High-Level Macro-Modeling and Estimation Techniques for Switching Activity and Power Consumption

Anand Raghunathan, Sujit Dey, and Niraj K. Jha

***Abstract*—We present efficient techniques for estimating switching activity and power consumption at the register-transfer level (RTL), using a combination of macro-modeling for datapath blocks, and control logic analysis techniques based on partial delay information. Previous work on estimating switching activity and power at the RTL has ignored the presence of glitches at various datapath and control signals. We demonstrate that glitches can form a significant component of the switching activity at signals in typical RTL circuits. In particular, for control-flow intensive designs, we show that the controller substantially affects the activity and power consumption in the datapath due to the presence of glitches at control signals. Since the final implementation of the controller is not available during high-level design iterations, we develop techniques that estimate glitching activity at control signals using control expressions and partial delay information. For datapath blocks that operate on word-level data, we construct piecewise linear models that capture the variation of output glitching activity and power consumption with various word-level parameters like mean, standard deviation, spatial and temporal correlations, and glitching activity at the block's inputs. For RTL blocks that operate on bit vectors that need not have an associated word-level value, we present accurate bit-level modeling techniques for glitching activity as well as power consumption. This allows us to perform accurate power estimation for control-flow intensive circuits, where most of the power consumed is dissipated in non-arithmetic components like multiplexers, registers, vector logic operators, etc. Experimental results on several RTL designs demonstrate the accuracy of the proposed estimation techniques. Our RTL power estimator produced estimates that were within 7% of those produced by an in-house power analysis tool on the final gate-level implementation, while being over 50× faster than its gate-level counterpart.**

***Index Terms*—Control logic, datapath, glitching, low-power design, macromodels, micro-architecture, power estimation, register transfer level.**

I. Introduction

TECHNIQUES for evaluating a design for various metrics like area, delay, and power consumption at all levels of the design hierarchy are an important part of the design process. While it is typically the case that lower level estimation tools offer higher estimation accuracy, their use to explore architectural tradeoffs during higher-level design tends to be prohibitively time-consuming. Several efficient techniques for estimating area and delay during high-level design have been proposed [1]–[4]. In this paper, we focus on the problem of estimating power consumption from RTL descriptions. Unlike previous approaches to high-level power estimation, our techniques are well-suited to control-flow intensive designs, which, as we show in this paper, have significantly different power consumption characteristics.

Recognizing the importance of power estimation, several researchers have investigated power estimation techniques ranging from the circuit level up to the system level. Circuit simulators like SPICE offer one of the most accurate ways of estimating power consumption but are also the most computationally expensive, which limits their use to only very small circuits. Transistor-level event-driven simulation techniques based on piecewise linear transistor models have been used to provide accurate full-chip power analysis much faster than SPICE [5]. However, they are still too expensive to use during each iteration of a design/synthesis loop. A large body of work has been devoted to logic-level power estimation for CMOS circuits. Logic-level techniques can be broadly classified as: 1) direct simulation-based techniques, which simulate the circuit response to specific input stimuli from which the power consumed can be computed; 2) techniques that compute signal value and transition probabilities at all the signals in the circuit, given the probabilities at the primary inputs, and use the computed probabilities to estimate power consumption; and 3) statistical or Monte-Carlo techniques that simulate generated input samples to the circuit till the monitored value of power consumption converges to within user-defined error and confidence levels. Power estimation techniques at the logic level are described in detail in [6]–[9].

With the trend toward designs starting at higher and higher levels of abstraction, researchers have devoted some attention to power estimation and optimization at the earlier stages of the design cycle, such as the register-transfer and behavior levels. Several studies have shown that the power optimization opportunities are significantly larger at the higher levels [6], [10]. Power analysis tools are required in order to:

1) validate that power budgets are met by the different parts of the design, and if not, identify the hot-spots in the design;
2) evaluate the effect of various optimizations and design modifications on power.

The use of high-level power analysis tools for the above purpose helps to greatly reduce the design cycle [9], [11], [12]. In

Manuscript received May 9, 2001; revised December 20, 2001.

A. Raghunathan is with NEC Laboratories America, Princeton, NJ 08540 USA (e-mail: anand@ccrl.nj.nec.com).

S. Dey is with the Department of Electrical and Computer Enigeering, University of California San Diego, La Jolla, CA 92093 USA.

N. K. Jha is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA.

Digital Object Identifier 10.1109/TVLSI.2003.812295

the absence of high-level power analysis tools, a power analysis iteration (e.g., to evaluate a design modification or alternative architecture) requires the designer to first synthesize and validate a lower level netlist, and then run a logic- or transistor-level power analysis tool to report power consumption. The large run times required by lower level power analysis tools, and to synthesize and validate a gate- or transistor-level netlist make this methodology highly inefficient for exploring high-level design tradeoffs, and infeasible for use in automatic high-level and system-level synthesis and optimization tools. In a design flow that uses high-level power analysis tools, tradeoffs at each level of the design hierarchy are supported by corresponding power analysis tools at the same level, leading to fewer and faster design iterations [11], [12].

In general, estimation techniques at higher levels of design abstraction tend to be more efficient due to the associated reduction in complexity (size) of the design, and sometimes due to the easy availability of functional information which may be difficult to extract at the lower levels.

The reduced complexity of power analysis at the higher levels does not come without a penalty. The absolute accuracy of high-level power analysis tools tends to be lower than analysis tools at the lower levels of the design hierarchy. However, high-level power analysis tools are still very useful to guide high-level design tradeoffs, if their results provide relative accuracy (i.e., they are able to correctly predict whether a design modification will result in an increase or decrease in power consumption) and monotonicity (i.e., they are able to properly rank order a set of candidate designs in terms of power consumption) [11], [12]. With the use of high-level power analysis tools for exploring design tradeoffs, the role of lower level power analysis tools is limited to supporting lower level optimizations, and verifying that the power budgets are met with a high level of confidence.

II. Related Work

In this section, we briefly describe the previous work on RTL/architectural power estimation techniques. We classify all the previous work into three broad approaches to power estimation, namely *analytical models*, *characterization-based macro-modeling*, and *control logic analysis techniques*. It is important to note that it is often necessary to use different estimation techniques for different parts of a design (such as arithmetic macro blocks, control logic, memory, clock network, and I/O).

Analytical power modeling techniques attempt to correlate power consumption to measures of design complexity, using very little information from the functional specification. For example, the chip estimation system [13] computes the power consumed in a logic block based on its estimated gate count, the average switching energy and load capacitance per gate, the clock frequency, and the average switching activity factor. The user is required to provide estimates for each of these parameters, leaving a lot to his/her judgment. While such techniques are error-prone in general, they may be accurate for some parts of a chip for which the complexity parameters are easy to estimate (e.g., memories and clock networks [14]). A recently proposed class of analytical techniques, called *information-theoretic approaches*, estimate average activity and capacitance factors for logic blocks based on the entropy of their input and output signals [15]–[19].

In **characterization-based macro-modeling**, the idea is to obtain and characterize a lower level implementation of circuit macro-blocks (that may be already available in the case of "hard" or "firm" macro blocks, or may need to be synthesized in the case of "soft" macro blocks). A gate-level or transistor-level tool is used to estimate the power consumption of the macro block for various input sequences, called *training sequences*. Based on this data, a macro-model or "black-box" model is constructed, which describes the power consumption of the block as a function of various parameters, e.g., the signal statistics of the block inputs and outputs. Characterization-based macro-models are best suited for bottom-up and meet-in-the-middle design methodologies [20], such as high-level synthesis based design flows, where hard or firm macro blocks can be instantiated from a component library. The accuracy of characterization-based macro-models stems from the fact that a lower level implementation is used to construct the macro-models. However, the training sequences used to construct the macro-model cannot be exhaustive due to efficiency considerations. Hence, a macro-model may be "biased" by the training sequences used during the characterization process. In addition to the above problem, macro-models may introduce inaccuracies since the results of the characterization experiments are fit into a pre-determined function template or model, resulting in some errors due to interpolation or extrapolation.

One of the early architecture-level power estimation techniques, the PFA technique [21], characterized architectural blocks by simulating their implementations with random input sequences. The inability of the PFA technique to account for the dependency of power on input signal statistics was addressed in the dual bit-type (DBT) method [22]. Activity-sensitive power models [22]–[32] alleviate the above deficiency by constructing and utilizing a model for power consumption that is a function of the signal statistics at a macro block's boundaries.

Control logic analysis techniques deal with the control or random logic parts of a design. The activity-based control (ABC) model [24] is an example of a controller power estimation technique. Since the control logic is typically small in size compared to the datapath, several approaches perform a fast synthesis of the controller, followed by a gate-level simulation to estimate its power consumption [33]. In addition to computing the power consumed in the control logic, it is also very important to take into account its impact on the power consumption of the rest of the design. The importance of considering spatial and temporal correlations between control signals was demonstrated in [34]. It is also important to take into account the glitching activity at the control signals since it can have a significant impact on the power consumption in the rest of the design [28].

III. Paper Overview and Summary of Contributions

The aim of this paper is to provide high-level switching activity and power estimation techniques that are sufficiently

accurate and efficient to drive various high-level design optimizations, e.g., high-level synthesis subtasks such as module selection, scheduling, clock selection, resource sharing, etc. Our estimation techniques work at the structural RTL, where we assume that the circuit is a network of datapath components (macro blocks) and control logic. We present improvements over the state-of-the-art in the areas of characterization-based macro block power estimation, and fast control logic analysis during high-level design.

Even though it is known that glitching can account for a significant part of a circuit's power consumption and that estimating glitching power is necessary for obtaining even relative accuracy at the RTL, previous work on estimating switching activity at the RTL has ignored the presence of glitches at various datapath and control signals. For example, known characterization-based macro-modeling techniques for datapath blocks consider glitching within RTL blocks, but assume that the inputs to the blocks are glitch-free. We demonstrate that glitches form a significant component of the switching activity in typical RTL circuits, and present techniques to estimate the glitching activity at control as well as datapath signals in RTL circuits. We also develop glitching-activity-sensitive power models for various RTL blocks.

For blocks that operate on word-level data, we present systematic procedures to construct piecewise linear models that capture the variation of output glitching activity and power consumption with word-level signal statistics like mean, standard deviation, spatial and temporal correlations, and glitching activity at the block's inputs. We believe that these word-level modeling techniques improve on the state-of-the-art because they enable us to exploit efficient word-level simulation techniques for macro blocks, do not constrain the glitching activity or power function (e.g., to a linear or quadratic function), and do not require an understanding of the internals of a macro block during the characterization process. For RTL blocks that operate on data that may not have an associated word-level value, we present accurate bit-level modeling techniques for glitching activity as well as power consumption. This allows us to perform accurate power estimation for control-flow intensive circuits, in which a large fraction of the total power consumed is dissipated in non-arithmetic components like multiplexers, vector logic operators, etc., and where the effects of bit-level statistics may not be well reflected by word-level signal statistics.

While the control logic itself may often consume only a small fraction of the total power (e.g., 5%), it has a significant bearing on the power consumption in the datapath. Thus, it is important to accurately estimate signal statistics, including glitching activities, at control signals. A significant bottleneck to control logic analysis is the lack of complete delay information and interdependence of datapath and control logic timing. We present novel techniques that use partial delay information to identify and utilize timing relationships between control logic signals (at the RTL) that will hold with a high degree of confidence at the lower levels as well.

Experimental results on several RTL designs demonstrate the accuracy of the proposed estimation techniques. The RTL estimates obtained are within 7% of those produced by an in-house power analysis tool on the final gate-level implementation, while switching activity estimates are typically within 10% of an accurate gate-level estimation. Hence, we believe that our techniques can be used to efficiently assess the power tradeoffs involved in making various high-level design decisions.

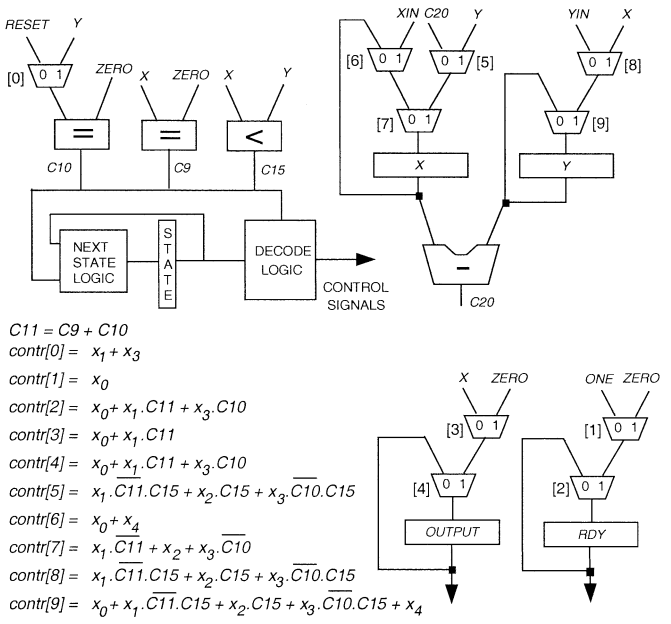


Fig. 1. GCD RTL circuit.

