth independent variables (i.e., , $r_{ij} = \mathcal{E}[a_i a_j]$). Since in our case the independent variables may take value 0 or 1 only, the elements of $\mathbf{R}$ cannot be greater than 1. An upper bound for the trace of $\mathbf{R}$ is then

$$tr[\mathbf{R}] \leq n + m + 1. \tag{10}$$

Even if convergence can be ensured easily, a more subtle tradeoff is involved in the choice of μ. If the convergence of the iteration is too fast, the accuracy of the final solution can be compromised. A measure of the accuracy of the solution is defined in Widrow and Stearns [1985]: the

misadjustment M. LMS tries to find the optimum $\mathbf{c}$ by minimizing the mean square error (MSE) of the linear model. However, due to the nonlinearity of the dependent variable, the minimum MSE ($minMSE$) is always greater than 0. Hence, error $\epsilon^{(k)}$ is usually nonzero and Eq. (8) keeps changing the assignment of $\mathbf{c}$ in a neighborhood of the optimum. The misadjustment is an adimensional measure of the distance between the current solution and the best one: $M = (MSE - minMSE) / minMSE$. It can be shown that the average value of M at the end of the adaptive process is estimated by the following formula [Widrow and Stearns 1985]:

$$M = \mu \cdot tr[\mathbf{R}] \tag{11}$$

Since $\mathbf{R}$ cannot be controlled, the only way to reduce the misadjustment is to reduce μ. On the other hand, μ is directly proportional to the speed of convergence: it can be shown that the convergence to the minimum MSE is exponential with a time constant T_{MSE} (expressed in the number of iterations) given by Widrow and Stearns [1985]:

$$T_{MSE} = \frac{1}{4\mu\lambda_n} \tag{12}$$

where λ_n is the smallest eigenvalue of matrix $\mathbf{R}$. It is apparent that the choice of μ is paramount for obtaining a satisfying regression model with LMS. In our implementation, μ is chosen as one tenth of the (worst-case) maximum value allowed; the user can, however, override this choice by either specifying μ or specifying the maximum number of patterns for which the characterization must be run. In the second case, a warning is given if the value of μ appears to be larger than the upper bound of Eq. (9).

The advantages of LMS over the least squares solution are numerous. First, the computational requirements are mild: the iteration formula has complexity linear in the number of inputs and outputs. Second, no additional storage of past data is required. Third, the characterization process can be performed on the fly, while the simulation is running, and several units' instances can be characterized at the same time. Finally, the LMS solution retains the desirable properties of robustness and statistical significance of the least squares solution.

4.2 Impact on Accuracy

A key advantage of the tuning process is that adaptive characterization is performed in the same conditions in which the model will be used. This improves the model accuracy, since it reduces the effect of approximations due to the modeling assumptions of (i) linear dependence on the switching activities and (ii) negligible dependence on other parameters.

The effect of the linear approximation is depicted in Figure 10(a). Suppose e is a nonlinear function of Boolean variables a_1 and a_2. It is not

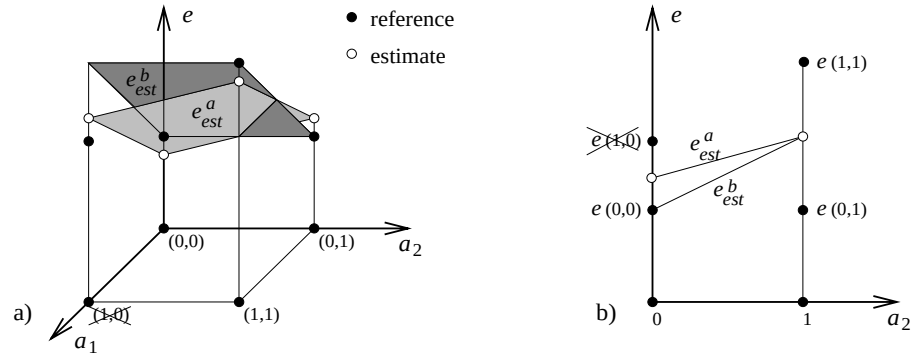


Fig. 10. Effects of modeling assumptions on the accuracy of the estimate. (a) Linear approximation of a non-linear function; (b) single-variable estimator of a two-variable function. The inherent inaccuracy due to the modeling assumptions can be minimized by using significant samples for characterization.

possible to find a linear function of a_1 and a_2 taking the same values of e for all assignments of the independent variables. The linear estimator found by least squares fitting (e^a_{est}) actually distributes the inherent inaccuracy among the four points of the input space as well as possible. The accuracy of e^a_{est} for a given input assignment (e.g., (0,1)) cannot be improved without impairing its accuracy in some other position. However, if there is an input configuration (e.g., (1,0)) that never occurs, then we do not care about the corresponding value of e. Hence, we can try to find a better estimator of e in the remaining points by sacrificing the accuracy of e_{est} in the *don't care* point. In the trivial case of Figure 10(a), the simplified fitting problem has an exact solution represented by e^b_{est}.

When modeling power consumption, *don't care* conditions can be found due to the embedding of the unit in the design.

Example 2. Consider Add1 in Figure 11: an 8-bit adder is used to perform 6-bit additions. The two most significant bits of the input words are always 0. As a consequence, the set of possible assignments of the input/output activities is reduced by a factor of 2^4. The subsequent *don't care* conditions can be used to improve the accuracy of the linear estimator. For a carry-lookahead implementation of Add1, the root mean square error of the power estimate reduces from 23% to 19% if *don't care* conditions are exploited.

