Register-Transfer Level Estimation Techniques for Switching Activity and Power Consumption

Anand Raghunathan *
Dept. of EE, Princeton Univ.
Princeton, NJ 08544

Sujit Dey
C&CRL, NEC USA
Princeton, NJ 08540

Niraj K. Jha †
Dept. of EE, Princeton Univ.
Princeton, NJ 08544

Abstract

We present techniques for estimating switching activity and power consumption in register-transfer level (RTL) circuits. Previous work on this topic has ignored the presence of glitching activity at various data path and control signals, which can lead to significant underestimation of switching activity. For data path 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 data 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*. 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. Experiments on example RTL designs resulted in power estimates that were within 7% of those produced by an in-house power analysis tool on the final gate-level implementation.

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, 2, 3, 4]. In this paper, we focus on the problem of estimating power consumption from RTL descriptions.

Designs at the architecture or RT level are characterized by the instantiation of pre-designed macro blocks including arithmetic operators, multiplexers, registers, vector logic operators, *etc*. In order to avoid the increase in computational complexity introduced by expanding these blocks to lower-level descriptions, it is necessary to develop power models for these library blocks. The development of such "black-box" models is an important part of high-level power estimation, and is typically a one-time cost incurred during library development. Another important task to be performed in RTL power estimation is the estimation of switching activity and signal statistics at various signals in the RTL circuit, that are then fed into the power models for each block to estimate power consumption.

* Supported by NEC C&C Research Labs
† Supported by NSF under Grant No. MIP-9319269.

One of the early architecture level power estimation techniques, called the power factor approximation (PFA) method [5], characterized the power consumption in architectural blocks by simulating their implementations using random input sequences. The inability of the PFA technique to account for the dependency of power consumption in embedded modules on their input statistics was addressed in [6] using a dual bit-type (DBT) model for word-level signals. Activity-sensitive capacitance models were developed for various library components, and coupled with zero-delay activity information derived from RTL simulation to estimate power consumption. As pointed out in [6], the DBT model is most applicable to data-flow intensive designs, since it assumes that each multi-bit signal can be associated with a word-level "value", whose probability distribution satisfies certain assumptions. In [7, 8], the use of computational entropy and informational energy as measures of switching activity was proposed. An activity-based control model to estimate controller power consumption was presented in [9].

Most previous work on RT level power estimation has not focused on control-flow intensive designs, which have significantly different power consumption characteristics. Unlike data-flow intensive designs, non-arithmetic components like multiplexers, registers, vector logic operators, and the controller dominate the total power consumption. Due to the complex control flow, the controller has a significant impact on the power consumption of the circuit, necessitating accurate activity analysis techniques for control signals. In addition, control-flow intensive designs often contain multi-bit signals that do not collectively have any meaning as a "number" and hence cannot be modeled using techniques such as the DBT model. Previous work has also ignored the presence of glitching activity at various signals in the RTL circuit, and its effect on power consumption, which can lead to significant errors in the power estimates as shown in the following section.

II. Motivation

We illustrate some of the issues involved in RTL power estimation through the analysis of an example RTL circuit shown in Figure 1, that computes the greatest common divisor (`GCD`) of two numbers. The RTL blocks used in the `GCD` data path are one subtractor, three (one less-than ($<$) and two equal-to ($=$)) comparators, registers, and multiplexers. The controller is sub-divided into the state register, the next state logic, and the decode logic that generates the control signals for the data path. The control expressions implemented by the decode logic are also shown in Figure 1 (control signal $contr[i]$ feeds the select input of the data path multiplexer marked $[i]$). The control expressions are logic expressions involving the decoded state variables $x_0 \ldots x_4$ (x_i is 1 when the controller is in state s_i), and status signals generated by the data path like the outputs of comparators.