IV. Motivation

We illustrate some of the important features of control-flow intensive designs through the analysis of an example RTL circuit shown in Fig. 1, which computes the greatest common divisor (GCD) of two numbers. The RTL blocks used in the GCD datapath are one subtracter, three comparators [one less-than $(<)$ and two equal-to $(=)$], registers, and multiplexers. The controller is subdivided into the state register, the next state logic, and the decode logic that generates the control signals for the datapath. The control expressions implemented by the decode logic are also given in Fig. 1. The variables $x_0, \ldots, x_4$ in the control expressions represent the decoded state variables, i.e., x_i is 1 when the controller is in state s_i. The control expressions also involve status signals generated from the datapath like the outputs of comparators. While the datapath typically consists of several predesigned macro blocks, the control logic is often subject to logic synthesis optimizations before it is mapped to the technology library.

The GCD RTL circuit was mapped to NEC's CMOS6 technology library [35], and NEC's in-house simulation-based gate-level power estimation tool, CSIM [36], was used to monitor the switching activity, both including and excluding glitches, at selected datapath and control signals.[1] It bears mentioning at this point that since CSIM is a commercial tool, it has been calibrated with SPICE and benchmarked within 10% of

[1]The simulator models each $0 \rightarrow 1$ or $1 \rightarrow 0$ transition as half a transition, leading to fractional numbers used in examples throughout this paper. The simulator uses an inertial delay model to capture the effect of glitch attenuation at gates.

TABLE I
ACTIVITIES WITH/WITHOUT GLITCHES FOR VARIOUS SIGNALS OF THE GCD CIRCUIT

Control signal	Activity		Datapath signal	Activity	
	Total	W/O Gl.		Total	W/O Gl.
$contr[0]$	71	70.5	$dp2[7..0]$	71.5	21.5
$contr[1]$	22	22	$dp4[7..0]$	92	26
$contr[2]$	72	20	$dp5[7..0]$	1124.5	247
$contr[3]$	42	20	$dp7[7..0]$	1044.5	273
$contr[4]$	72	20	$dp9[7..0]$	321.5	80.5
$contr[5]$	55.5	54			
$contr[6]$	22	22			
$contr[7]$	50	20			
$contr[8]$	55.5	54			
$contr[9]$	77	70.5			

SPICE. The power and delay models for individual library cells used in CSIM were constructed using SPICE. It incorporates several state-of-the-art gate-level power simulation techniques, including state-dependent power modeling, accurate glitch filtering using inertial delay model, etc. The results are reported in Table I. The numbers shown in the table demonstrate that switching activity estimates that ignore glitches can be quite inaccurate for data as well as control signals.

Conventional RTL power estimation techniques would compute the power consumed in each of the RTL blocks in the GCD circuit using only zero-delay activity information at the block inputs. The inaccuracies in switching activity estimates in turn lead to inaccuracies in RTL estimates for power consumption. In order to explore the ramifications of the above assumption, we performed the following experiments. First, the entire GCD circuit implementation was simulated using CSIM and several typical input sequences, to estimate its average power consumption. The power consumption was reported to be 1.64 mW. This figure includes the effect of glitch propagation across RTL blocks, since the entire circuit was used in the simulation. Next, we performed an RTL simulation[2] using the same input trace, and collected traces at the inputs of each embedded circuit block. The implementation of each RTL block in the GCD circuit was simulated separately (the controller was considered as a single block) using the zero-delay traces derived in the previous step, and the individual power estimates were accrued to yield a power estimate of 1.32 mW for the entire circuit (an under estimate of 19.5%).

As mentioned in Section I, glitches generated at the control signals (outputs of the controller) themselves have a significant impact on the datapath power consumption. In order to study this effect, we performed another experiment with the GCD RTL circuit. The circuit was partitioned into two separate subcircuits, the datapath, and the controller. Zero-delay traces were collected at the inputs of each subcircuit using RTL simulation, and used to simulate implementations of the controller and datapath *separately*. Thus, the effects of glitches at: 1) the datapath outputs (status signals) that feed the controller; and 2) the controller outputs (control signals) that feed the datapath were ignored for this experiment, while glitch proliferation within each subcircuit was considered. The sum of the individual power estimates for the controller and datapath was 1.45 mW, indicating that it is important to consider the effects of glitches on the status and control signals during switching activity and power estimation for the datapath. In comparison, when we applied the glitching activity and power estimation techniques presented in this paper to the GCD circuit, we obtained an estimate of 1.53 mW, that corresponds to an error of 6.7% compared to running gate-level power estimation on the entire gate-level circuit.

[2]The RTL simulation results in zero-delay traces (that do not include glitches) at various datapath and control signals.

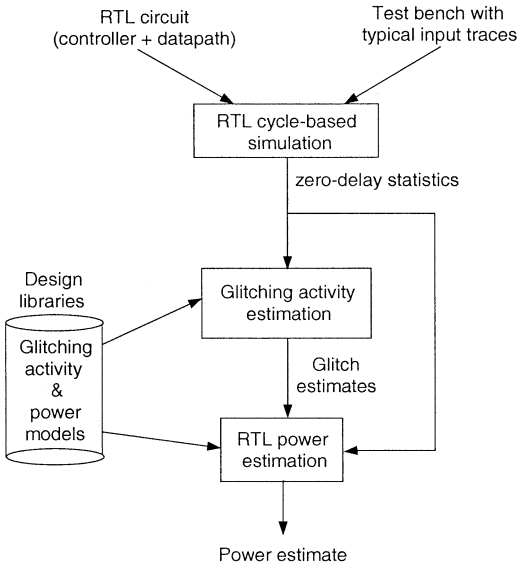


Fig. 2. Overview of our RTL power estimation tool flow.

V. RTL POWER ESTIMATION METHODOLOGY

The flowchart shown in Fig. 2 provides an overview of our RTL power estimation technique, which consists of three separate phases. The aim of the first phase is to obtain the zero-delay statistics at various signals in the RTL circuit. We use RTL cycle-based simulation for this purpose. The given RTL circuit description is simulated using a test bench containing a long sequence of typical input sequences, created using an understanding of the design's functionality and interface. However, these test benches could also be automatically generated in order to conform to given primary input statistics. During the simulation run, we collect a variety of word- and bit-level signal statistics that are used during the later phases of our technique. The advantages of using simulation to calculate zero-delay statistics are its flexibility (it is relatively straightforward to use the procedure for mixed-level descriptions, e.g., logic-RTL and RTL-behavioral descriptions), and the high speed of RTL cycle-based simulation. However, it is also possible to easily integrate other techniques of calculating zero-delay statistics (e.g., entropy based techniques, or Monte-Carlo techniques) into our power estimation methodology.

As mentioned in Section IV, switching activity estimates for various datapath and control signals can be highly underestimated if glitch generation and propagation in the RTL circuit are ignored. The second phase of our tool augments the zero-delay signal statistics derived in the first phase by

estimating glitching activity for signals in the RTL circuit. As we illustrate later, glitching activity at the output of an RTL block depends on the zero-delay signal statistics as well as glitching activity at the block's inputs. Hence, the glitching activity estimation procedure traverses the RTL circuit starting at primary inputs/register outputs, which are assumed to be glitch-free, and "propagates" glitching activity information forward through RTL circuit blocks until we reach primary outputs/register inputs. The glitching activity estimation phase is explained in more detail in Sections VI and VII.

In the third phase, the zero-delay signal statistics and glitching activity estimates derived in the first two phases are used to calculate power consumption in each RTL module. For this purpose, we develop and use power models for various RTL blocks like functional units, comparators, multiplexers, and registers. The power model captures the variation of power consumption as a function of zero-delay as well as glitching activity at the block's inputs. Our procedure for the generation and use of power models differs from those presented in [22] by accounting for glitches, and the fact that we use bit-level models for RTL blocks that operate on bit-vectors that may not be associated with a word-level value (e.g., multiplexers, registers, bit-vector concatenation and splitting, bitwise logic operations, etc.). Our power modeling techniques are explained in detail in Section VIII.

VI. Estimating Glitching Activity in the Controller

In this section, we develop techniques to estimate glitch generation and propagation through the controller, which can significantly affect the total power consumption as shown in the previous section. The controller's inputs are the status signals from the datapath (possibly including outputs of functional units such as comparators, or data register values), while its outputs are the control signals that feed the datapath. Clearly, glitch generation and propagation in the control logic can be exactly estimated only if detailed information regarding the structure of the controller implementation and delays are provided. However, the final implementation of the controller is typically *not available* during high-level design iterations. Moreover, the timing properties of signals within the control logic depend on timing information of the status signals (controller inputs), which in turn may transitively depend on timing information at other datapath signals. Thus, deriving accurate timing information at control signals could require synthesizing the entire controller and datapath, which is often prohibitively expensive to perform within high-level design iterations. We propose techniques to compute the switching activity at control signals by combining zero-delay activity estimates derived from RTL simulation, with glitching activity estimates derived from *control expressions*, as explained in this section.

The control logic is typically represented as *control expressions* during the high-level synthesis process. These control expressions are expressed in the form

$$\text{contr} = \sum_i x_i . \left(\prod_j Cij \right) \tag{1}$$

where x_i represents a decoded controller state variable (corresponding to controller state s_i), Cij represents a status signal (e.g., the output, or inverted output, of a comparator from the datapath), $\sum$ represents the Boolean OR operation, and $\prod$ and . represent the Boolean AND operation. Each product term in the control expression is derived to flag the occurrence of a particular combination of values at the status signals when the controller state is s_i. Note, that, the representation for the control logic described above assumes that the encoding used for the controller state is known. However, there is no restriction on the encoding itself, i.e., any desired state encoding can be used for the controller.

In general, glitching activity may be present at a control signal due to the following reasons. The status signals (Cij) and decoded state signals may themselves carry glitches, which propagate through the control logic, causing the control signals to be glitchy. On the other hand, the control logic can also generate a significant amount of glitches. Our glitching activity estimation procedure is based on separately estimating and summing, for each gate in the control logic, the glitching activity generated in the gate and the glitching activity propagated through the gate from its inputs.

A. Estimating Glitch Generation in the Controller

Glitch generation occurs when a gate's input signals satisfy certain logic (values assumed by the gate's inputs) and temporal (timing relationship between events at the gate's inputs) conditions. While the logic conditions can be easily monitored during functional simulation, the interdependence of datapath and control logic timing makes it difficult to estimate control logic timing accurately without synthesizing the complete circuit. In this section, we describe a novel technique to estimate glitch generation that is based on extracting and using *partial delay information* in the form of timing relationships between signals in the control logic.

In general, the *logic conditions* necessary for glitch generation at a gate during an interval of time are as follows:

- there should be at least one rising and at least one falling transition at the gate's inputs;
- no input should assume a steady controlling[3] logic value throughout the interval under consideration.

Assuming an inertial delay model, the temporal condition for glitch generation in an AND gate is as follows:

- the earliest falling transition arrives after the latest rising transition by an interval that is greater than the gate's inertial delay.

Similar conditions can be derived for glitch generation in other types of gates.

Given a control expression in a sum-of-products form, as shown in (1), the logic conditions for glitch generation for each product term (conjunctive or AND expression) and the disjunctive (OR) expression combining the product terms can be easily monitored during the first phase of our RTL power estimation procedure, i.e., zero-delay RTL simulation. Checking whether

[3]A controlling input value for a gate uniquely determines the value at the gate output, irrespective of values at the gate's other inputs.

the temporal conditions for glitch generation are satisfied in an accurate manner, however, is not as straightforward. Accurately predicting the generation of glitches at a gate requires a knowledge of the exact times at which each of the gate's inputs makes a transition, if any, in each clock cycle. One possible approach that has been suggested in other contexts in power estimation to tackle the lack of accurate delay information is to make a pessimistic assumption, i.e., assume that glitches are generated at a gate whenever the logic conditions for glitch generation are satisfied [37]. However, in practice, this pessimistic assumption often leads to substantial over-estimates of glitches at control signals, as shown in the following example.

Example 1: Consider the control signal $\mathrm{contr}[2]$ in the GCD RTL circuit of Fig. 1. The control expression for $\mathrm{contr}[2]$ is $x_0 + x_1.C11 + x_3.C10$. We would like to estimate the glitching activity at control signal $\mathrm{contr}[2]$, given the traces for each of the decoded state and comparator output signals that were captured during the zero-delay RTL simulation phase. In this case, signals $C10$, $C11$, x_1, x_3 were found to be glitch-free,[4] simplifying the problem to that of estimating glitch *generation* at $\mathrm{contr}[2]$.

For the time being, let us make pessimistic assumptions to tackle the lack of availability of complete temporal information, i.e., we conclude that glitches are generated whenever the logic conditions for glitch generation are satisfied. Clearly, the first product term (x_0) cannot generate any glitches. From the simulation traces, we computed the statistics for logic conditions for glitch generation at the second and third product terms

$$\begin{aligned}
&\text{Case 1}: \mathrm{Count}(x_1 \downarrow, C11 \uparrow) = 15,\\
&\text{Case 2}: \mathrm{Count}(x_1 \uparrow, C11 \downarrow) = 20,\\
&\text{Case 3}: \mathrm{Count}(x_3 \downarrow, C10 \uparrow) = 35,\\
&\text{Case 4}: \mathrm{Count}(x_3 \uparrow, C10 \downarrow) = 30.
\end{aligned}$$

In the above equations, the symbols $\uparrow$ and $\downarrow$ denote the rising and falling transitions, respectively. The expression $\mathrm{Count}(x_1 \downarrow, C11 \uparrow)$ represents the number of instances (consecutive pairs of cycles) in the simulation trace where x_1 makes a falling transition, while $C11$ simultaneously makes a rising transition. Since these numbers refer to counts of events that occur at different points in time (clock cycles), they can be added. Thus, we conclude that the glitching activity generated due to the second and third product terms is 35 and 65, respectively. The glitches generated at each product term propagate to the output unmitigated, since the decoded state variables are mutually exclusive. In addition, for the given simulation traces, it was observed that the logic conditions for glitch generation were *never* satisfied for the disjunctive (OR) term. Hence, the glitching activity at control signal $\mathrm{contr}[2]$ was estimated to be 100 transitions over the entire simulation period. A comparison with the glitching activity observed using CSIM for the same input traces and reported in Table I $(72 - 20 = 52)$ shows that the glitching activity at $\mathrm{contr}[2]$ was over-estimated by the pessimistic approach by as much as 92.3%! ■

[4]In general, we do not make any assumptions about the absence of glitching at the status inputs or decoded state variables—any subset of them could be glitchy.