A second kind of modeling error is introduced by neglecting some of the parameters that affect the dependent variable. Assume, for instance, that we want to approximate function $e(a_1, a_2)$ of Figure 10(a) using a linear function of a_2 only. For each value of a_2, e may assume two different values depending on the value of a_1 that does not take part in the model (see Figure 10(b)). If a_1 is neglected, the value of e at a given a_2 is a random

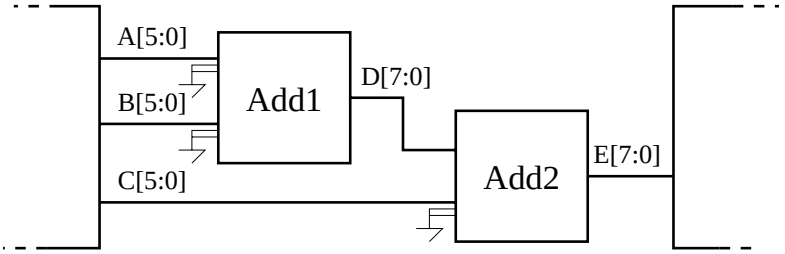


Fig. 11. Example design.

variable $e(\cdot\ , a_2)$ taking values $e(0, a_2)$ and $e(1, a_2)$. The value $e(a_2)$ that we try to fit is the mean of this random variable.

Even if a linear estimator can be found that exactly matches $e(a_2)$ (e^a_{est} for function e of Figure 10(b)), there is a residual modeling error due to the effect of a_1. The mean square error made by neglecting a_1 is the variance of $e(\cdot, a_2)$ for a fixed value of a_2. In general, since the variation of e for a given configuration of the independent variables depends on the variation of (some of) the parameters that do not take part in the model, there is a relation between the variance of e and the distribution of the neglected variables. The smaller the set of possible values of the neglected parameters, the smaller the variance of e.

Referring to the example in Figure 10(b), if we know that the input configuration $(a_1, a_2) = (1, 0)$ never occurs, the mean value of $e(\cdot, 0)$ reduces to $e(0, 0)$ and its variance to 0. As a consequence, there is no error in using $e^b_{est}(a_2)$ for $a_2 = 0$ as a model of e.

In our case, there are two sets of neglected parameters that may have a sizable impact on power consumption: the initial states of the input signals $\mathbf{x}^i$ and their arrival times $\mathbf{t}$. For a given activity vector $\mathbf{a}$, the energy consumed by a unit is to be considered as a random variable $e(\mathbf{a}, \cdot, \cdot)$ with distribution dependent on the number of configurations that the neglected parameters ($\mathbf{x}^i$ and $\mathbf{t}$) may assume, corresponding to the same activity vector.

Example 3. Consider an 8-bit carry-lookahead adder. Figure 4(b) represents the distribution of e for a given value of $\mathbf{a}$ and for a null vector of arrival times (corresponding to aligned input transitions). If the adder is used to implement Add1 in Figure 11, four input signals are stuck at 0, thus reducing the set of possible input patterns, and ultimately affecting the distribution of e. In particular, the standard deviation of e (i.e., the inherent modeling error) changes from 9.6% to 7.8%.

The input arrival times are even more critical: they are analog quantities and their relative values (i.e., *input skews*) play a sizable role in power consumption.

Example 4. Referring to Figure 11, assume that signals A, B, and C are perfectly aligned in time. Due to the propagation delay of Add1, there is a skew T between the transitions of C and D at the inputs of Add2. In particular, since T is larger than the propagation delay through a single gate, the misalignment between C and D causes disjoint power consuming phenomena within Add2. This ultimately results in a larger overall power dissipation.

Since the vector $\mathbf{t}$ of input arrival times may take an infinite number of configurations (and there are no general pruning criteria), off-line power characterizations are usually based on the simplifying hypothesis of aligned input transitions. Unfortunately, this assumption is often violated in practice, causing crude underestimates of power consumption.

On the other hand, if the power model is adaptively characterized on its real context, the effect of input misalignments is implicitly modeled, since the actual arrival times are used during characterization. Notice that, to do this, timing information is to be propagated during RTL and gate-level simulations. In our implementation, we use average propagation delays for RTL modules and an accurate pattern-dependent delay model for signal propagation through gate-level units [Bogliolo et al. 1996]. This is discussed in the next section.

Example 5. Consider Add2 in Figure 11 and assume we have a linear power estimator characterized off-line under the 0-skew assumption. If we use the model to estimate the power consumption of Add2 during cycle-accurate RTL simulation, we obtain a 54% underestimation of the average power consumption, due to the neglected effect of input misalignments. On the other hand, if an event-driven simulation paradigm is adopted, the power model of Add2 is evaluated twice in each clock cycle (corresponding to the subsequent transitions of input vectors C and D), leading to a 23% overestimation of the average power consumption. Consider, in contrast, a linear model adaptively characterized *in situ*. No simplifying timing assumptions are made during characterization: the actual timing information is available at the unit's boundaries and a real delay model is used for signal propagation at the gate level. The LMS automatically adjusts the model's parameters in order to fit the power consumption of the unit under a real delay model. Even if the model is evaluated in the context of a cycle-accurate RTL simulation, it provides power estimates with a 2% average error from gate-level simulation (the pattern-by-pattern mean-square error is 17%). Event-driven simulation is no longer required to improve the accuracy of the RTL power estimate.