The `GCD` RTL circuit was mapped to a commercial technology library, and an in-house simulation-based gate-level power estimation tool, CSIM [10], was used to monitor the switching activity including and excluding glitches at selected data path and control

ICCAD '96
1063-6757/96 $5.00 (c) 1996 IEEE

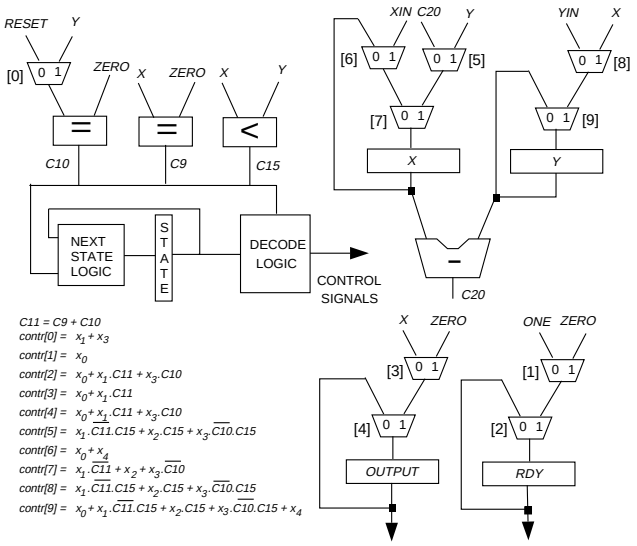


Figure 1: The GCD RTL circuit

Table 1: 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			

signals [1]. The results are reported in Table 1. 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.

Known 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. 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 $1.64mW$. Next, we performed a zero-delay RTL simulation, 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 signal traces derived in the previous step, and the individual power estimates were added up to yield a power estimate of $1.32mW$ for the entire circuit (an under-estimate of 19.5%).

Glitches generated at the control signals (outputs of the controller) may themselves have a significant impact on the data path power consumption. In order to study this effect, we performed another experiment with the GCD RTL circuit, where we collected zero-delay traces at the controller/data path interface, and used these traces to estimate power consumption for implementations of the controller and data path separately. Thus, the effects of glitches at (i) the data path outputs (status signals) that feed the controller, and (ii) the controller outputs (control signals) that feed the data path were ignored for this experiment, while glitch proliferation within each sub-circuit was considered. The sum of the individual power estimates for the controller and data path was $1.45mW$, 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 data path. 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 $1.53mW$, that corresponds to an error of 6.7% compared to running CSIM on the entire gate-level circuit.

III. RTL power estimation methodology

The flowchart shown in Figure 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 signal statistics at various signals in the RTL circuit. We use RTL cycle-based simulation for this purpose. The test bench can either be provided by the designer based on an understanding of the design's functionality and interface, or can be automatically generated in order to conform to given primary input statistics. During the simulation run, we monitor the various signals in the RTL circuit to collect a variety of word- and bit-level statistics. The advantages of using simulation for this purpose are its flexibility (*e.g.* it is relatively easy to use for mixed-level descriptions), and the high speed of RTL cycle-based simulation.

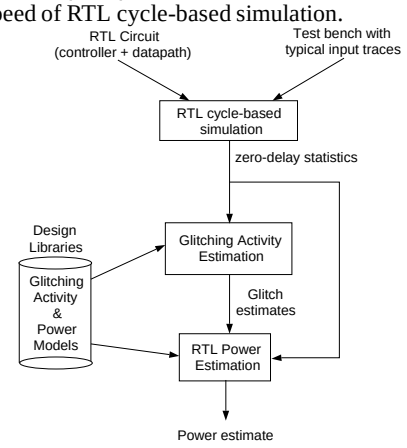


Figure 2: Overview of our RTL power estimation tool flow

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 (Section IV). 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, that are assumed to be glitch-free, and "propagates" glitching activity information forward through RTL circuit blocks until we reach primary outputs/register inputs.

In the third phase, the zero-delay signal statistics and glitching activity estimates are used to calculate power consumption in each RTL module. For this purpose, we have developed power models for various RTL blocks like functional units, comparators, multiplexers, and registers (Section V). These power models capture 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 pre-

[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.

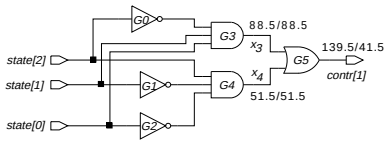


Figure 3: Implementation of control signal $contr[1]$ in the `barcode` RTL circuit

sented in [6] in 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, vector logic operations, *etc*).