Although exact arrival time information at various signals is not available, it is often possible to derive *partial information* about delays from RTL descriptions or during high-level synthesis. For example, the outputs of comparators can often be assumed to arrive later than the decoded present state signals, even when we do not have any knowledge of their exact arrival times. In general, inputs to each gate in the control logic can be divided into *early* arriving signals, *late* arriving signals, and signals whose arrival time information is assumed to be *unknown*. Even a small set of relative timing relationships between inputs to a gate can be used to refine glitch generation estimates, as shown in the following example.

Example 2: Let us revisit control signal $\mathrm{contr}[2]$ in the GCD circuit that was used for the discussions in Example 1. Suppose we are allowed to make the assumption that the comparator output signals, $C10$ and $C11$, arrive after the decoded state variables, x_0, x_1, and x_3. Consider Case 1 ($x_1 \downarrow$, $C11 \uparrow$) in the equations presented above. Since the rising transition arrives later than the falling transition in this case, the temporal conditions for glitch generation are *not satisfied* for this case. Similarly, it can be seen that Case 3 does not satisfy the temporal conditions for glitch generation in the third product term. The revised glitching activity estimate for $\mathrm{contr}[2]$ is 50, which represents an error of only 4% with respect to the number reported by CSIM. ■

The explanations used in the above example can be generalized as follows. RTL timing analysis techniques [1], [2], [4] have been proposed for controller/datapath circuits, which can provide an estimate of the minimum clock period of the implementation. These techniques can identify the longest topological as well as sensitizable (true) paths using bit-level delay models for datapath macro blocks and gate delay models for control logic. For our subsequent explanations, we assume that we have two procedures, RTL_MAX_DELAY_EST() and RTL_MIN_DELAY_EST(), which compute the latest and earliest arrival times (i.e., the longest and shortest path), respectively, for each signal in the control logic. Suppose that this information is pre-computed and stored in two arrays $\mathrm{Latest\_Arrivals}$ and $\mathrm{Earliest\_Arrivals}$, as indicated by procedure DERIVE_RTL_DELAY_INFORMATION shown in Fig. 3. Consider two signals i and j in the control logic for which transitions have been detected in a cycle of zero-delay simulation. If $\mathrm{Earliest\_Arrivals}[i] - \mathrm{Latest\_Arrivals}[j] > k$, we can conclude that the transition at signal i arrives after the transition at signal j with an error margin of k. The larger the value of k, the larger the confidence we have that the above conclusion will hold at the lower levels. Fig. 3 shows a procedure GATE_GLITCH_GENERATION which, when invoked on each gate in the control logic after each cycle of zero-delay simulation, will determine whether glitch generation occurred at the gate output. A user-specified parameter, *Thresh*, that is passed to this procedure selects an appropriate safety factor or error margin used for inferring partial delay relationships. Setting *Thresh* to be equal to a value larger than the longest path in the RTL circuit will result in the use of the pessimistic approach (without any delay information) for estimating glitch generation. Lower values of *Thresh* naturally provide less

```
Procedure DERIVE_RTL_DELAY_INFORMATION(RTL_Ckt, Library)
{
    Latest_Arrivals = RTL_MAX_DELAY_EST(RTL_Ckt, library);
    Earliest_Arrivals = RTL_MIN_DELAY_EST(RTL_Ckt, Library);
}
Procedure GATE_GLITCH_GENERATION(Gate, Thresh, Latest_Arrivals, Earliest_Arrivals)
{
    latest_nc_c = 0;
    earliest_c_nc = ∞
    foreach input signal in_i of gate {
        if(in_i remains at a controlling value) {
            return(0); //controlling value, hence no glitch generation
        }
        if(in_i has a controlling→non-controlling transition) {
            earliest_c_nc = min(earliest_c_nc, Earliest_Arrivals[in_i]);
        }
        if(in_i has a non-controlling→controlling transition) {
            latest_nc_c = max(latest_nc_c, Latest_Arrivals[in_i]);
        }
    }
    if(earliest_c_nc == ∞ OR latest_nc_c == 0) {
        return(0); //only one type of transition, hence no glitch generation
    }
    if(earliest_c_nc − latest_nc_c > Thresh) {
        return(0); //no glitch generation possible, based on partial delay info.
    } else {
        return(1);
    }
}
```

Fig. 3. Procedure for estimating glitch generation at a gate in the control logic using partial delay information.

pessimistic estimates, however, very low values of *Thresh* will make the estimate more sensitive to variations introduced by interconnect between RTL components, effects of random logic synthesis, clock skew, etc. From our experiments, we found that setting *Thresh* to 15% of the circuit's longest path produced consistent and reasonable results.

B. Glitch Propagation Through the Control Logic

We next explain our procedures for estimating glitch propagation through the control logic, i.e., from the inputs of the control logic or internal signals to the controller's outputs. We use a procedure based on the gate-level activity estimation technique proposed in [38] for this purpose. The procedure is illustrated below.

Consider again the generic control expression given in (1). Consider a particular comparator output, $C1$, which we have predicted to be glitchy based on our datapath glitching activity models. Let us rewrite the control expression by separating the product terms into terms in which $C1$ appears, terms in which $\bar{C}1$ appears, and terms that do not depend on $C1$

$$\text{contr} = C1.\text{contr}_{C1} + \bar{C}1.\text{contr}_{\bar{C}1} + \sum_{\text{product terms indep. of } C1} x_i \cdot \prod_j Cj$$

$$\text{contr}_{C1} = \sum_{\text{product terms with } C1} x_i \prod_{C_j \neq C1} Cj$$

$$\text{contr}_{\bar{C}1} = \sum_{\text{product terms with } \bar{C}1} x_i \prod_{Cj \neq \bar{C}1} Cj. \quad (2)$$

In order for glitches at $C1$ to propagate to the control signal, at least one of the product terms it is involved in must have noncontrolling side inputs (i.e., 1), and the result of all other product terms should evaluate to 0. Hence, the following equation can be utilized to estimate the propagation of glitches to the control signal

$$Gl(C1) * \mathrm{P}\left((\text{contr}_{C1} = 1 \text{ OR } \text{contr}_{\bar{C}1} = 1) \text{ AND } \sum_{\text{product terms indep. of } C1} x_i \cdot \prod_j Cj = 0 \right). \quad (3)$$

In (3), $Gl(C1)$ represents the glitching activity at $C1$. The term $P(.)$ in (3) can be thought of as the probability that the control signal will be "sensitized" to glitches at $C1$. This probability can be computed easily during the zero-delay RTL simulation phase. The same technique can be applied to model the propagation of glitches from other controller inputs and internal signals (i.e., outputs of each product term) to a controller output.

The following example illustrates a potential inaccuracy of the above procedure for estimating glitch propagation, and proposes a new technique to address the problem.

Example 3: Consider control signal $\mathrm{contr}[5]$ of the GCD RTL circuit. The control expression for $\mathrm{contr}[5]$ is $x_1.C\bar{1}1.C15 + x_2.C15 + x_3.C\bar{1}0.C15$. For this example, we focus on the propagation of glitches from $C15$ to $\mathrm{contr}[5]$ through the product term $x_2.C15$. Using (3), and the observation that the product terms are mutually exclusive, we can establish the contribution of the product term of interest to be $Gl(C15)*P(x_2 = 1)$. From the RTL simulation traces, we calculated the value of $P(x_2 = 1)$ to be 0.647. Combining this number with the estimated glitching activity at $C15$, we obtain an estimate of 27.3 (cumulative activity for the entire simulation run) for the glitching activity due to the term $x_2.C15$. However, the glitching activity reported by CSIM for the entire simulation run was only 1.0! Upon further investigation into this discrepancy, it was observed that the conditions for glitch generation at signal $C15$ were *negatively correlated* to the conditions for glitch propagation through the chosen product term. In other words, in consecutive pairs of cycles in which the controller made a state transition into state s_2, the data inputs of the comparator were actually unchanged, leading to no propagation of glitches from $C15$ through the product term $x_2.C15$.

We resolve the above problem by predicting the glitches at $C15$ *separately* for each controller state. For example, in order to predict the glitch propagation for the product term $x_2.C15$, we estimate the glitching activity at $C15$ for only those consecutive pairs of cycles where the final controller state is s_2. Since we are decomposing the glitches at $C15$ into separate estimates for each state, we refer to this technique as *glitching activity decomposition*. In general, the process of glitching activity decomposition involves identifying a set of disjoint conditions whose probabilities add up to 1, and estimating glitching activity separately under each condition. For example, the set of conditions $\{\mathrm{state} = s_1, \mathrm{state} = s_2, \mathrm{state} = s_3, \mathrm{state} = s_4\}$ satisfies this requirement. In our work, we only use the controller state as the basis for glitching activity decomposition (since the controller state often determines whether the data values at the inputs to the comparators change, and since we empirically found it to yield satisfactory results). The benefit of glitching activity decomposition is that it exposes any correlations between the conditions required for glitch generation and those required for glitch propagation, leading to an improvement in the accuracy of the glitch estimates. Note, that in order to compute a separate figure for glitching activity at $C15$ in state s_2, we must in turn compute separate figures in state s_2 for the zero-delay statistics and glitching activity at the inputs of the comparator that generates $C15$. However, in practice we did not observe any computational bottlenecks due to decomposition, since 1) the decomposition of statistics by state was limited to only the transitive fanins of those comparators that were found to generate glitches and 2) separate statistics were computed only for those states that were related to a glitchy comparator output through a product term. In the current example, we found that when the controller made a state transition to state s_2, the temporal correlation at the inputs of the comparator was very high, leading to minimal generation of glitches in s_2. The predicted glitching activity at the output of the AND gate implementing $x_2.C15$ now becomes 3.0, which is much closer to the number reported by CSIM. ■

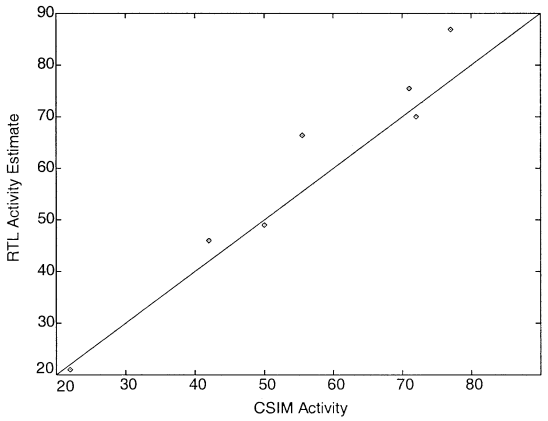


Fig. 4. Scatter plot of switching activity at control signals. RTL estimate versus CSIM.

In order to get a feel for the accuracy of our switching activity estimation techniques for control signals, we provide results comparing our estimates to the switching activity measured after a complete gate-level implementation using CSIM, for all distinct control signals in the GCD RTL circuit (except $\mathrm{contr}[1]$, which is not glitchy). The scatter plot shown in Fig. 4 shows the results of our experiment. The x-coordinate represents the total switching activity reported by CSIM for the control signal, while the y-coordinate represents the switching activity estimated using our RTL activity estimation procedure. As a reference, the plot also shows a solid line for the equation $y = x$, i.e., points in the scatter plot close to this line indicate a high accuracy in the RTL estimates. The figure indicates that our control signal switching activity estimation techniques produce estimates that are quite close to the activity numbers obtained using CSIM after a time-consuming implementation of the complete GCD controller and datapath.

We believe that our control expression based glitching analysis techniques work satisfactorily for control logic in RTL circuits generated through high-level synthesis tools, as opposed to arbitrary logic circuits. Also, note that while each individual control expression is represented as a two-level expression, that is not the same as flattening the entire cone of logic driving the control signals into two-levels. If we trace back the cone of logic driving a control signal until we reach registers, it may contain several cascaded components, including datapath components (arithmetic units, comparators), and control expressions. Thus, the accuracy of the control logic switching activity estimates should be attributed to a combination of various techniques, and not just the control expression based techniques.

VII. Modeling Glitch Generation and Propagation in Datapath Blocks

For datapath blocks which operate on multibit input signals that are associated with a word-level value (e.g., adders, subtracters, multipliers, and various comparators), previous work

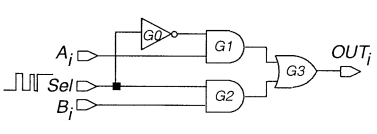
[22] has shown that it is possible to construct activity-sensitive power models that utilize the statistics (mean, standard deviation, etc.) of the word-level value associated with each multibit input signal, rather than consider signal statistics for each input bit. Several datapath blocks, however, do not associate any word-level value to their multibit input signals. Common examples of such blocks are multiplexers, registers, vector logic operations, logic shift units, etc. We model each bit-slice of such units separately.