In summary, adaptive characterization allows us to find the best linear estimator for the power consumption of a unit in the context in which it operates. This means that model coefficients are automatically set in order to realize the best fit of the dependent variable corresponding to those configurations of the independent variables that really occur in practice.

Example 6. For the design in Figure 11, we obtained completely different estimators for Add1 and Add2, leading to an overall root mean square error of 18% on pattern dependent power estimates. To check the consistency of this result we tried to use the model obtained forAdd2 (Add1) for both units. The overall mean-square error became 35% (25%).

Similar conclusions are drawn in Hsieh [1996] regarding adaptive power macromodeling. Interestingly, the methodology proposed in Hsieh [1996] differs from ours in two ways: first, a statistical criterion is used to decide how many patterns are needed for tuning the macromodel; second, the model is not tuned by adjusting its coefficients, but a regression equation is obtained to correlate its prediction with actual power measurements. For further details, see Hsieh [1996].

4.3 Implementation Issues

The adaptive power model bridges the gap between fast RTL simulations and accurate gate-level power estimates, but it requires a simulation engine supporting both levels of abstraction. Moreover, a straightforward push-button interface to the synthesis environments is necessary to allow iterative design improvements. To meet these requirements, we implemented the procedures for RTL power characterization and estimation as additional features of PPP [Bogliolo et al. 1996], a unified synthesis and simulation tool based on Verilog-HDL.

The basic simulation engine of PPP is Verilog-XL, which parses the hierarchical description of the network and performs event-driven logic simulation. Routines for gate and RT level power characterization/evaluation have been implemented in C and integrated into the logic simulator using the Programming Language Interface (PLI) of Verilog-XL. The gate-level power model used in PPP is presented in Bogliolo et al. [1995]. It allows accurate and efficient power estimations with accuracy within 5% from HSPICE, even for local single-pattern estimates. The adaptive characterization procedure of the power model is described next.

We refer to the simple situation in Figure 11, in which two instances of the same soft macro (namely, an 8-bit adder) are to be synthesized in the context of a larger design. We choose a carry-lookahead trial implementation for the adders and run the adaptive modeling procedure to build the power consumption model for the two units. In the Verilog description, each unit is represented by a module containing both the gate-level implementation of the carry-lookahead and a task calling the LMS routine.

During characterization, the gate-level implementation of the two units is accurately simulated in the context of an RTL simulation of the entire design. In this way, typical input vectors (with actual timing) are automatically generated for Add1 and Add2 as a typical test sequence is applied at the primary inputs of the design.

Suppose that the circuit is to be used at a frequency of 50MHz. The characterization phase then consists of applying a sequence of input patterns with a 20ns time period. According to Eq. (8), the kth *training step*

(i.e., the kth sample) for the model of a unit is based on its switching activity $\mathbf{a}^{(k)}$ at time $k \cdot 20$ns and on its energy consumption $e^{(k)}$ in the time period $[k \cdot 20\text{ns}, (k + 1) \cdot 20\text{ns}]$. Both $\mathbf{a}^{(k)}$ and $e^{(k)}$ are available at the end of the time period, when all transient phenomena are extinguished: the value of $e^{(k)}$ is provided by the gate-level simulation of the implementation, while $\mathbf{a}^{(k)}$ is obtained as the exclusive OR between the current and previous values of the I/O vectors.

During characterization, the adaptive procedure is automatically called at the end of each time period for each unit in the design. At the first call, a linear model is created having an independent variable for each input/output signal and a default vector of coefficients, $\mathbf{c}^{(0)}$. Since the initial assignment of the fitting coefficients does not impact the final model, we use 0 as default value for the elements of $\mathbf{c}^{(0)}$. Nevertheless, in an iterative design flow, the coefficients of an already characterized model can be used as an initial guess to speed-up subsequent recharacterizations after marginal design improvements.

At subsequent calls, Eq. (8) is used to update coefficients, with a convergence speed controlled by $\mu = 0.1/(1 + n + m)$. The characterization process terminates when the model reaches convergence. Convergence is tested with the following procedure. First, LMS is run for at least $1 + n + m$ samples (to prevent premature convergence). After the first steps, a "moving window" average error ϵ_{av} is computed, representing the average error of the macromodel over the last $1 + n + m$ samples. Notice that ϵ_{av} is updated for every new sample. The variations of ϵ_{av} are monitored. Whenever its peak-to-peak variation over $1 + n + m$ successive values falls below 10% of its average value, LMS adaptation is assumed to have reached convergence. In case of slow convergence, we also set an upper limit to the number of LMS iterations. This limit is set to $20 \cdot (n + m + 1)$. The final value of $\mathbf{c}$ is stored in a file in order to be used for subsequent power analysis or recharacterizations.

In our example situation, both units are simultaneously characterized and their modeling processes take the same time. In general, however, different units may require different characterization times. Whenever the model of a unit has reached convergence, its gate-level implementation is automatically disabled and replaced by the RTL model to speed-up simulation.

Furthermore, the overall size of the units under characterization may exceed the limiting size for gate-level simulation. This may happen either because the overall design have been synthesized at the same time, or because several synthesis tasks have been performed before running the characterization step. In both cases, the set of units can be partitioned and the characterization step repeated for each subset. In our implementation, a control mechanism is available that automatically selects and characterizes clusters of units without stopping simulation. This is done by switching on the fly between gate-level and RT-level representations.