IV. Estimating glitching activity at the RT level

In this section, we present our "models" for glitch generation in and propagation through various components of the RTL circuit.

A. Glitching activity at the control signals

The controller's inputs are the status signals from the data path (typically outputs of comparators or combinations thereof), while its outputs are the control signals that feed the data path. The control logic is usually represented as *control expressions* during the high-level synthesis process. These control expressions are expressed in the form

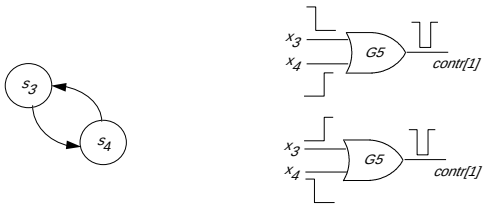
$$contr = \sum_i x_i . \left(\prod_j Cj \right) \quad (1)$$

where x_i represents a decoded controller state variable (corresponding to controller state s_i), Cj represents a status signal, which is typically the output (or inverted output) of a comparator from the data path, and $\sum$ and $\prod$ represent the Boolean OR and AND operations, respectively. 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. The status signals (Cj) may themselves carry glitches, that propagate through the control logic, causing the control signals to be glitchy. On the other hand, the control logic can itself generate a significant amount of glitching activity.

Accurate estimation of glitch generation and propagation in the control logic requires detailed information regarding the structure of the controller implementation and delays. However, the final implementation of the controller is typically *not available* during high-level design iterations. Hence, we estimate glitching activity at control signals using their control expressions.

Estimating Glitch Generation from Control Expressions. Glitch generation in the control logic is a result of the interaction of certain logic and temporal conditions, as illustrated by the following example.

Example 1: Let us consider a portion of an RTL circuit that is a pre-processor for a `barcode` reader. We focus on a particular control signal, $contr[1]$, whose implementation is given in Figure 3. Signals $state[2]$, $state[1]$ and $state[0]$ are fed by the flip-flops of the controller. Signals x_3, x_4 and control signal $contr[1]$ are annotated with their transition counts including and excluding glitches, indicating glitch generation at gate $G5$. Consider the partial state transition graph for the controller that is shown in Figure 4(a). The figure indicates a loop involving states s_3 and s_4, that is executed a large number of times. Figure 4(b) shows how the inputs and output of gate $G5$ vary under these two state transitions. A transition from s_3 to s_4 causes a rising transition on x_4 and a falling transition on x_3. However the rising transition on x_4 arrives later than the falling transition on x_3, due to the delays of inverters $G1$ and $G2$, resulting in a $1-0-1$ static hazard or glitch at the output of gate $G5$. A similar explanation holds for the controller state transition from s_4 to s_3. The generation of glitches at $G5$ can be attributed to the following two conditions:



(a) (b)

Figure 4: (a) Partial STG for `barcode` controller, and (b) Generation of glitches at gate $G5$

- Logic: correlation between (simultaneous occurrence of) rising and falling transitions at the inputs of $G5$.
- Temporal: the controlling [2] to non-controlling transition at the input of $G5$ arrives earlier. ■

In general, the logic conditions necessary for glitch generation at a gate 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 logic value.

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 (Equation (1)), during the zero-delay RTL simulation, we maintain a distinct *glitch counter* for each product term, and also for the OR expression combining the product terms. In each cycle, we check the previous and current values at the variables involved in an AND or OR expression to see whether the logic conditions for glitch generation are satisfied. If they are, we increment the corresponding glitch counter to indicate the possibility of glitch generation in the current simulation cycle.

As mentioned earlier in this section, checking whether the temporal conditions for glitch generation are satisfied in an accurate manner requires the final implementation for the control logic, which is typically not available when performing high-level design optimizations. One possible approach 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. However, such an assumption often leads to substantial over-estimates of glitching activity, as shown in the following example.