A. Bit-Level Glitch Generation and Propagation Models

Bit-level modeling allows us to build more accurate glitching activity models, and to consider the effects of bit-level statistics that may not be well reflected by word-level signal statistics in certain situations. For instance, if a bit-vector signal b, consisting of n bits, is split into two smaller bit-vectors b_1 and b_2 that consist of k and $n-k$ bits, respectively, we would like to estimate the glitching activity at b_1 and b_2 separately. The extra computational effort spent here is well justified for control-flow intensive designs where the total circuit power consumption may be dominated by power consumption in multiplexers, registers, and bit-manipulation operators. Note, that bit-level models are constructed only for modules where different bit-slices do not have any dependency. By dependency, we mean that the output of one bit-slice is an input to another bit-slice. The case, where different bit-slices share an input, is handled by our approach, as illustrated in the following example using a multiplexer. In addition, the efficiency of bit-level modeling derives from the fact that each bit-slice depends on a reasonably small number of input bits. In the case of, say, a barrel shifter (with variable bidirectional shift), each output bit can depend on all the input bits. Such a block would be subject to word-level macro-modeling in our approach.

We illustrate bit-level glitching activity models through the example of a 2-to-1 multiplexer. We first develop a simple model for the generation of glitches in a multiplexer when its inputs are glitch free. A multiplexer bit-slice has two data input bits A_i and B_i, and one select input Sel. The number of distinct input vector pairs that can be applied at A_i, B_i, and Sel is $2^3 * 2^3 = 64$. Since the above number is small, it is possible to simulate the implementation of a 1-bit multiplexer for the exhaustive set of 64 input vector pairs, and build a look-up table that stores the glitches generated at the output for each vector pair. The lookup table can be thought of as a six-dimensional array $\mathrm{Mux\_gl\_gen}[]$, and the entry of the table corresponding to present and previous input values $A_i(t), B_i(t), \mathrm{Sel}(t), A_i(t-1), B_i(t-1), \mathrm{Sel}(t-1)$ is written as $\mathrm{Mux\_gl\_gen}[A_i(t), B_i(t), \mathrm{Sel}(t), \mathrm{A}_i(t-1), \mathrm{B}_i(t-1), \mathrm{Sel}(t-1)]$. During the zero-delay RTL simulation phase, we compute the glitch generation at each bit-slice of a multiplexer by looking up the appropriate entry of the $\mathrm{Mux\_gl\_gen}[]$ table.

The output of a multiplexer can also be glitchy due to the propagation of glitches from the data and select inputs. We model the propagation of glitches from a data input to the multiplexer output as being "regulated" by the probability that the glitchy data input is selected. For a 1-bit slice, assuming A_i is selected when $\mathrm{Sel} = 0$, the glitching activity at the multiplexer output due to propagation from A_i is given by



A	B	W/O Gl.	With Gl.
0	0	0.5	0.5
0	1	1.0	3.0
1	0	1.0	3.0
1	1	0.5	3.5

Fig. 5. Modeling propagation of glitches from a multiplexer select signal.

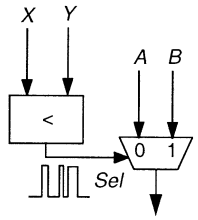


Fig. 6. Circuit used to compute the coefficients D_{01}, D_{10}, and D_{11}.

$\mathrm{Gl}(\mathrm{A}_i) * \mathrm{P}(\mathrm{Sel} = 0)$. A similar explanation holds for the propagation of glitches from B_i. The glitches from the select signal of a multiplexer may propagate to the output through multiple reconvergent paths, depending on the values of the data inputs, as shown in the following example.

Example 4: Consider the gate-level implementation of an embedded multiplexer bit-slice that is shown in Fig. 5. The table shown in Fig. 5 reports the glitches at the multiplexer output for all possible values of the data signal bits A_i and B_i. In the $\langle 0,0 \rangle$ case, glitches on select signal Sel are killed at AND gates $G1$ and $G2$ due to controlling side inputs. When data inputs are $\langle 0,1 \rangle (\langle 1,0 \rangle)$, glitches on Sel propagate through gates $G2$ and $G3$ ($G1$ and $G3$). Finally, when data inputs are $\langle 1,1 \rangle$, glitches on Sel propagate through gates $G1$ and $G2$. The output of the multiplexer is glitchy as a result of the interaction of the glitchy signal waveforms at $G1$ and $G2$. The exact manner in which the waveforms interact depends on the propagation and inertial delays of the various wires and gates in the implementation. We conclude that the glitch propagation from the select input of a multiplexer to its output is affected by the *spatial correlation* between the data inputs. Hence, simply measuring the signal probabilities at each data input bit to a multiplexer will not suffice. ■

Our model for glitch propagation from the select signal of the multiplexer to its output is given by the following:

$$Gl(Sel) * (D_{01} * P(A_i = 0, B_i = 1) + D_{10} * P(A_i = 1, B_i = 0) + D_{11} * P(A_i = 1, B_i = 1)).$$

The probabilities $P(A_i = 0, B_i = 1)$, $P(A_i = 1, B_i = 0)$, and $P(A_i = 1, B_i = 1)$ are monitored for each multiplexer bit-slice during the zero-delay simulation phase. The constants D_{01}, D_{10}, and D_{11} depend on the exact implementation of the multiplexer, and are computed by performing experiments using the circuit configuration shown in Fig. 6. The comparator is used to generate glitches at the select input to the multiplexer by feeding appropriate vector sequences at its inputs. In order to calculate D_{01}, the multiplexer's data inputs are fixed to $A = 0 \ldots 0$ and $B = 1 \ldots 1$. CSIM reports the values of $\mathrm{Gl}(Sel)$ and $\mathrm{Gl}(\mathrm{OUT}_i)$ for each i. Note that $Gl(OUT_i)$ also includes the effects of glitch generation in the multiplexer. Hence, we use the zero-delay traces at the multiplexer inputs to estimate glitch

generation in the multiplexer using the look-up table based procedure described earlier. We subtract the estimated glitch generation from $\mathrm{Gl}(\mathrm{OUT}_i)$, average the resulting difference over all i, and divide by the value of $\mathrm{Gl}(Sel)$ to obtain the value of D_{01}. The coefficients D_{10} and D_{11} are calculated similarly.

In summary, the glitching activity at the output of an n-bit multiplexer with data inputs A and B, select input Sel and output OUT is calculated using the following:

$$\begin{aligned}
\mathrm{Gl(OUT)} &= \sum_{i=1}^{n} \mathrm{Gl_Gen}(i) + \mathrm{Gl_Prop_From_}A_i \\
&\quad + \mathrm{Gl_Prop_From_}B_i \\
&\quad + \mathrm{Gl_Prop_From_}Sel \\
&\mathrm{Gl_Gen}(i) \text{ is accrued during RTL simulation} \\
&\qquad \text{using the Mux_}gl_gen[]\text{table.} \\
\mathrm{Gl_Prop_From_}A_i &= Gl(A_i) * P(Sel = 0) \\
\mathrm{Gl_Prop_From_}B_i &= Gl(B_i) * P(Sel = 1) \\
\mathrm{Gl_Prop_From_}Sel &= Gl(Sel) * (D_{01} * P(A_i = 0, B_i = 1) \\
&\quad + D_{10} * P(A_i = 1, B_1 = 0) \\
&\quad + D_{11} * P(A_i = 1, B_i = 1)). \qquad (4)
\end{aligned}$$

Note that the glitch generation model is dependent on the implementation of the multiplexer. However, since we are simulating the lower level netlist of the multiplexer when constructing the model, we will take this effect into account when constructing the glitch generation and propagation models. The example in Fig. 5 uses a simple implementation as an example. However, our models do not assume or require that a multiplexer is always implemented in this way. For example, in the case of a pass-transistor implementation, constant D_11 in the multiplexer glitch propagation model will be zero.

B. Word-Level Models for Glitching Activity

We next focus on datapath blocks that operate on multibit input signals that are associated with a word-level value (e.g., adders, subtracters, multipliers, and various comparators). As shown in [22], for such units it is possible to utilize the statistics (mean, standard deviation, etc.) of the word-level value associated with each multibit input signal rather than consider signal statistics for each input bit. The glitching activity at the output of an embedded datapath block depends on its functionality as well as its implementation details, zero-delay statistics at the input signals (e.g., mean, standard deviation, spatial and temporal correlations in the case of signals with a numeric value), and glitching activity at the inputs themselves.

It is possible to also use the signal statistics of the module outputs as parameters in the glitching and power macro models. For any input vector, the value at the output of a combinational macro block is completely determined by the values at the inputs of the module. However, the same property does not hold in the statistical domain. Since a specific set of signal statistics is only a "partial specification" of an input sequence, the input signal statistics may not completely determine the output signal statistics. Indeed, subsequent work has explored this possibility in a slightly different context [30], and shown that good accuracy can be obtained while employing a small number of variables in the macro model.

Our decision to use only input signal statistics was due to the fact that, in a statistical sense, we cannot really consider the statistical properties (mean, standard deviation, temporal correlation) of the output signal of a macro block as independent variables with respect to the statistical properties of its inputs. This has implications for both generation of patterns for characterization as well as the statistical analysis tests used (e.g., *ANOVA*[5]). However, with additional effort to address the above issues, we believe it is quite possible to extend the overall approach presented in this paper to consider output signal statistics.

We construct glitch models for various library components through a process of characterization using a lower (gate-)level implementation of the block, and a gate-level power simulation tool. The characterization process consists of constructing controlled experiments (simulation runs) by selectively varying one or more of the controllable variables (zero-delay statistics and glitching activities at the block inputs), and observing the value of the dependent variable (glitching activity at the block output). Given a set of sample data points obtained from the characterization experiments (observations), there are several statistical regression techniques [39] that can be used to build a model that predicts the output glitching activity. One possible approach is to attempt to fit a particular function (e.g., a linear function) to the observed data points, by tuning the function's parameters or constants in such a way that some metric (e.g., the sum of error squares) is minimized. However, it is often the case that the observed dependency of glitches at the output of an RTL block on some of the controllable variables is not well modeled by such simple relationships. An alternative approach, that we have found to be much more flexible and suited to automation is the use of one or more *piecewise linear models* for capturing the relationship between glitches and the various controllable variables.

We illustrate the process of deriving the glitching activity models for the case of an 8-bit subtracter with inputs named A and B, and output OUT. In general, the glitching activity at OUT can be written as

$$\begin{aligned}
\mathrm{Gl}_{\mathrm{OUT}} = f_{gl}(&\mathrm{Mean}_A, \mathrm{Mean}_B, \mathrm{SD}_A, \mathrm{SD}_B, \mathrm{TC}_A, \\
&\mathrm{TC}_B, \mathrm{SC}_{A,B}, \mathrm{Gl}_A, \mathrm{Gl}_B). \qquad (5)
\end{aligned}$$

The first seven parameters of $f_{gl}()$ represent the zero-delay signal statistics at A and B. Mean_A represents the mean or expected word-level value represented by signal A, SD_A represents its standard deviation, TC_A is the temporal correlation coefficient that represents the correlation between consecutive values that appear at signal A, $\mathrm{SC}_{A,B}$ is the spatial correlation coefficient of A and B [40]. Gl_A represents the glitching activity at A. Parameters with subscripts B have a similar meaning. The brute-force approach for building a model for $f_{gl}()$ would involve discretizing the range of variation of each of the parameters with a desired granularity, generating input sequences that correspond to each possible set of values for the parameters, and simulating the implementation of the subtracter to observe the glitching activity at the subtracter's output for

[5]ANOVA stands for Analysis Of VAriance, which is a popular technique used for statistical inference and testing.

each case. Assuming that each parameter can assume k possible values, the above approach will require k^n simulations, where n is the number of parameters or independent variables considered. In the case of the subtracter, $n = 9$, and even assuming $k = 5$ leads to 1.95 million simulation runs! Clearly, the brute-force approach is undesirable, in spite of the fact that building the models is a one-time cost for a given component library. We use two techniques to avoid the combinatorial explosion in the number of simulation runs required.

The first technique, called *variable elimination*, attempts to reduce the number of independent variables in the glitching activity model by identifying those variables whose variations affect the dependent variable (output glitches) minimally. We use techniques from multivariable data analysis for this purpose. Given a set of samples (each sample consists of a set of values for the independent variables $x_1, \ldots, x_n$, and the corresponding observed value that the dependent variable y assumes), we can use the *ANOVA* test to check whether the null hypothesis for any given variable x_i is true, i.e., whether different values of x_i had any impact on the observed sample values of y [39]. We used a commercial statistical analysis package, StatPlan IV [41] for performing ANOVA tests on our samples.

The second technique, called *model decomposition*, attempts to decompose the function $f_{gl}()$ into multiple subfunctions by partitioning the set of parameters into smaller groups of variables such that the effects of variables from different groups on the dependent variable interact minimally. Again, it is possible to use ANOVA techniques to obtain a quantitative evaluation of the interaction of the effects of two independent variables on the dependent variable from a given set of samples, as follows:

- for each pair of independent variables v_i and v_j, we compute the correlation coefficient that determines the effect of the interdependence of v_i and v_j on the dependent variable;
- we construct an edge-weighted undirected graph, called the *model decomposition graph*, in which vertices represent the independent model variables and an edge, whose weight equals the appropriate correlation coefficient determined in the previous step, exists between every pair of vertices; note that the model decomposition graph is a complete graph;
- we partition the model decomposition graph into cliques such that the sum of the clique weights is maximized (the weight of a clique is defined as the sum of its edge weights).