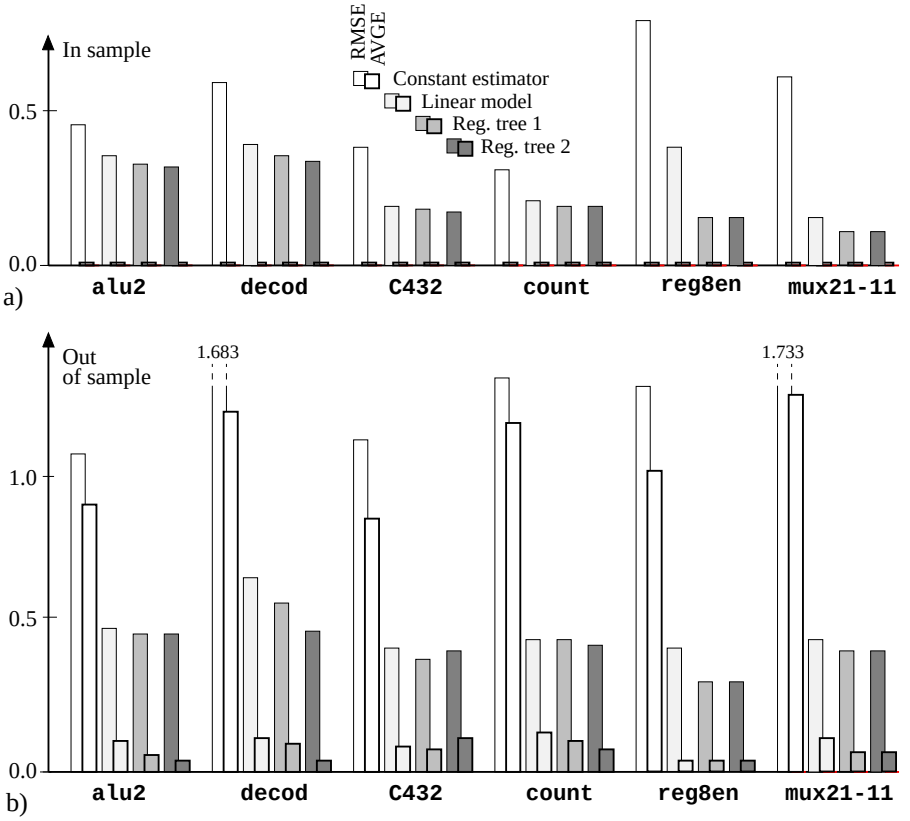


Fig. 12. Comparison of energy estimates provided by constant estimators, linear models, and regression trees of depth 1 and 2, for some benchmark circuits: `alu2`, `decod`, `C432` and `count` are taken from the MCNC benchmark suite, `reg8en` is an 8-bit register with enable, `mux21-11` is an 11-bit multiplexer. (a) In sample accuracy (i.e., accuracy obtained using the same input statistics of the sample used for characterization); (b) out of a sample accuracy (i.e., accuracy obtained using reduced input activity).

5. EXPERIMENTAL RESULTS

We performed three sets of experiments to (i) validate our regression models on benchmark circuits; (ii) test their accuracy on a real-world complex design; and (iii) evaluate the effectiveness of *in situ* model tuning. Experimental results are discussed in detail in the following sections.

5.1 Model Validation on Benchmark Circuits

First of all, we checked the advantage of using linear and nonparametric regression models instead of constant (pattern-independent) estimators. For this experiment we used combinational benchmarks from the MCNC suite [Yang et al. 1991], as well as gate-level implementations of logic and arithmetic functions and registers. Circuits were mapped on a library of gates with accurate power models and simulated with PPP [Bogliolo et al. 1996].

For each circuit, a random sequence of $20(n + m + 1)$ test patterns with 50% signal and transition probabilities was used to perform *off-line* characterization. The average power consumption measured during the characterization phase was taken as a constant estimator while least squares fitting was used to perform linear regression. Moreover, the tree regression procedure was run twice on each benchmark to characterize regression trees of depth 1 and 2. As expected, when there are no explicit control signals (as in most MCNC benchmarks) splitting is mainly used to capture non-linearities and splitting variables are usually selected among transition flags (i.e., among the independent variables of linear equations). In contrast, when control inputs that determine the operational mode of the unit exist, their values are automatically recognized as more significant splitting variables. For instance, the power models of registers and multiplexers were automatically split on the initial values of *enable* and *select* signals, respectively.

Accuracy was tested running concurrent gate-level and RT-level simulations. Two different test sequences (of 200 vectors each) were used: the first one with the same input statistics used for characterization to test *in-sample* accuracy; the second one with a lower average input activity (of 20% instead of 50%), to test *out-of-sample* accuracy. Representative experimental results are reported in Figure 12 in terms of two measures of error: the relative root mean square error ($RMSE = \sqrt{MSE}/e_{avg}$) and the relative error on the average estimate ($AVGE = |e_{avg} - e_{avg}^{est}|/e_{avg}$).

In-sample accuracy is shown in Figure 12(a). By construction, all models give the same (correct) average estimate ($AVGE$ is always negligible), but the $RMSE$ is different, and gives us a significant idea of improved quality of pattern-dependent models. We can observe that regression trees perform better than simple linear regressions, and their $RMSE$ decreases when we increase their depth.