Example 2: We would like to estimate the glitching activity at control signal $contr[2]$ in the `GCD` RTL circuit of Figure 1. The control expression for $contr[2]$ is $x_0 + x_1.C11 + x_3.C10$. In this case, the signals $C10$ and $C11$ were found to be glitch-free, simplifying the problem to that of estimating glitch generation at $contr[2]$.

Clearly, the first product term (x_0) cannot generate any glitches. From the simulation traces, we counted the number of times the logic conditions for glitch generation were satisfied for the second and third product terms.

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

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

In the above equations, the symbols $\uparrow$ and $\downarrow$ denote the rising and falling transitions, respectively. From the above numbers, one could conclude that the glitching activity generated due to the second and third product terms is 35 and 65, respectively. [3] The glitches generated due to each product term propagate to the output un-mitigated, since the decoded state variables are mutually exclusive. From the given traces, it was observed that the logic conditions for glitch generation at an OR gate were never satisfied by the outputs of the product terms. Hence, the glitching activity at control signal $contr[2]$ was estimated to be 100 transitions over the entire simulation period. A comparison with the glitching activity observed by CSIM for the same input traces and reported in Table 1 ($72 - 20 = 52$) shows that the glitching activity at $contr[2]$ was over-estimated by as much as 92.3%! ∎

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. Inputs to the control logic are divided into three groups - *early* arriving signals, *late* arriving signals, and signals whose arrival time information is assumed to be *unknown*. We assume that each controller input that is marked as late-arriving arrives significantly later than any input signal that is marked as early-arriving. No assumption is made involving the arrival time of a signal marked unknown. Similarly, no assumption is made about the relationship between the arrival times of two signals that are both marked either early or late. When the temporal conditions for glitch generation at a gate involve signals whose arrival time relationship is unknown, we revert to the pessimistic approach of only checking logic conditions.

Example 3: Let us revisit the control signal $contr[2]$ in the GCD circuit that was used for the discussions in Example 2. 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. Similarly, it can be seen that $Case\ 3$ does not satisfy the temporal conditions for glitch generation. The revised glitching activity estimate for $contr[2]$ is 50, which represents an error of only 4% with respect to the number reported by CSIM. ∎

Glitch propagation through the control logic. Consider again the generic control expression given in Equation (1). Consider a particular comparator output, $C1$, that we have predicted to be glitchy based on our data path glitching activity models. Let us re-write the control expression by separating the product terms into terms in which $C1$ appears, terms in which $\bar{C1}$ appears, and terms that do not depend on $C1$.

$$\begin{aligned} contr &= C1.contr_{C1} + \bar{C1}.contr_{\bar{C1}} \\ &+ \sum_{product\ terms\ indep.\ of\ C1} x_i . \prod_j Cj \end{aligned} \quad (2)$$

$$contr_{C1} = \sum_{product\ terms\ with\ C1} x_i \prod_{Cj \neq C1} Cj$$

$$contr_{\bar{C1}} = \sum_{product\ terms\ with\ \bar{C1}} x_i \prod_{Cj \neq \bar{C1}} Cj$$

[3] Note that each time the conditions for glitch generation at a gate are satisfied, the output undergoes two transitions. However, for compatibility with CSIM, which counts each $0 \rightarrow 1$ and $1 \rightarrow 0$ transition as half a transition, we do not multiply by the factor of two.

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 non-controlling 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.

$$\begin{aligned} Gl(C1) \quad * \quad & P\{(contr_{C1} = 1\ OR\ contr_{\bar{C1}} = 1)\ AND \\ & \sum_{product\ terms\ indep.\ of\ C1} x_i \prod_j Cj = 0\} \end{aligned} \quad (3)$$

In the above equation, $Gl(C1)$ represents the glitching activity at $C1$, and is multiplied by 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. Equation (3) assumes that the conditions for glitch generation at $C1$ and the conditions for the glitches propagating through the control logic are uncorrelated, which can lead to errors in the estimated activities. We resolve this problem by maintaining separate data statistics for each state, and for each signal in the transitive fanin of $C1$, and hence compute separate glitching activity estimates for each state [11].