Each clique obtained in the last step represents a subfunction. The sum of the clique weights is an indicator of the quality of the decomposition. We impose a limit on the size of any clique in order to restrict the size of the resulting model subfunctions. Although the problem of minimum weight clique partitioning is $\mathcal{NP}$-hard [42], the sizes of the graphs we encountered in practice were quite small (number of vertices ≤ 10). Hence, we employed an exact branch-and-bound algorithm.

In the case of the subtracter, for example, the basic model of (5) can be decomposed into the following:

$$\mathrm{Gl_{OUT}} = f_{gl_1}(\mathrm{Mean}_A, \mathrm{Mean}_B) * f_{gl_2}(\mathrm{SD}_A, \mathrm{SD}_B) * f_{gl_3}(\mathrm{TC}_A, \mathrm{TC}_B) * f_{gl_4}(\mathrm{SC}_{A,B}) * f_{gl_5}(\mathrm{Gl}_A, \mathrm{Gl}_B). \quad (6)$$

The independent variables have been partitioned in (6) into the groups, $\{\mathrm{Mean}_A, \mathrm{Mean}_B\}$, $\{\mathrm{SD}_A, \mathrm{SD}_B\}$, $\{\mathrm{TC}_A, \mathrm{TC}_B\}$, $\{\mathrm{SC}_{A,B}\}$, and $\{\mathrm{Gl}_A, \mathrm{Gl}_B\}$. The above partition was based on the observation that variables within each group have a significant interaction in their effect on the dependent variable, while the variables in distinct groups are relatively independent in their effects. We investigated both additive (equivalently, subtractive) and multiplicative (equivalently, divisive) relationships between the subfunctions. The multiplicative relationship was chosen intuitively due to the property that a wider range of variation is possible in the product through a given variation of one of the multiplicands. In practice, it also resulted in better models in terms of mean square error or average absolute error at the fitting points. As before, assuming we discretize the domain for each parameter into five distinct regions, we would need to perform simulations for $5^2 + 5^2 + 5^2 + 5^1 + 5^2 = 105$ different sets of parameter values, which can be performed much more efficiently compared to the approach of building a single huge piecewise linear model from (5).

Having partitioned the set of independent variables into smaller groups as shown in (6), we proceed to build piecewise linear models for each of the subfunctions $f_{gl_1}(), \ldots, f_{gl_5}()$. Note that the subfunctions are composed using a multiplicative relationship. Hence, we can view one of the subfunctions, say $f_{gl_1}()$, as a *base power model*, and the remaining subfunctions as multiplicative correction factors. The base glitch model in the form of a contour plot is shown in Fig.7 (a). Consider the subfunction $f_{gl_2}(\mathrm{SD}_A, \mathrm{SD}_B)$. In order to construct the model for this subfunction, we perform controlled experiments where we first discretize the range of variation of SD_A and SD_B into a finite number of uniformly spaced points. For each point that corresponds to distinct values for SD_A and SD_B, we construct long vector sequences that have the desired values for SD_A and SD_B. Well-known algorithms exist to generate sequences whose means, standard deviations, and spatial and temporal correlations conform to desired values [43]. These methods assume a particular distribution (we used Gaussian or normal distributions for our experiments) that has zero mean and unit standard deviation, scale them to the desired mean and standard deviation, and then transform them so as to introduce the desired spatial and temporal correlations. The subtracter implementation is simulated using CSIM and the values of glitching activity at the output are recorded. The plot in Fig.7 (b) shows the results in the form of a contour plot. Note that the values in the plot are normalized to the case of $\mathrm{SD}_A = \mathrm{SD}_B = 25.6$, since $f_{gl_2}()$ represents a multiplicative correction factor to the base power model ($f_{gl_1}()$), which was derived assuming SD_A and SD_B to be fixed to 25.6. Given the values of SD_A and SD_B for an embedded subtracter, the value of $f_{gl_2}()$ is estimated from the values at the four points nearest to it in the discretized $\mathrm{SD}_A - \mathrm{SD}_B$ space, using standard linear interpolation techniques. The models for $f_{gl_3}(), \ldots, f_{gl_5}()$ are provided in Fig.7 (c)–(e).

In order to generate the model for $f_{gl_5}()$, we needed to generate input sequences to the subtracter that have varying *glitching activities* at A and B. In order to do that, we used the circuit configuration shown in Fig. 6. We added two multiplexers to feed the inputs of the subtracter, and connected

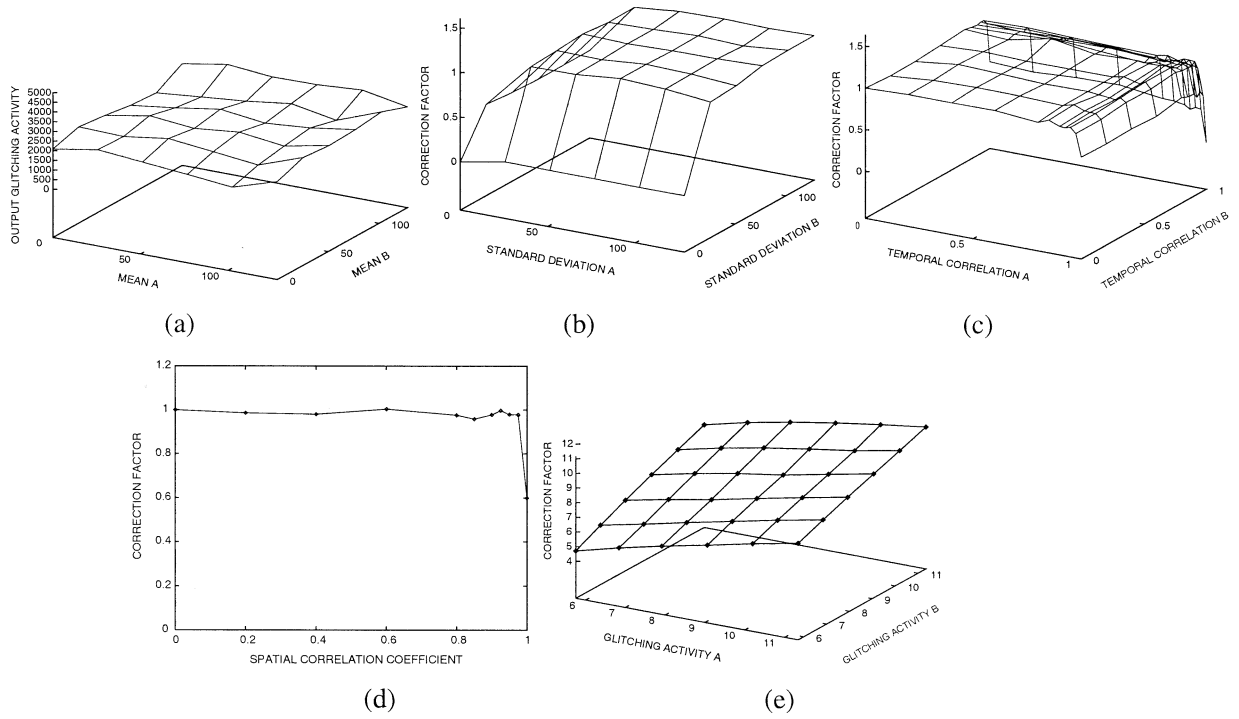


Fig. 7. Glitching activity models for an 8-bit subtracter.

their select input to the output of a $(<)$ comparator. Given a sequence of input vectors to the $(<)$ comparator that is known to generate glitches, we can control the glitches at the outputs of the multiplexers through the spatial correlations at their inputs, as illustrated earlier in this section.

VIII. RTL Power Models

Given the zero-delay and glitching activity estimates obtained from the first two phases of our tool, the third phase uses several word- and bit-level power models that we have developed for various RTL blocks, which we discuss in this section.

A. Word-Level Power Modeling Techniques

Our word-level black-box models and the procedures used to derive them differ from those presented in [22], the main conceptual difference being that our models also account for glitching activity at the inputs of a block. The process of deriving power models is similar to that of deriving glitching activity models, except that the dependent variable now becomes the total power consumption as opposed to the glitching activity at the output of the block. As explained in Section VII, we attempt to simplify the modeling process by using the techniques of variable elimination and model decomposition.

As an example, consider an 8-bit less-than $(<)$ comparator. The power model for the $(<)$ comparator is decomposed into subfunctions, and the model for each subfunction is illustrated by the plots in Fig. 8. Note that glitchy inputs can cause a substantial increase in the power consumption of the comparator, as shown in Fig.8 (e), confirming the importance of considering glitching activity at the inputs of various RTL blocks during estimation.

B. Bit-Level Power Models

For datapath blocks which do not associate any word-level value to their multibit input signals, we derive bit-level models for power consumption. The power models use the bit-level signal and zero-delay transition probabilities, correlations and glitching activity to calculate power consumption in each bit-slice of the block separately. For example, the power consumption in an n-bit multiplexer is modeled as follows:

$$\begin{aligned}
\text{Mux_Power} = &\sum_{i=1}^{n} \text{Base_Power}(i) \\
&+ Sel_\text{Glitch_Power}(i) \\
&+ A_\text{Glitch_Power}(i) \\
&+ B_\text{Glitch_Power}(i) \\
\text{Base_Power}(i) &\text{ is accrued during RTL simulation} \\
&\text{using the Mux_power[] table} \\
Sel_\text{Glitch_Power}(i) = &\, Gl(Sel) * (K_{Sel00} * P(A_i = 0, B_i = 0) \\
&+ K_{Sel01} * P(A_i = 0, B_i = 1) \\
&+ K_{Sel10} * P(A_i = 1, B_i = 0) \\
&+ K_{Sel11} * P(A_i = 1, B_i = 1)) \\
A_\text{Glitch_Power}(i) = &\, Gl(A_i) * (K_{A0} * P(Sel = 0) \\
&+ K_{A1} * P(Sel = 1)) \\
B_\text{Glitch_Power}(i) = &\, Gl(B_i) * (K_{B0} * P(Sel = 0) \\
&+ K_{B1} * P(Sel = 1))
\end{aligned} \tag{7}$$

The term Base_Power represents the power consumption in the multiplexer ignoring glitching activity at its inputs. The remaining three terms are additive correction factors used to capture the effect of glitches at the select and data inputs. As with the case of the glitch generation model, we apply all 64

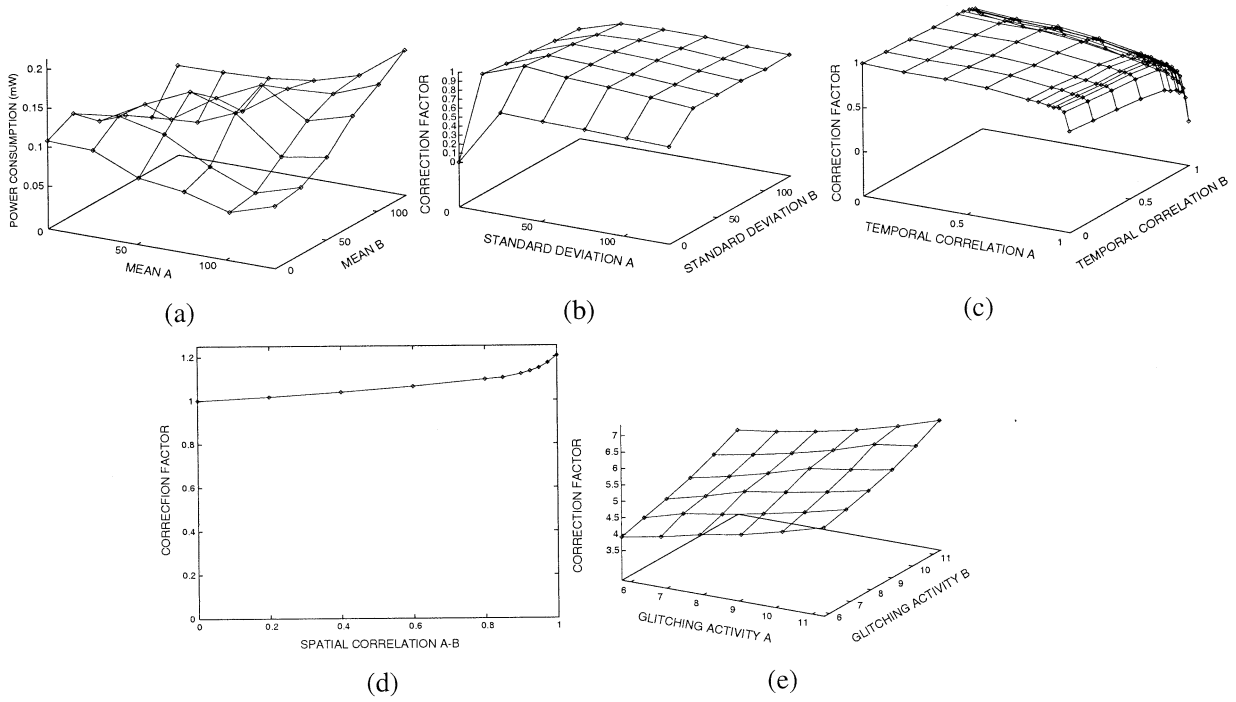


Fig. 8. Word-level power models for a less-than ($<$) comparator.

possible vector pairs to a 1-bit multiplexer, measure the power consumption using CSIM, and store the results in the form of a table called $\mathrm{Mux\_Power}[]$. During the RTL zero-delay simulation phase, we look up the entries of this table using the present and previous values at the inputs of each multiplexer bit-slice to determine its power consumption. The presence of glitches at the select input can significantly increase the power consumption in the multiplexer. We model the power consumption separately for the cases when the data inputs are 00, 01, 10, and 11. The coefficients $K_{Sel00}, \ldots, K_{Sel11}$ are obtained by performing controlled experiments with the circuit configuration shown in Fig. 6 by fixing the multiplexer data inputs to 00, 01, 10, and 11, feeding an input sequence to the comparator that causes it to generate glitches, observing the power consumption for the multiplexer reported by CSIM, and subtracting the base power factor to avoid double counting. Note that even when glitches at Sel do not propagate to the output, they could cause power consumption internal to the multiplexer. Hence, we have a separate K_{Sel00} term in the expression for $Sel\_\mathrm{Glitch\_Power}(i)$, even though such a term was not used in the corresponding glitching activity model for multiplexers. The purpose of $A\_\mathrm{Glitch\_Power}(i)$ and $B\_\mathrm{Glitch\_Power}(i)$ is to similarly account for the effect of glitching activity at A_i and B_i.