Out-of-sample results are reported in Figure 12(b). In this case, both error measures are significant and give us a better idea of the improved flexibility of pattern-dependent models. While the accuracy of the constant estimator is unacceptably degraded both in average and instantaneous power estimates, the error made by pattern-dependent models on average power estimates is around 15%. Unfortunately, the RMSE is quite high, proving that the models cannot be used to provide accurate instantaneous power estimates.

Although it is clear that linear regression outperforms simple pattern-independent models, the choice between regression trees and standard linear regression is not straightforward. Regression trees are superior by construction when usage patterns are similar to those used for characterization, while standard linear regression may lead to better results when different input statistics are used. We remark that, for each benchmark circuit, we used the same sample of power consumption data to characterize all power models. However, the statistical significance of the least squares solution depends on the ratio between the sample size s and the

Table I. Macros Instantiated in the Datapath of Figure 13

Nome	n	m	Gates
CMPXX-11	22	1	48
CMPGT-11	24	2	45
EXPSBS	25	13	110
INC-11	12	12	51
MUX21-11	24	11	48
MUX51-11	60	11	118

number of fitting coefficients to be determined. In our experiment the ratio was $s/1$ for the constant estimator $s/(n + m + 1)$ for the linear model, and $s/(n + m + 1)2^k$ for a regression tree of depth k. If we keep splitting the model without increasing the size of the sample, we reduce the statistical significance of leaf regressions, eventually impairing the quality of the power estimates provided by the model. For benchmark circuit C432, the reduced sample size per fitting coefficient was responsible for the degradation of the out-of-sample estimates provided by a regression tree of depth 2. More robust regression trees could be obtained by increasing the sample size.

5.2 Accuracy Evaluation on a Complex Design

We tested our RTL power estimates on a complex design of practical interest, namely, a fully-functional high-performance IEEE standard floating-point adder/subtracter in double precision described in Verilog HDL. The design consisted of four units: the mantissa datapath (53–bits wide), the exponent datapath (11–bits wide), the rounding logic and the control logic (to set the various rounding modes and to select floating-point sum or subtraction). The adder was designed to perform an addition/subtraction per clock cycle. The design was built starting from a library of hard macros. We discuss the power estimation of the exponent computation block.

Looking at Figure 13, we observe that several inputs to the exponent logic are controls coming from the mantissa datapath and the control logic. Additionally, the design has several internal signal reconvergences. Obviously, the uniform white input distribution hypothesis is not the only one that is not valid for macros in the exponent logic, but we cannot even assume any simple distribution (such as that proposed in Landman and Rabaey [1995]) for the numerous control inputs. Moreover, different instances of the same macro have completely different I/O statistics, depending on their location within the circuit.

We precharacterized both linear regression models and 1-level regression trees for the six building blocks of the exponent datapath (see Table I). According to a typical characterization paradigm, macros were characterized off-line using uniformly distributed random patterns with no relation with those provided during simulation. Power models were then used to back-annotate the functional description of each macro and evaluated during a fully-RTL simulation of the whole floating-point adder (with

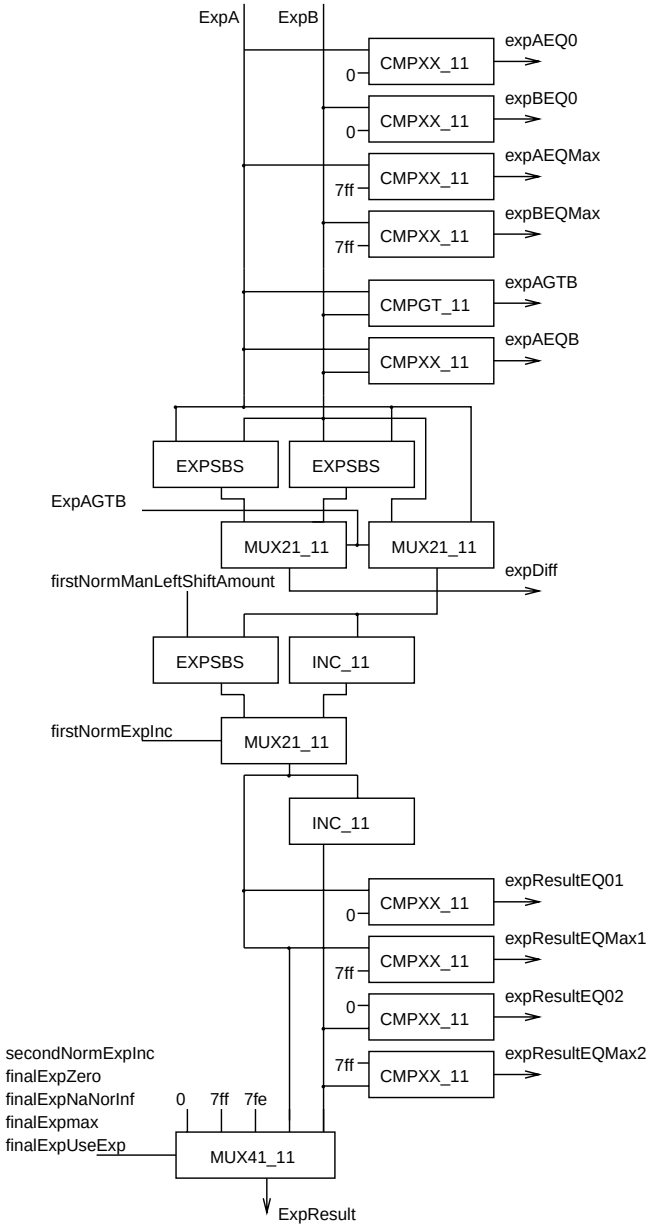


Fig. 13. High-level schematic of the exponent logic of a double-precision IEEE standard floating-point adder.

power estimation mode activated only for the units of the exponent block). Assuming the macros to be external IPs, their regression-based power models could be precharacterized and supplied by the IP provider, thus enabling the user to perform accurate power simulations without actually knowing low-level implementation details.