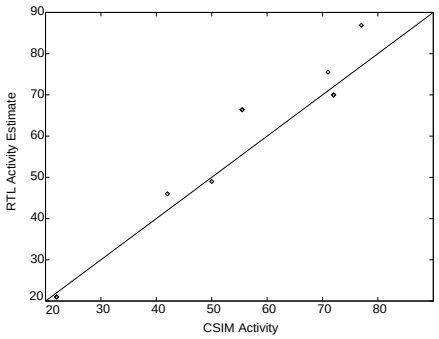


Figure 5: Scatter plot of Switching Activity at Control Signals: RTL Estimate v/s CSIM

In order to get a feel for the accuracy of our switching activity estimation techniques for control signals, we obtained a complete gate-level implementation of the GCD circuit and estimated switching activities using CSIM. The scatter plot shown in Figure 5 plots the switching activity estimated using our RTL estimation techniques (y-axis) *vs.* the switching activity reported by CSIM (x-axis), for each distinct control signal. As a reference, the plot also shows a solid line for the equation $y = x$. The figure indicates that our 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 data path.

B. Modeling glitch generation and propagation in data path blocks

For data path blocks which operate on multi-bit input signals that are associated with a word-level value (*e.g.* adders, subtractors, multipliers, and various comparators), previous work [6] has shown that it is possible to construct activity-sensitive power models that utilize word-level statistics (mean, standard deviation, *etc.*). Several data path blocks, however, do not associate any word-level value to their multi-bit 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. This allows us to build more accurate glitching activity models for such blocks, and to consider the effects of bit-level statistics that may not be well reflected by word-level signal statistics in certain situations. Since multiplexers play an important role in glitch generation and propagation for control-flow intensive designs, we explain glitching activity model for multiplexers in detail next.

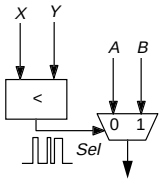
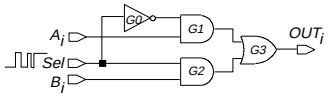


Figure 6: Circuit used to study glitch propagation from a multiplexer's select input



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

Figure 7: Modeling propagation of glitches from a multiplexer select signal

Glitch generation and propagation in multiplexers. 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 equations.

$$Gl(OUT) = \sum_{i=1}^{n} Gl\_Gen(i) + Gl\_Prop\_A_i + Gl\_Prop\_B_i + Gl\_Prop\_Sel \quad (4)$$

The term $Gl\_Gen(i)$ in the above equation represents the generation of glitches in the ith bit-slice of the multiplexer, and is calculated as follows. 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, we 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 for each vector pair. The lookup table can be thought of as a six-dimensional array $Mux\_gl\_gen[]$, and the entry of the table corresponding to present and previous input values $A_i(t), B_i(t), Sel(t), A_i(t-1), B_i(t-1), Sel(t-1)$ is written as $Mux\_gl\_gen[A_i(t), B_i(t), Sel(t), A_i(t-1), B_i(t-1), Sel(t-1)]$. During the zero-delay RTL simulation phase, we accrue the glitch generation at each bit-slice of a multiplexer by looking up the appropriate entry of the $Mux\_gl\_gen[]$ table.

The output of a multiplexer can also be glitchy due to the propagation of glitches from the data and select inputs. The terms $Gl\_Prop\_A_i$ and $Gl\_Prop\_B_i$ in Equation (4) represent the glitch propagation from A_i and B_i, respectively, and are computed using the following equations.

$$Gl\_Prop\_A_i = Gl(A_i) * P(Sel = 0)$$
$$Gl\_Prop\_B_i = Gl(B_i) * P(Sel = 1)$$

In order to study the propagation of glitches from the select signal of a multiplexer, we implemented the circuit shown in Figure 6, where the embedded multiplexer's select input is glitchy while its data inputs are not. Figure 7 shows a bit-slice of the multiplexer, and a table that reports the activity at OUT_i for all possible values of A_i and B_i. In the $< 0, 0 >$ case, glitches on select signal Sel are killed at AND gates $G1$ and $G2$ due to controlling side inputs that arrive early. When data inputs are $< 0, 1 >$ ($< 1, 0 >$), glitches on Sel propagate through gates $G2$ and $G3$ ($G1$ and $G3$). Finally, when data inputs are $< 1, 1 >$, 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 $G2$. 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 equation.