The power model for an n-bit register that has a data input IN, a clock input CLK, and output OUT, is given by the following:

$$\mathrm{Reg\_Power} = n * K_{\mathrm{CLK}} * \mathrm{Act(CLK)} + \sum_{i=1}^{n} (K_{\mathrm{OUT}_i} * \mathrm{Zero\_Del\_Act(OUT}_i) + K_{\mathrm{IN}_i} * \mathrm{Gl(IN}_i)) \quad (8)$$

The first term accounts for the power dissipated due to the clock line switching. The value of K_{CLK} is measured by simulating the implementation of a register while holding constant the value of IN. The second term accounts for the power dissipated due to the zero-delay activity at IN or OUT, i.e., when the value stored in the register changes. Note that $\mathrm{Zero\_Del\_Act(OUT}_i) = \mathrm{Zero\_Del\_Act(IN}_i)$. The value of K_{OUT_i} is determined by simulating the register under a long, glitch-free input sequence, subtracting the contribution of the clock transitions, and dividing the residual power per bit-slice by the average bit-level activity at the output of the register. The last term models the effect of glitches at the input of a register. The value of K_{IN_i} is determined by simulating the register under a glitchy input sequence and subtracting estimates for the effect of the first two terms. Note that usually $K_{\mathrm{OUT}_i} = K_{\mathrm{OUT}_j}$ for all i, j.

C. Controller Power Estimation

In order to obtain an estimate for the total RTL circuit power consumption, we must also estimate the power consumption in the controller. As mentioned previously, the complete controller implementation is typically not available until logic synthesis optimizations have been performed. Hence, it is difficult to obtain an accurate estimate of the controller power. In [22], models for controller power were proposed based on the target implementation style (e.g., PLA, ROM, standard cell), and the number of states, inputs, and outputs of the controller. The controller consists of a state register, next state logic, and decode (or output) logic. Control expressions are typically used to represent the functionality of the next-state logic and decode logic during high-level synthesis. For the purposes of estimating controller power, we assume that the next state and decode logic are implemented in a straightforward manner from their control expressions. The zero-delay switching activity at various signals in the controller is monitored during the RTL simulation. Later,

we estimate glitching activity using the control expressions, as explained in Section VI. We multiply the total (zero-delay + glitching) activity at the output of each gate in the controller with approximate values for gate output capacitance (we assume typical values for a 2-input AND gate, 2-input OR gate, and INVERTER) to yield a power consumption estimate for the controller. The power consumption in the state register is estimated using the model for register power presented earlier.

IX. The Power Estimation Procedure

We next give some details of our activity and power analysis procedures. We first parse and compile the RTL hardware description language (HDL) description into an RTL data structure that is described below. An RTL circuit is represented using a directed graph data structure, $RCG = (V, A)$. Each vertex $v \in V$ represents a circuit component, and each arc $a \in A$ represents an interconnection between components. Each vertex has a distinct type that can assume one of five values: PI, PO, REG, OP, $CONTROL$. $PI(PO)$ nodes represent primary inputs (primary outputs), REG nodes represent registers, OP nodes represent instances of operators from the RTL component library (including arithmetic and bit-vector operators, multiplexers, etc.), and $CONTROL$ nodes represent the logic that generates a control signal to the datapath. Note that the above classification is not restrictive, since OP nodes may represent arbitrary operators. As explained later, $CONTROL$ nodes represent the control logic using two-level sum-of-products expressions called *control expressions*. The RTL library is represented as a collection of library components. Each library component contains, among other information, a reference to its area, delay, and power models. Library components are assumed to have an annotation as to whether they are *bit-level macro blocks* or *word-level macro blocks*. This classification of library components into *bit-level blocks* and *word-level blocks* is performed a priori on the basis of their functionality. For example, an adder, subtracter, or multiplier is classified as a word-level block, while a multiplexer or vector Boolean operator is classified as a bit-level block. Note that once this straightforward classification is performed, the appropriate glitching and power models are automatically invoked. As explained below, computation of bit-level and word-level signal statistics is performed automatically for each signal. Also, note that each RTL circuit block has a reference to its library component, which in turn contains the appropriate glitching and power models.

The RTL circuit is levelized from primary input/register output to primary output/register input. Simulation and glitching analysis are performed by traversing the RTL circuit in levelized order. In addition, an RTL delay estimator [4] is used to derive partial delay information in the form of timing relationships between signals that feed the same $CONTROL$ node.

Cycle-based simulation is performed on the input trace. The primary input values from the next vector of the input trace are applied at the PI nodes. REG nodes contain values that were written into them at the end of the previous clock cycle. The RTL simulation uses a hybrid of bit-level and word-level simulation techniques for the OP nodes to improve simulation efficiency. For example, a 32-bit adder may be simulated using just one native instruction. This involves bit-to-word and word-to-bit conversion, which is automatically performed by our cycle-based simulation procedure. Note that any cycle-based simulator which allows us to access the values at each internal signal in the RTL circuit may be used in our power estimation procedure.

During the cycle-based simulation process, zero-delay signal statistics are collected for each arc. The statistics collected for an arc differ depending on the type of its sink vertex, and the library component it represents an instance of, as follows.

- For arcs feeding REG, $CONTROL$, and OP nodes that represent bit-level blocks, we store bit-level signal statistics. Bit-level statistics for an arc representing a bit-vector include: *signal probabilities and (rising and falling) transition probabilities for each bit*.
- For arcs feeding OP nodes that represent word-level blocks, we store word-level signal statistics. Word-level statistics for an arc include: *mean, standard deviation, and (word-level) temporal correlation coefficient*. Note that a single wire (signal) in the RTL circuit may fan out to multiple blocks of different types. Thus, each signal in the RTL circuit may correspond to multiple arcs in the graph representation, and some of those arcs may have bit-level statistics while others may have word-level statistics computed during simulation.

In addition to the above, for each vertex that has more than one incoming arc, we store the *spatial correlations* among its inputs. For $CONTROL$ and OP nodes that represent bit-level blocks, we store the complete spatial correlations for each input bit-slice. For example, for a OP node that represents a multiplexer, we store for each bit-slice the probability of occurrence of each of the $2^3 = 8$ input combinations. For $CONTROL$ nodes, we only store correlations between inputs that feed the same product term. It bears mentioning at this point that our signal statistics calculation procedure only computes second-order temporal correlations, and separates spatial and temporal correlations.[6] These assumptions are reasonable since we are decomposing/partitioning the RTL circuit into registers and combinational components and we are concerned only with the relevant correlations between the inputs of combinational circuit blocks. Global (and multicycle) correlations are automatically accounted for since our estimation methodology is simulation-based.

The glitch analysis procedure traverses the levelized RTL circuit, and at each node, computes the glitching activity at its output using the zero-delay signal statistics, as well as glitching activities, at its inputs. Again, depending on the node type, we

[6]For a macro-block with two input signals a and b, we may need an arbitrarily long history of input values, i.e., $\mathrm{a(t)}, \mathrm{a(t-1)}, \ldots$, and $\mathrm{b(t)}, \mathrm{b(t-1)}, \ldots$ to determine its power consumption. In order to capture the statistical properties of the input history of length $\mathrm{k(t \ldots t-k+1)}$, we need to store the correlation between all pairs of variables in $\{\mathrm{a(t) \ldots a(t-k+1)}, \mathrm{b(t) \ldots b(t-k+1)}\}$, i.e., order-k statistics. For combinational blocks and registers, it is sufficient to store order-2 statistics, i.e., $\{\mathrm{a(t)}, \mathrm{a(t-1)}\}$, $\{\mathrm{b(t)}, \mathrm{b(t-1)}\}$, $\{\mathrm{a(t)}, \mathrm{b(t)}\}$, $\{\mathrm{a(t)}, \mathrm{b(t-1)}\}$, and $\{\mathrm{a(t-1)}, \mathrm{b(t)}\}$. The approximation we make is that we only store the first three terms, and ignore the last two terms, which we found to be reasonable for RTL power estimation.

TABLE II
STATISTICS OF RTL CIRCUITS USED FOR THE EXPERIMENTS

Circuit	RTL Circuit			Gate-level circuit	
	Datapath components	Controller	Max. Chain.	# Trans.	# FFs
GCD	1 Sub, 1 Lt_Comp, 2 Eq_Comp, 10 Mux, 4 Reg	5 states, 10 ctrl. signals	4	2,074	28
Barcode	2 Add, 4 Eq_Comp, 29 Mux, 8 Reg	7 states, 29 ctrl. signals	9	4,018	67
X.25	1 Add, 1 Sub, 1 Lt_Comp, 20 Mux, 10 Reg	8 states, 20 ctrl. signals	8	4,444	76
Vendor	2 Sub, 2 Lt_Comp, 25 Mux, 6 Reg	4 states, 25 ctrl. signals	9	3,446	36
Graphics	2 Add, 2 Sub, 2 Lt_Comp, 1 Eq_Comp, 50 Mux, 12 Reg	6 states, 50 ctrl. signals	7	7,742	85
Dealer	1 Add_Sub, 1 Lt_Comp, 1 $Decoder$, 2 Eq_Comp, 64 Mux, 40 Reg	14 states, 33 ctrl. signals	8	15,842	309
Poly	1 Add_Sub, 1 Mul, 11 Mux, 7 Reg	10 states, 11 ctrl. signals	4	9,554	116

apply the appropriate bit-level or word-level glitching activity model. In the case of a $CONTROL$ node, we use the (functional bit-level) signal statistics and glitching activities at its inputs, together with the timing relationships between its inputs, to estimate glitching activity at its output, as explained in Section VI.

The power estimation procedure also traverses the levelized RCG, and applies the appropriate power model to compute the power consumed in each component.

X. EXPERIMENTAL RESULTS

We performed experiments in order to evaluate the proposed RTL power estimation techniques using the following RTL circuits: the GCD circuit shown in Fig. 1, a barcode preprocessor circuit (Barcode), a circuit implementing a part of the X.25 communications protocol (X.25), a vending machine controller (Vendor), a line-drawing procedure from a graphics controller (Graphics), the dealer process in an implementation of a Blackjack card game player (Dealer), and a fourth-order polynomial (Poly). All the RTL circuits were obtained by synthesizing them from behavioral specifications using the SECONDS high-level synthesis system [44], [45]. Table II presents various statistics for the RTL circuits used in our experiments, including the number of datapath macro blocks of each type, the number of states and control signals (an indicator of the controller complexity), the maximum number of levels of chaining in the RTL circuit, and the number of transistor pairs and flip-flops (FFs) in the gate-level implementation of the complete circuit derived after logic synthesis. Each of the following is counted as a single level for computing the maximum chaining level: a 2-input functional unit, a 2-to-1 multiplexer, a comparator, and a cone of control logic. The gate-level implementations vary in complexity from about 2,000 transistors to about 16 000 transistors, and from 28 FFs to 309 FFs. Since these examples do not represent highly arithmetic-intensive computations, most bit-vector signals in the datapaths had 8 bits, and hence most datapath macro-blocks were 8-bit components. The datapaths are characterized by an abundance of multiplexers to perform conditional signal assignments, and contain significant amounts of both data and control chaining. The controllers vary from 4 to 14 states, and the number of their outputs (control signals) varies from 10 to 50.

Characterization details. The RTL circuits instantiate blocks from an in-house RTL library. We built glitching activity as well as power models for relevant library blocks. For components characterized using word-level macro-models, we constructed separate macro-models for bit-widths of 2, 4, 8, 12, 16, 24, 32, and 64. The piecewise linear models used for characterizing glitching activity and power consumption at the word-level are represented as lookup tables, with interpolation performed dynamically upon access. For our experiments, we implemented two different approaches to perform characterization for each point (entry) in the lookup table of a module. The first was to use a fixed-length characterization or training set of 10 000 pseudo-random input vectors for each entry. The second was to continue the simulation until convergence, i.e., the observed mean did not change by more than a prespecified percentage upon further simulation of a prespecified number of additional input vectors. In all cases, convergence occurred before simulating 10 000 input vectors. Hence, we chose the results of the fixed-length characterization experiment. Note that it is also possible to use more complex sampling and stopping criteria when determining each entry in the lookup tables. However, since the circuits being simulated during characterization are relatively small (just individual RTL components), we do not believe that more complex techniques are necessary in this context. This was also borne out by our characterization experiments as mentioned above. The characterization run for an RTL library containing around 300 modules required around six hours of CPU time on a SPARCstation 20 with 128 MB main memory. Most of this time was spent in file I/O, and by the gate-level power estimation tool in simulating the module netlists for the characterization patterns. The file I/O overhead can be significantly reduced by source code integration of

the various tools as opposed to the scripting language based integration that is employed in our current implementation. Running the ANOVA tests was quite efficient, and took less than five seconds for each module.