Table II. Experimental Results for 19 Units in the Design of Figure 13. GLS (*gate level simulation*) denotes the reference values of power consumption for a clock period of 100ns. Three estimates are compared: LR (*liner models*), RG1 (*regression trees of depth 1*), and SPA (*static power analysis* based on the same regression tree)

Unit		GLS	AVGE (%)		
Instance	Macro	(μW)	LR	RT1	SPA
AZero	CMPXX-11	368	16.5	10.9	9.5
AMax		339	9.3	5.3	6.9
BZero		372	16.3	12.2	11.0
BMax		337	9.4	4.1	3.8
EQCompare		449	5.5	3.5	4.0
ResultZero1		704	35.1	28.1	28.6
ResultZero2		686	27.9	21.9	22.0
ResultMax1		651	31.2	21.5	23.2
ResultMax2		640	25.4	20.8	21.0
GTCompare	CMPGT-11	448	12.9	10.1	10.0
SubAminusB	EXPSBS	1178	9.1	1.2	3.7
SubBminusA		1216	9.0	1.6	2.9
ExpAdj1		729	17.2	38.5	39.2
ExpInc1	INC-11	585	16.3	16.3	16.3
ExpInc2		842	10.4	10.4	10.4
ExpFirstNormSelect	MUX21-11	742	22.2	6.8	7.6
ExpDiffSel		392	18.0	44.1	43.3
ExpTopSel		546	12.7	9.8	11.4
ExpFinalSelect	MUX51-11	596	23.9	17.5	18.1
Entire Data Path		**11820**	**15.2**	**7.7**	**8.5**

We used an event-driven simulation paradigm with a *constant-delay model* (an average propagation delay statically associated with each macro) for signal propagation through combinational macros. The event-driven paradigm allowed us to account for multiple transitions caused by the propagation of signals through paths of unequal length. As shown in Example 5, the pure cycle-based simulation paradigm causes an unacceptable degradation in the quality of the power estimates. Event-driven simulation trades off some simulation performance for increased accuracy in estimation.

Three different power estimators were used for each of the 19 units in the exponent logic: (i) linear regression models (LR); (ii) regression trees of depth 1 (RT1) used in the context of an event-driven RTL simulation; and (iii) static power analysis (SPA) based on the same regression trees. Signal and transition probabilities for SPA were collected during the same simulation run used for the dynamic evaluation of LR and RT1.

Table II reports the relative errors (AVGE) for the instances of the units and for the complete block, with respect to accurate gate-level simulations with real delay models. The compensation between overestimates and underestimates for the units explains why the global average error is smaller than the error on the single blocks. Notice also that, for some instances of the macros, the error is large, so it would be misleading to assume that comparisons between the estimated power consumption of

Table III. Comparison Between Linear Power Models Characterized *off-line* by Least Squares Fitting and Adaptive Models Characterized *in situ* for 19 Units in the Design of Figure 13

Unit		Off-Line Model		Adaptive Model	
Instance	Macro	RMSE	AVGE	RMSE	AVGE
AZero	CMPXX-11	23.8	11.3	8.1	0.2
AMax		24.8	8.9	14.6	2.4
BZero		23.1	10.2	8.2	3.1
BMax		25.0	1.1	14.7	2.3
EQCompare		21.5	11.6	15.9	5.3
ResultZero1		61.1	60.7	38.5	9.2
ResultZero2		60.8	58.4	41.0	6.1
ResultMax1		59.4	57.6	39.9	7.4
ResultMax2		63.9	55.5	37.3	7.0
GTCompare	CMPGT-11	18.1	2.5	15.4	1.7
SubAminusB	EXPSBS	23.2	7.1	17.9	3.6
SubBminusA		24.8	6.9	17.7	1.6
ExpAdj1		59.6	22.3	39.1	10.1
ExpInc1	INC-11	61.9	50.8	38.7	14.3
ExpInc2		66.2	65.8	41.3	7.5
ExpFirstNormSelect	MUX21-11	72.3	69.4	48.4	15.6
ExpDiffSel		35.2	14.9	33.3	14.3
ExpTopSel		44.2	43.4	22.6	10.2
ExpFinalSelect	MUX51-11	79.6	77.1	39.6	7.6
Entire Data Path		**42.3**	**34.6**	**22.9**	**6.1**

different instances within the design can be made with the same degree of confidence. The range in accuracy for estimating the power consumed by the units is due to widely varying input statistics. Regression trees generally outperform linear regression, but sometimes they perform substantially worse (namely, for ExpAdj1 and ExpDiffSel). This is another proof of the trade-off between accuracy and robustness that complicates the choice between linear regression and regression trees.