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

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 Figure 6.

Word-level models for glitching activity. We next focus on data path blocks that operate on multi-bit input signals that are associated with a word-level value (*e.g.* adders, subtractors, multipliers, and various comparators). The glitching activity at the output of an embedded data path 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. We construct glitch models from the implementations of various library components through a process of characterization. We perform controlled experiments (simulation runs) by selectively varying one or more of the controllable variables (input zero-delay statistics and glitching activities), and observing the value of the dependent variable (glitching activity at the block output). Given the set of samples obtained from the characterization experiments, there are several statistical techniques that can be used to build a model that predicts the output glitching activity [12]. An approach that we have found to be 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 subtractor with inputs named A and B, and output OUT. In general, the glitching activity at OUT can be written as

$$Gl_{OUT} = f_{gl}(Mean_A, Mean_B, SD_A, SD_B, TC_A, TC_B, SC_{A,B}, Gl_A, Gl_B) \quad (5)$$

The first seven parameters of $f()$ 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, $SC_{A,B}$ is the spatial correlation coefficient of A and B. 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 subtractor to observe the glitching activity at the subtractor's output for 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 subtractor, $n = 9$, 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.

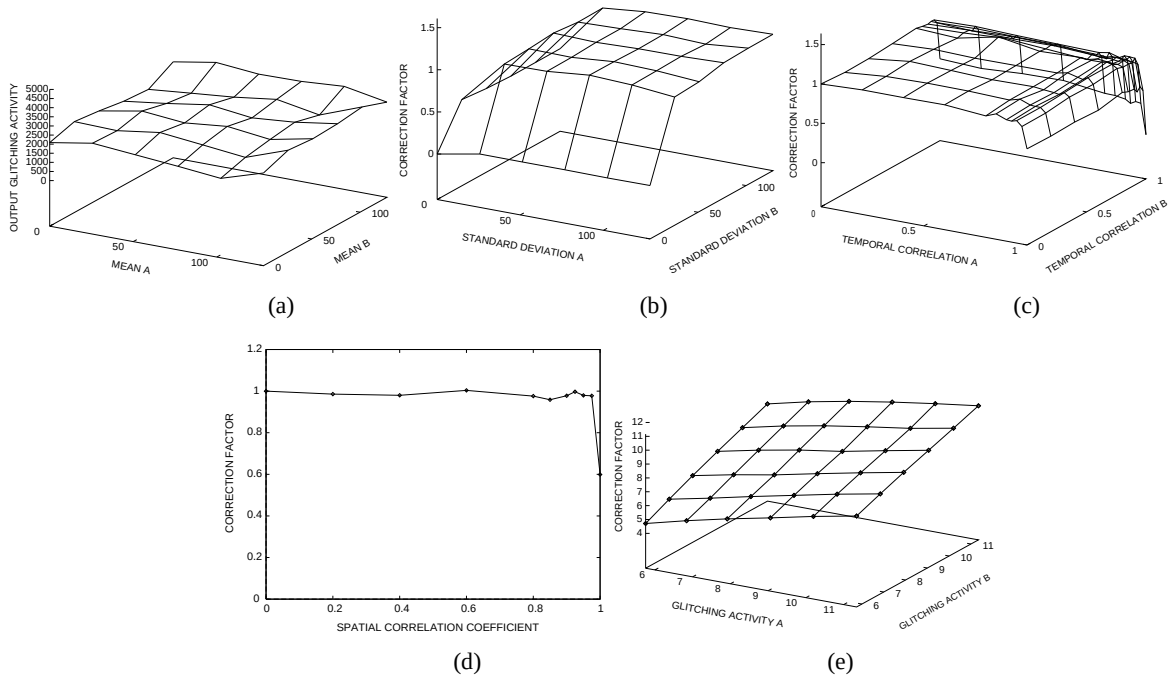


Figure 8: Glitching activity models for an 8-bit subtractor

The first technique, called *variable elimination*, attempts to reduce the number of independent variables in the glitching activity model by identifying ones whose variations affect the dependent variable (output glitches) minimally. We use techniques from multi-variable data analysis for this purpose. Given a set of samples (each sample consists of a set of values for the independent variables $y_1 \ldots y_n$, and the corresponding observed value that the dependent variable (z) assumes), we can use the *ANOVA test* [4] to check whether the null hypothesis for any given variable y_i is true, *i.e.*, whether different values of y_i had any impact on the observed sample values of z [12].