For the first phase of our power estimation tool (zero-delay RTL simulation), we used a long test bench of typical input stimuli that were derived using knowledge of the functionality of the design and its environment. As the simulation proceeded, zero-delay statistics for various signals were collected. The zero-delay statistics were then used to predict glitching activity at various datapath and control signals using the models presented in the previous sections. Thereafter, zero delay and glitch statistics were used to calculate a number for power consumption for each datapath block and for the controller. Note that, in these experiments, since the vectors that are applied to the various embedded blocks are determined by the remaining circuitry, and are not part of the characterization vectors, these results reflect the out-of-sample accuracy of the various glitching and power macro-models.

In order to evaluate the accuracy of our RTL power estimation tool, we performed logic synthesis optimizations on the RTL circuit, mapped the controller and datapath to NEC's CMOS6 technology library [35], and estimated power consumption using the tool CSIM [36]. It bears mentioning that CSIM has been calibrated with SPICE and benchmarked within 10% of SPICE. As mentioned before, it incorporates several state-of-the-art gate-level power simulation techniques, including state-dependent power modeling, accurate glitch filtering using inertial delay models, etc. The power models for individual library cells used in CSIM were constructed using SPICE. Thus, we believe that CSIM is a reasonable reference point to compare against for RTL power estimation.

We attempted to perform experiments to demonstrate the following: 1) the accuracy of our RTL power estimation tool compared to estimates obtained using CSIM; and 2) the importance of estimating glitching activity and using it to enhance the accuracy of power estimates for the various RTL circuit blocks. Table III reports the power estimates obtained using CSIM after a complete gate-level implementation of the RTL circuit (column **CSIM**), RTL power estimates obtained using our tool (column **RTL Est.**), and RTL estimates obtained by ignoring the effects of glitching activity at the inter-RTL-component control and datapath signals (column **RTL Est. W/O Inter-Comp. Gl.**). In order to obtain RTL power estimates ignoring the effects of glitches, we used our tool but instructed it not to perform the second phase of glitching activity estimation. Note that, the estimates thus obtained do not represent a zero-delay power estimate, since they do include the effects of glitching internal to each RTL component. For the second and third cases, the table reports the power estimate as well as the percentage error with respect to CSIM. The results indicate that the presented RTL switching activity and power estimation techniques result in power estimates that range from within 3.61% to within 6.71% of those obtained after the final gate-level implementation of the circuit. The results also demonstrate the importance of considering glitching activity at control and datapath signals during RTL power estimation.

TABLE III
POWER ESTIMATION RESULTS

Circuit	CSIM Pow. (mW)	RTL Est. Pow. (mW)	RTL Est. %Err	RTL Est. W/O Inter-Comp. Gl. Pow. (mW)	RTL Est. W/O Inter-Comp. Gl. %Err
GCD	1.64	1.53	6.71	1.28	21.95
Barcode	2.82	2.94	4.25	2.41	14.54
X.25	3.38	3.18	5.92	2.89	14.49
Vendor	4.71	4.54	3.61	3.96	15.92
Graphics	9.65	9.25	4.14	7.39	23.42
Dealer	8.57	8.11	5.37	7.04	17.85
Poly	7.16	6.88	3.91	6.37	11.03

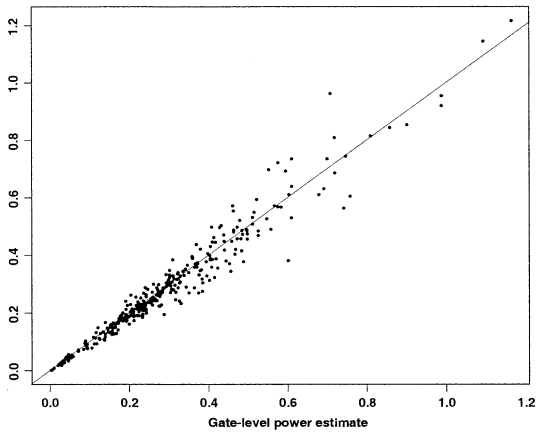


Fig. 9. Scatter plot of RTL versus gate-level power estimates for individual RTL components.

A. Instance-by-Instance Estimation Accuracy

The results presented in Table III demonstrate the accuracy of our tool in the context of estimated power consumed in the entire RTL circuit. While that in itself is a useful end application, there may be several scenarios where the designer or an optimization tool may require estimates of power consumption for individual components in the circuit. In such scenarios, the accuracy of the instance estimates are also important. In order to evaluate the accuracy of power estimates generated by our tool for individual RTL components, we have provided in Fig. 9, a scatter plot of the RTL power estimate versus the gate-level power estimate, for all components in all the seven RTL circuits considered in Tables II and III. The line in the figure indicates the ideal $y = x$ line, i.e., points that lie on this line represent components for which the RTL and gate-level power estimates exactly match. We computed the *mean absolute error* and *median absolute error* over all points in the scatter plot to be 8.96% and 6.31%, respectively. Note that since we are using *absolute* errors, positive and negative errors do not cancel out. The mean and median errors (allowing positive and negative errors to cancel out) were -2.11% and -2.03%, respectively. The maximum absolute error was 32.50%. The maximum error was caused due to very few outliers. By excluding four outliers out of the set of 340 points in the scatter plot, the maximum absolute error dropped to 18.1%.

Fig. 10 provides a similar scatter plot of the normalized switching activity estimate of each control and datapath signal in all the RTL circuits considered (the plot contains a total of

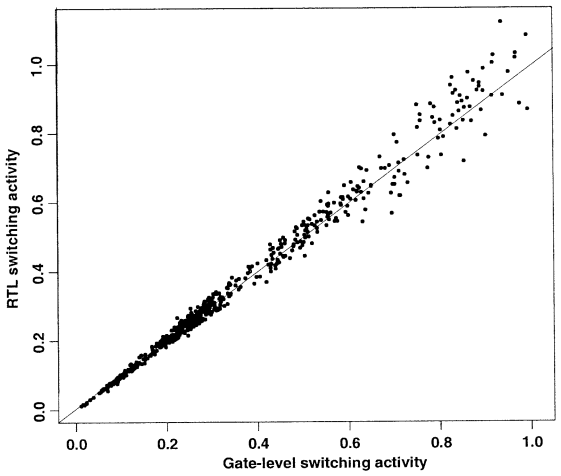


Fig. 10. Scatter plot of RTL versus gate-level switching activity estimates at individual signals.

511 points). In order to best represent the control signals as well as datapath signals in the same plot, the activity numbers for multibit signals are per-bit. Note that, for signals with higher switching activity, the activity estimated by our RTL tool is somewhat pessimistic, due to the partial delay assumptions made in the control logic analysis procedure of Section VI. The mean and median absolute errors for all points in Fig. 10 were 5.67% and 4.51%, respectively. The maximum absolute error was 22.91%.

B. In-Sample Versus Out-of-Sample Accuracy

Characterization-based macro-modeling techniques, including the ones used in this paper, are constructed using specific input sequences. For such macro models, it is important to systematically evaluate accuracy on input sequences that are dissimilar to those used for characterization (referred to as out-of-sample accuracy). As described in Sections VII and VIII, our macro-modeling procedure is based on characterizing the input signals to a macro block using parameters that represent their signal statistics (mean, standard deviation, spatial and temporal correlations, and glitching activity). These parameters can be thought of as constituting a multidimensional characterization space. A point in the characterization space corresponds to a specific assignment of values to the macro-model parameters. Macro modeling is performed by generating a uniform grid that covers this parameter space, and characterizing the power (or output switching activity) of a macro block at all points that lie on the grid. With that background, in our context, we define in-sample accuracy as the accuracy of our macro models for input sequences whose signal statistics lie at the characterization grid points, and out-of-sample accuracy as the accuracy of our macro models for input sequences whose signal statistics lie at points that are not on the characterization grid.

Table IV summarizes the results of testing the power macro models using pseudo-randomly generated in-sample and out-of-sample sequences. To generate the in-sample sequences, we randomly chose 100 points from the characterization grid. For each point, we generated a test sequence of 10 000 vectors that had signal statistics corresponding to the values given by the chosen point. We initialized our test sequence generator with different seeds to ensure that the sequences generated had similar statistical properties as, but were not identical to, the sequences used for characterization. To generate out-of-sample sequences, a similar procedure was used, except that we started with points that did not lie on the characterization grid. For comparison, we also include the accuracy of the power macro- models when they are used in the context of the RTL circuits described in Tables II and III. Table V presents similar results to compare the in-sample and out-of-sample accuracy of our switching activity estimation techniques. In both tables, the error represents the difference between the average power or switching activity reported by the gate-level power estimator and the proposed macro models.

Fig. 11 presents a histogram of error distributions for the performance of our model on out-of-sample input traces. The figure indicates that most of the out-of-sample test cases resulted in errors within $[-6\%, +4\%]$.

C. Computational Efficiency

The CPU times required for gate-level power simulation varied from 23 s (for an input trace of 5280 clock cycles for example circuit `GCD`) to 273 s (15 300 clock cycles for example circuit `Dealer`). The CPU times required for RTL power estimation were all under five seconds. All experiments were performed on a SPARCstation 20 with 128 MB main memory. The experiments indicated a speedup of $10\times$ to $50\times$ for our RTL power estimator compared to CSIM for the example circuits shown in Table III. It is important to note in this context that the time required to obtain power estimates through lower level (e.g., gate-level) estimation tools depends on two factors: 1) the time taken to synthesize a complete gate-level netlist from the RTL circuit and 2) the time required to run the gate-level power estimation tool. Our comparison of CPU times only considers the second component for gate-level power estimation. The time required to synthesize a gate-level netlist can vary significantly depending on the optimization effort used during synthesis (e.g., fast synthesis versus synthesis with full optimization). Thus, it is clearly difficult to present a single number to represent the synthesis time for a given circuit. For the logic synthesis tool used in our experiments, the time taken to run a fast synthesis script (RTL HDL compilation, simple macro-block expansion and one-to-one technology mapping of generic gates) varied from 77 to 571 s. The time required to run a typical optimizing logic synthesis script (that also included technology independent delay optimization with area recovery) varied from 185 to 1181 s. Clearly, even fast synthesis followed by gate-level power simulation is infeasible for use in exploring RTL design tradeoffs that involve comparison of a large number of candidate circuits, or variants of the same design. Another scenario where more efficient power estimation is needed is high-level synthesis for low power, where hundreds of different RTL implementations may be compared for a given behavioral specification [46]–[49]. In such situations, clearly there is a

TABLE IV
IN-SAMPLE AND OUT-OF-SAMPLE ACCURACY OF THE PROPOSED POWER MACRO-MODELING TECHNIQUES

Test seq.	Error Range [Min,Max] (%)	Mean Abs. Error (%)	Median Abs. Error (%)
In-sample	[-4.31,+5.07]	1.24	1.37
Out-of-sample	[-15.61,+12.43]	3.44	2.79
Embedded in RTL ckt.	[-32.50,+28.41]	8.96	6.31

TABLE V
IN-SAMPLE AND OUT-OF-SAMPLE ACCURACY OF THE PROPOSED SWITCHING ACTIVITY MACRO-MODELING TECHNIQUES

Test seq.	Error Range [Min,Max] (%)	Mean Abs. Error (%)	Median Abs. Error (%)
In-sample	[-9.61,+9.93]	4.52	4.14
Out-of-sample	[-17.84,+19.42]	6.14	5.33
Embedded in RTL ckt.	[-19.23,+22.91]	5.67	4.51

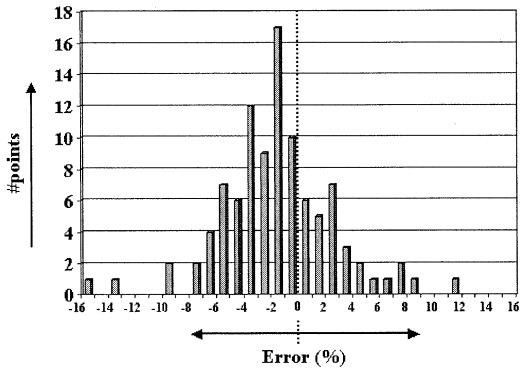


Fig. 11. Error histogram for the proposed macro-models under out-of-sample test sequences.

need for faster power estimation, which can only be provided by RTL power estimation techniques. As can be seen from the above numbers, our RTL power estimator offers one to two orders of magnitude improvement in total power estimation time compared to synthesizing a gate-level netlist and running a gate-level power estimator.

XI. CONCLUSIONS

We presented techniques for switching activity analysis and power estimation at the register transfer level. The significant features of our techniques are the following: 1) we consider the generation and propagation of glitches through the controller and datapath while performing power estimation; 2) we combine the use of word-level modeling for datapath blocks that associate a word-level value to their operands, and accurate bit-level modeling for other datapath blocks; and 3) we demonstrate that our techniques are well suited for estimating power consumption in control-flow intensive designs that have significantly different power-consumption characteristics from dataflow intensive designs. We believe that our power estimation techniques will be useful in assisting a designer or a high-level synthesis tool to evaluate the impact of various design decisions on switching activity and power consumption.