The last column refers to the static evaluation of the regression tree based on transition probabilities. We observe that some inaccuracy is sometimes introduced by the correlation between the splitting variable and (some of) the other ones. Notice that, for the two instances of macro INC-11, there are no differences between the three power estimators. In fact, the splitting criterion used to construct the regression tree was not satisfied and the automatic characterization procedure returned a traditional regression model instead of a regression tree. As long as linear models are used, static power analysis based on transition probabilities is equivalent to dynamic evaluation performed during simulation.

Finally, we remark that the RMSE of the power estimates provided by all models was sometimes larger than 50%, further demonstrating that regression models do not provide significant pattern-by-pattern estimates. Nevertheless, pattern dependence grants them flexibility.

5.3 Effect of In-Situ Model Tuning

A last set of experiments was run on the design of Figure 13 to verify the effectiveness of the adaptive characterization procedure performed *in situ*. To this purpose, we entered the validation phase of the iterative design flow of our example floating-point adder, with adaptive linear models associated with each unit in the exponent datapath. The adaptive model of a unit is nothing but a linear regression equation whose coefficients can be adaptively modifyed by the LMS algorithm. We used the same initial guess for the fitting coefficients of all instances of the same macro: namely, the coefficients of the linear models characterized off-line. We then ran concurrent RTL and gate-level simulation to adjust the fitting coefficients in order to improve the accuracy of the power models of each instance, as described in Section 4.

Using PPP, the full floating-point adder was simulated at the RT level, with the exception of macros in the exponent logic, for which gate-level implementations were also simulated to provide reference values for LMS. The primary input patterns provided to the adder were taken from those used for testing the functional correctness of the design.

The *in-situ* tuning of the adaptive linear models was performed concurrently for all instances in the exponent datapath. The gate-level representation of each instance was simulated only until convergence of the corresponding model. After convergence, the model was automatically used to back-annotate the RTL functional description of the unit, thus progressively switching back to a fully-RTL simulation with back-annotated power models.

Only 100 iterations were required for the LMS to converge for units with few I/O signals (such as the instances of CMPXX-11), while up to 400 training steps were required for units with more I/O signals (such as MUX51-11). As stated in Section 4, the initial guess does not affect the asymptotic value of the fitting coefficients. However, fewer iterations are usually required if the initial guess is close to the optimal assignment. We repeated the experiment using a null vector as a default initial guess for all power models, thus using LMS to characterize the linear models from scratch. About 800 iterations were required for all models to converge. To evaluate performance, consider that about 50 ms per pattern were required on a DEC5000/260 workstation to perform cycle-based simulation of the structural RTL description of the floating-point adder with no power information. The impact of evaluating regression models in a cycle-accurate context was negligible (below 5%), while the concurrent RTL and gate-level simulation of all the units in the exponent datapath took about 2s per pattern. The overall time spent in tuning the precharacterized power models was of about 10 minutes, while more than 20 minutes were spent to perform on-line characterization from scratch.

After convergence, 10000 patterns were simulated at the RTL to evaluate the quality of the adaptive regression models. Table III reports the accuracy of the power models of each instance, in terms of relative root mean

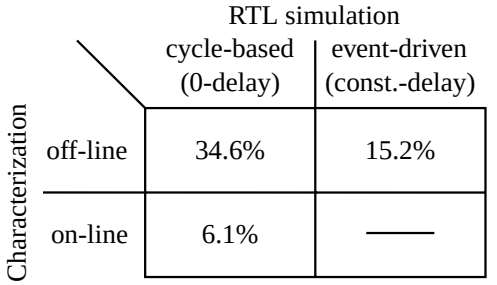


Fig. 14. Relative error of the average power estimates provided by linear regression models with different characterization and simulation paradigms. All data refer to the design of Figure 13.

square error (RMSE) and relative error on average estimate (AVGE) with respect to gate-level simulation. The accuracy of linear models characterized off-line (with uniform white inputs) is also reported for comparison. The RTL simulation of the entire design took less than 10 minutes, while more than 5 hours were spent to run the equivalent gate-level simulation.

The quality of the adaptive models characterized *in situ* is substantially higher than that of the off-line models: the average improvement is of 35% on RMSE and of 75% on AVGE. Notice also that error compensation is not improving the overall accuracy of the estimates. For the entire exponent logic, we obtained RMSE = 42.3 and AVGE = 34.6 using RTL models characterized off-line, and RMSE = 22.9 and AVGE = 6.1 using on-line characterization.

The off-line linear models reported here for comparison are exactly the same as in the previous experiment. However, the average errors reported in the fourth column of Table III are substantially worse (on average) than those in the fourth column of Table II. What changed is the simulation paradigm. As discussed in Example 5, off-line characterization cannot account for misaligned input transitions. When using precharacterized power models at the RTL, event-driven simulation is required to take timing into account, thus avoiding crude underestimates of actual power consumption. In contrast, if the power models are characterized *in situ*, the input stimuli for gate-level power simulation are directly provided by the environment with the actual timing. As a consequence, the reference power information provided by gate-level simulation implicitly accounts for real signal misalignments. If the power model for a unit is adaptively tuned in order to fit the real-delay power consumption at the cycle boundaries, timing information is no longer required at the RTL to obtain accurate power estimates. Timing will implicitly be taken into account by the adaptive models, even if they are evaluated at the RTL in a cycle-accurate context.