The second technique, called *model decomposition*, attempts to decompose the function $f()$ into multiple sub-functions 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 standard 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.

In the case of the subtractor, we found that the basic model of Equation (5) can be decomposed into the following equation.

$$\begin{aligned} Gl_{OUT} &= f_{gl_1}(Mean_A, Mean_B) * f_{gl_2}(SD_A, SD_B) \\ &* f_{gl_3}(TC_A, TC_B) * f_{gl_4}(SC_{A,B}) \\ &* f_{gl_5}(Gl_A, Gl_B) \end{aligned} \quad (6)$$

The independent variables have been partitioned in the above equation into the groups, $\{Mean_A, Mean_B\}$, $\{SD_A, SD_B\}$, $\{TC_A, TC_B\}$, $\{SC_{A,B}\}$, and $\{Gl_A, Gl_B\}$. 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 reasonably independent in their effects. As before, assuming we discretize the domain for each parameter into 5 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 Equation (5).

Note that the sub-functions are composed using a multiplicative relationship. Hence, we can view one of the sub-functions, say $f_{gl_1}()$ as a *base glitch model*, and the remaining sub-functions as multiplicative correction factors. The base glitch model in the form of a contour plot is shown in Figure 8(a). Consider the sub-function $f_{gl_2}(SD_A, SD_B)$. In order to construct the model for this sub-function, we 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 [13]. The subtractor implementation is simulated using CSIM and the values of glitching activity at the output are recorded. The plot in Figure 8(b) shows the results. The values in the plot are normalized to the case of $SD_A = SD_B = 25.6$, since the base power model ($f_{gl_1}()$) 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 subtractor, the value of $f_{gl_2}()$ is estimated from the values at the four points nearest to it in the discretized SD_A, SD_B space, using standard linear intrapolation techniques. The models for $f_{gl_3}() \ldots f_{gl_5}()$ are provided in Figures 8(c) through 8(e).

In order to generate the model for $f_{gl_5}()$, we needed to generate input sequences to the subtractor that have varying *glitching activities* at A and B. In order to do that, we used a circuit configuration similar to that shown in Figure 6. We added two multiplexers to feed the inputs of the subtractor, and connected their select input to the output of a ($<$) comparator. Given a sequence of inputs 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 subsection.

V. RTL Power Models

Given the zero-delay and glitching activity estimates obtained from the first two phases of the tool, the third phase uses several