The techniques presented in this paper could be extended along several directions. The word-level macro-modeling techniques we employ perform quite well even when the "power function" is nonlinear, (since piece-wise linear models can be used to approximate arbitrary functions, provided the characterization granularity is sufficiently small). Nevertheless, it may be possible to further improve the accuracy of such macro models or reduce the number of gate-level simulation runs needed for characterization, by using more sophisticated interpolation techniques. As mentioned in Section VII, it may be possible to construct more efficient or more accurate macro models by including the output signal statistics as parameters. In general, in the case of more complex predesigned RTL components, it may also be necessary to include statistics of signals internal to the macro block as parameters in the macro model. The control expression based switching activity analysis techniques presented in our work have been shown to work very well for the control logic in circuits generated by high-level synthesis tools. For arbitrary random logic (e.g., flat gate-level descriptions of subcircuits), it may be possible to draw upon gate-level activity estimation techniques to improve the accuracy of activity and power analysis. Finally, for deep submicron technologies, accurate RTL power estimation requires an estimate of power consumed in the interconnect. This can be achieved by coupling our activity estimation techniques with design planning technologies that can estimate interconnect capacitance.

REFERENCES

[1] C. Ramachandran, F. J. Kurdahi, D. D. Gajski, A. C. H. Wu, and V. Chaiyakul, "Accurate layout area and delay modeling for system level design," in *Proc. Int. Conf. Computer-Aided Design*, Oct. 1992, pp. 355–361.

[2] A. Kuehlmann and R. Bergamaschi, "Timing analysis in high-level synthesis," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 349–354.

[3] P. K. Jha and N. D. Dutt, "Rapid estimation for parameterized components in high-level synthesis," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 296–303, Sept. 1993.

[4] S. Bhattacharya, S. Dey, and F. Brglez, "Provably correct high-level timing analysis without path sensitization," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 736–742.

[5] A. C. Deng, "Power analysis for CMOS/B1CMOS circuits," in *Proc. Int. Workshop Low-Power Design*, Apr. 1994, pp. 3–8.

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

[7] J. Monteiro and S. Devadas, *Computer-Aided Design Techniques for Low Power Sequential Logic Circuits*. Norwell, MA: Kluwer, 1996.
[8] M. Pedram, "Power minimization in IC design: principles and applications," *ACM Trans. Design Automation Electronic Systems*, vol. 1, pp. 3–56, Jan. 1996.
[9] E. Macii, M. Pedram, and F. Somenzi, "High level power modeling, estimation, and optimization," in *Proc. Design Automation Conf.*, June 1997, pp. 504–511.
[10] A. R. Chandrakasan and R. W. Brodersen, *Low-Power Digital CMOS Design*. Norwell, MA: Kluwer, 1995.
[11] J. Frenkil, "Tools and methodologies for low power design," in *Proc. Design Automation Conf.*, June 1997, pp. 76–81.
[12] W. Nebel, J. Sproch, and S. Malik, "Power analysis and optimization: spanning the levels of abstraction," in *Tutorial Notes, Int. Symp. Low-Power Electronics and Design*, Aug. 1997.
[13] K. D. Müller-Glaser, K. Kirsch, and K. Neusinger, "Estimating essential design characteristics to support project planning for ASIC design management," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 148–151.
[14] D. Liu and C. Svensson, "Power consumption estimation in CMOS VLSI chips," *IEEE J. Solid-State Circuits*, vol. 29, pp. 663–670, June 1994.
[15] D. Marculescu, R. Marculescu, and M. Pedram, "Information theoretic measures for energy consumption at the register-transfer level," in *Proc. Int. Symp. Low-Power Design*, Apr. 1995, pp. 81–86.
[16] F. N. Najm, "Toward a high-level power estimation capability," in *Proc. Int. Symp. Low-Power Design*, Apr. 1995, pp. 87–92.
[17] M. Nemani and F. N. Najm, "High level power estimation and the area complexity of Boolean functions," in *Proc. Int. Symp. Low-Power Electronics and Design*, Aug. 1996, pp. 329–334.
[18] C.-H. Chen and C.-Y. Tsui, "Toward the capability of providing power-area-delay tradeoff at the register transfer level," in *Proc. Int. Symp. Low-Power Electronics and Design*, Aug. 1998, pp. 24–29.
[19] M. Nemani and F. Najm, "High-level area and power estimation for VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 697–713, June 1999.
[20] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin, *High-level Synthesis: Introduction to Chip and System Design*. Norwell, MA: Kluwer, 1992.
[21] S. R. Powell and P. M. Chau, "Estimating power dissipation of VLSI signal processing chips: the PFA technique," in *Proc. VLSI Signal Processing IV*, Sept. 1990, pp. 250–259.
[22] P. Landman and J. M. Rabaey, "Architectural power analysis: the dual bit type method," *IEEE Trans. VLSI Syst.*, vol. 3, pp. 173–187, June 1995.
[23] P. F. Landman and J. M. Rabaey, "Black-box capacitance models for architectural power analysis," in *Proc. Int. Workshop Low-Power Design*, Apr. 1994, pp. 165–170.
[24] P. Landman and J. M. Rabaey, "Activity-sensitive architectural power analysis for the control path," in *Proc. Int. Symp. Low-Power Design*, Apr. 1995, pp. 93–98.
[25] T. Sato, Y. Ootaguro, M. Nagamatsu, and H. Tago, "Evaluation of architecture-level power estimation for CMOS RISC processors," in *Proc. Symp. Low-Power Electronics*, Oct. 1995, pp. 44–45.
[26] H. Mehta, R. M. Owens, and M. J. Irwin, "Energy characterization based on clustering," in *Proc. Design Automation Conf.*, June 1996, pp. 702–707.
[27] L. Benini, A. Bogliolo, M. Favalli, and C. De Micheli, "Regression models for behavioral power estimation," in *Proc. Int. Workshop Power and Timing Modeling, Optimization, and Simulation*, 1996.
[28] A. Raghunathan, S. Dey, and N. K. Jha, "Register-transfer level estimation techniques for switching activity and power consumption," in *Proc. Int. Conf Computer-Aided Design*, Nov. 1996, pp. 158–165.
[29] C. T. Hsieh, Q. Wu, C. S. Ding, and M. Pedram, "Statistical sampling and regression analysis for RT-level power evaluation," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1996, pp. 583–588.
[30] S. Gupta and F. N. Najm, "Power macromodeling for high level power estimation," in *Proc. Design Automation Conf.*, June 1997, pp. 365–370.
[31] Z. Chen and K. Roy, "A power macromodeling technique based on power sensitivity," in *Proc. Design Automation Conf.*, June 1998, pp. 678–683.
[32] M. Barocci, L. Benini, A. Bogliolo, B. Ricco, and G. De Micheli, "Lookup table power macro-models for behavioral library components," in *Proc. IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, Mar. 1999, pp. 173–181.
[33] R. P. Llopis and F. Goossens, "The Petrol approach to high-level power estimation," in *Proc. Int. Symp. Low-Power Electronics and Design*, Aug. 1998, pp. 130–132.
[34] D. I. Cheng, K.-T. Cheng, D. C. Wang, and M. Marek-Sadowska, "A new hybrid methodology for power estimation," in *Proc. Design Automation Conf.*, June 1996, pp. 439–444.
[35] *CMOS6 Library Manual*: NEC Electronics, Inc., Dec. 1992.
[36] *CSIM Version 5 Users Manual*: Systems LSI Division, NEC Corp., 1993.
[37] F. N. Najm and M. Y. Zhang, "Extreme delay sensitivity and the worst-case switching activity in VLSI circuits," in *Proc. Design Automation Conf.*, June 1995, pp. 623–627.
[38] F. N. Najm, "Transition density, a stochastic measure of activity in digital circuits," in *Proc. Design Automation Conf.*, June 1991, pp. 642–649.
[39] G. Casella and R. L. Berger, *Statistical lnference*. Belmont, CA: Duxbury, 1990.
[40] S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists*. New York: Wiley, 1987.
[41] *StatPlan IV Program Manual*: The Futures Group, 1989.
[42] M. R. Carey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
[43] D. E. Knuth, *The Art of Computer Programming, Vol 2: Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1980.
[44] S. Bhattacharya, S. Dey, and F. Brglez, "Performance analysis and optimization of schedules for conditional and loop-intensive specifications," in *Proc. Design Automation Conf.*, June 1994, pp. 491–496.
[45] ——, "Clock period optimization during resource sharing and assignment," in *Proc. Design Automation Conf.*, June 1994, pp. 195–200.
[46] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 12–31, Jan. 1995.
[47] A. Raghunathan and N. K. Jha, "SCALP: an iterative-improvement-based low-power datapath synthesis system," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 1260–1277, Nov. 1997.
[48] J. M. Chang and M. Pedram, "Register allocation and binding far low power," in *Proc. Design Automation Conf.*, June 1995, pp. 29–35.
[49] K. S. Khouri, C. Lakshminarayana, and N. K. Jha, "High-level synthesis of low-power control-flow intensive circuits," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 1715–1729, Dec. 1999.

Anand Raghunathan (S'93–M'97–SM'00) received the B.Tech. degree in electrical and electronics engineering from the Indian Institute of Technology, Madras, India, in 1992, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, in 1994 and 1997, respectively.

He is currently a Senior Research Staff Member at NEC Laboratories America, Princeton, NJ, where he leads several projects related to the research and development of system-on-chip architectures, design methodologies, and design tools with emphasis on high-performance, low power, and testable designs. He has coauthored *High-Level Power Analysis and Optimization* (Norwell, MA: Kluwer, 1998), and a book chapter in *System-Level Power Optimization For Wireless Multimedia Communications* (Norwell, MA: Kluwer, 2002). He has presented full-day and embedded conference tutorials on low-power design, wireless communication system design, and considering testability during high-level design. He holds or has filed for 14 U.S. patents in the areas of advanced system-on-chip architectures, design methodologies, and VLSI CAD.

Dr. Raghunathan has served as a Member of the technical program committees of several IEEE and ACM conferences. He is a Member of the organizing committee of the VLSI Test Symposium (1998–2003), where he is currently the Program Chair. He has served as Associate Editor of the IEEE TRANSACTIONS ON VLSI SYSTEMS, and as a Member of the Editorial Board of IEEE Design & Test of Computers. He is currently Vice-Chair of the Tutorials & Education Group at the IEEE Computer Society's Test Technology Technical Council. He received Best Paper Awards at the IEEE International Conference on VLSI Design (1998 and 2003) and at the ACM/IEEE Design Automation Conference (1999 and 2000), and two Best Paper Award nominations at the ACM/IEEE Design Automation Conference (1996 and 1997). He received the Patent of the Year Award (an award recognizing the invention that has achieved the highest impact) from NEC in 2001. He was the recipient of an IEEE Meritorious Service Award, and was elected as a Golden Core Member of the IEEE Computer Society in 2001.

Sujit Dey received the Ph.D. degree in computer science from Duke University, Durham, NC, in 1991.

He is currently a Professor in the Department of Electrical and Computer Engineering, University of California, San Diego (UCSD), where his research group is developing configurable platforms, consisting of adaptive wireless protocols and algorithms, and deep submicron adaptive system-on-chips, for next-generation wireless appliances as well as network infrastructure devices. He is affiliated with the California Institute of Telecommunications and Information Technology, the UCSD Center for Wireless Communications, and the DARPA/MARCO Gigascale Silicon Research Center. Prior to joining UCSD in 1997, he was a Senior Research Staff Member at the NEC Computer and Communications Research Laboratories (CCRL), Princeton, NJ. He has coauthored more than 100 publications, including journal and conference papers, a book on low-power design, and several book chapters. He is a coinventor of 9 U.S. patents, and has two other patents pending. He has presented numerous tutorials and invited talks, and participated in panels, in the topics of low-power wireless systems design, hardware–software embedded systems, and deep-submicron system-on-chip design and test.

Dr. Dey has been the General Chair, Program Chair, and Member of organizing and program committees of several IEEE conferences and workshops. He received Best Paper awards at the Design Automation Conferences in 1994, 1999, and 2000, and the 11th VLSI Design Conference in 1998, and several best paper nominations.

Niraj K. Jha (S'85–M'85–SM'93–F'98) received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1981, the M.S. degree in electrical engineering from the State University of New York (SUNY) at Stony Brook in 1982, and the Ph.D. degree in electrical engineering from the University of Illinois, Urbana, IL, in 1985.

Currently, he is a Professor in the Department of Electrical Engineering at Princeton University, Princeton, NJ. He is coauthor of two books, *Testing and Reliable Design of CMOS Circuits* (Norwell, MA: Kluwer, 1990) and *High-Level Power Analysis and Optimization* (Norwell, MA: Kluwer, 1998). He has authored or coauthored more than 200 technical papers. He has coauthored four papers which have won the Best Paper Award at ICCD'93, FTCS'97, ICVLSID'98, and DAC'99. His research interests include low-power hardware and software design, computer-aided design of integrated circuits, digital system testing, and distributed computing. He is currently an Editor for the *Journal of Electronic Testing: Theory, and Applications* (JETTA). He has served as the Guest Editor for the JETTA special issue on high-level test synthesis. He has also served as the Program Chairman of the 1992 Workshop on Fault-Tolerant Parallel and Distributed Systems. He is the Director of the Center for Embedded system-on-a-chip Design funded by New Jersey Commission on Science and Technology.

Dr. Jha has served as an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: ANALOG AND DIGITAL SIGNAL PROCESSING and of IEEE TRANSACTIONS ON VLSI SYSTEMS. He is currently serving as an Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN. He is the recipient of the AT&T Foundation Award and the NEC Preceptorship Award for research excellence.