To further clarify this important feature of adaptive modeling, Figure 14 shows how the accuracy of the linear models depends on the characterization and simulation styles. If the models are characterized off-line (with arbitrary assumptions on input statistics and timing) and then evaluated

in an event-driven context (with constant delay models for event propagation through RTL macros), the relative error on the average power estimates for the floating-point adder is about 15%. If the same models are used in a cycle-based context, the relative error becomes larger than 34%. In fact, cycle-accurate simulation is inherently less accurate (and more efficient) than the event-driven one. However, the adaptive models characterized on-line and evaluated in a cycle-based context outperform both results, providing a relative error of about 6%. Event-driven simulation is no longer required to improve accuracy, as long as input misalignments are implicitly taken into account during characterization.

6. CONCLUSIONS

In this work we analyzed the application of linear and nonparametric regression for the automatic construction of RTL power macromodels for registers and combinational units. We proposed two approaches for the automatic construction of regression-based models: namely, off-line and on-line adaptive characterizations. During on-line characterization, power macromodels are tuned on the fly while simulating each macro instance within its environment (i.e., *in situ*). on-line characterization is based on the LMS algorithm, which has small computational overhead and guarantees asymptotic convergence to the optimum (in the least squares sense) regression model. Experimental results for practical designs show that on-line, *in-situ* characterization achieves high accuracy with reduced computational overhead.

ACKNOWLEDGMENTS

We thank Michele Favalli at University of Ferrara for his comments and suggestions.

REFERENCES

BENINI, L., BOGLIOLO, A., FAVALLI, M., AND DE MICHELI, G. 1996. Regression models for behavioral power estimation. In *Proceedings of the Workshop on Power and Timing Modeling, Optimization and Simulation* (Sept. 1996), 179–187.

BOGIOLO, A., BENINI, L., AND RICCÒ, B. 1996. Power estimation of cell-based CMOS circuits. In *Proceedings of the 33rd Annual Conference on Design Automation* (DAC '96, Las Vegas, NV, June 3–7), T. P. Pennino and E. J. Yoffa, Eds. ACM Press, New York, NY, 433–438.

BOGLIOLO, A., BENINI, L., DE MICHELI, G., AND RICCO, B. 1995. Accurate logic level power estimation. In *Proceedings of the IEEE Symposium on Low Power Electronics*, IEEE Computer Society Press, Los Alamitos, CA, 40–41.

BOWERMAN, B. L. AND O'CONNELL, R. T. 1990. Linear statistical models-An applied approach. PWS-Kent.

BREIMAN ET AL., L. 1993. *Classification and Regression Trees*. Chapman & Hall, London, UK.

GUPTA, S. AND NAJM, F. N. 1997. Power macromodeling for high level power estimation. In *Proceedings of the 34th Annual Conference on Design Automation* (DAC '97, Anaheim, CA, June 9–13), E. J. Yoffa, G. De Micheli, and J. M. Rabaey, Eds. ACM Press, New York, NY, 365–370.

HSIEH, C.-T., WU, Q., DING, C.-S., AND PEDRAM, M. 1996. Statistical sampling and regression analysis for RT-level power evaluation. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '96, San Jose, CA, Nov. 10–14, 1996), R. A. Rutenbar and R. H. J. M. Otten, Eds. IEEE Computer Society Press, Los Alamitos, CA, 583–588.

LANDMAN, P. E. AND RABAEY, J. M. 1995. Architectural power analysis: The dual bit type method. *IEEE Trans. Very Large Scale Integr. Syst. 3*, 2 (June 1995), 173–187.

LIOY, A. AND MACII ET AL., E. 1997. Accurate entropy calculation for large logic circuits based on output clustering. In *Proceedings of the Great Lakes Symposium on VLSI* (Mar.), 70–75.

MARCULESCU, D., MARCULESCU, R., AND PEDRAM, M. 1996. Information theoretic measures for power analysis. *IEEE Trans. Comput.-Aided Des. 15*, 6, 599–610.

MEHTA, H., OWENS, R. M., AND IRWIN, M. J. 1996. Energy characterization based on clustering. In *Proceedings of the 33rd Annual Conference on Design Automation* (DAC '96, Las Vegas, NV, June 3–7), T. P. Pennino and E. J. Yoffa, Eds. ACM Press, New York, NY, 702–707.

NAJM, F. N. 1994. A survey of power estimation techniques in VLSI circuits. *IEEE Trans. Very Large Scale Integr. Syst. 2*, 4 (Dec. 1994), 446–455.

NEMANI, M. AND NAJM, F. 1996. Towards a high-level power estimation capability. *IEEE Trans. Comput.-Aided Des. 15*, 6, 588–598.

POWELL, S. AND CHAU, P. 1990. Estimating power dissipation of VLSI signal processing chips: the PFA technique. *VLSI Tech. Bull. IV*, 250–259.

VSI ALLIANCE, 1997. Architecture document, Version 1.0. VSI Alliance. http://www.vsi.org/library.htm

WIDROW, B. AND STEARNS, S. D. 1985. *Adaptive Signal Processing*. Prentice-Hall signal processing series. Prentice-Hall, Inc., Upper Saddle River, NJ.

YANG, S. 1991. Logic synthesis and optimization benchmarks user guide version 3.0. Microelectronics Center of North Carolina, Research Triangle Park, NC.

Received: August 1997; revised: February 1998; accepted: October 1998