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

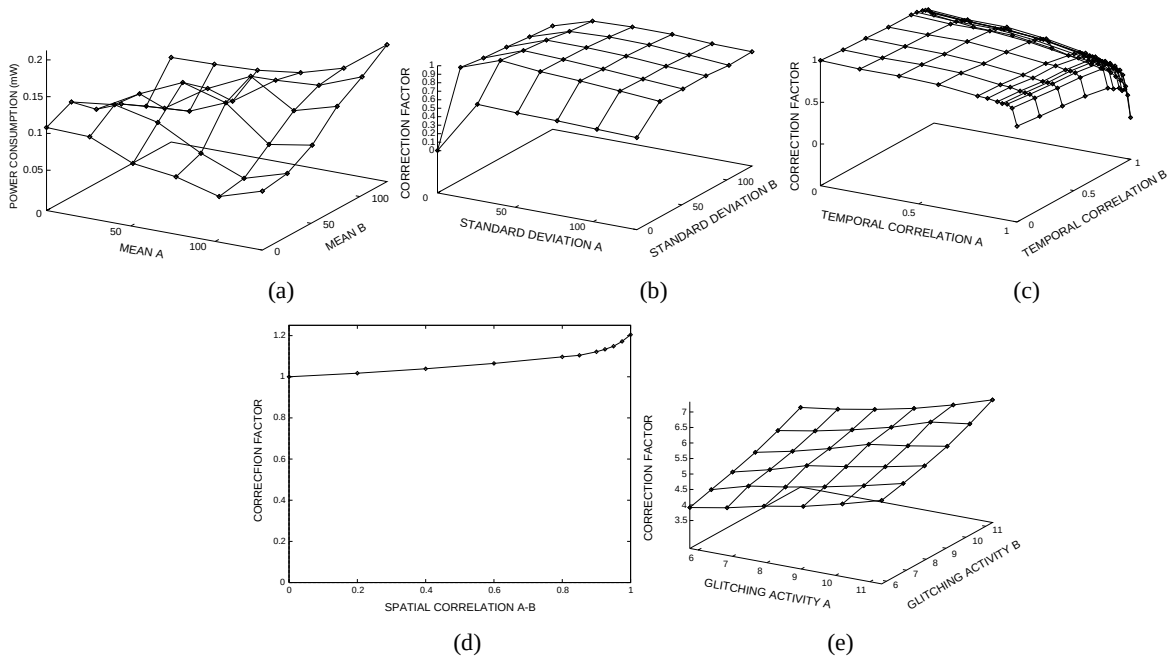


Figure 9: Word-level power models for a less-than ($<$) comparator

word- and bit-level power models that we discuss in this section.

Word-level power modeling techniques. 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 Subsection IV.B, 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 was decomposed into sub-functions, and the model for each sub-function is illustrated by the plots in Figure 9. Note that glitchy inputs can cause a substantial increase in the power consumption of the comparator, as shown in Figure 9(e), confirming the importance of considering glitching activity at the inputs of various RTL blocks during estimation.

Bit-level power models. For data path blocks which do not associate any word-level value to their multi-bit input signals, we derive power models that use the bit-level signal and 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.

$$\begin{aligned} Mux\_Power &= \sum_{i=1}^{n} Base\_Power(i) \\ &\quad + A\_Gl\_Power(i) \\ &\quad + B\_Gl\_Power(i) \\ &\quad + Sel\_Gl\_Power(i) \end{aligned} \tag{7}$$

$$\begin{aligned} A\_Gl\_Power(i) &= Gl(A_i) * (K_{A0} * P(Sel = 0) \\ &\quad + K_{A1} * P(Sel = 1)) \\ B\_Gl\_Power(i) &= Gl(B_i) * (K_{B0} * P(Sel = 0) \\ &\quad + K_{B1} * P(Sel = 1)) \\ Sel\_Gl\_Power(i) &= Gl(Sel) * \\ &\quad (K_{Sel00} * P(A_i = 0, B_i = 0) \\ &\quad + K_{Sel01} * P(A_i = 0, B_i = 1) \\ &\quad + K_{Sel10} * P(A_i = 1, B_i = 0) \\ &\quad + K_{Sel11} * P(A_i = 1, B_i = 1)) \end{aligned}$$

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 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 $Mux\_Power[]$. 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 calculate the $Base\_Power$ term.

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 $K_{Sel00}, ..., K_{Sel11}$ are obtained by performing controller experiments with the circuit configuration shown in Figure 6 by fixing the multiplexer data inputs to 00, 01, 10, and 11 (note that even when glitches at Sel do not propagate to the output, they could cause power dissipation internal to the multiplexer). The purpose of the $A\_Gl\_Power(i)$ and $B\_Gl\_Power(i)$ 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 equation.

$$\begin{aligned} Reg\_Power &= n * K_{CLK} * Act(CLK) \\ &\quad + \sum_{i=1}^{n} K_{OUT_i} * Act(OUT_i) \\ &\quad + \sum_{i=1}^{n} K_{IN_i} * Gl(IN_i) \end{aligned} \tag{8}$$

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 the value of IN constant. The second term accounts for the power dissipated when the value stored in the register changes. 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 $K_{OUT_i} = K_{OUT_j}$ for all i, j.

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 [6], 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 Subsection IV.A. We multiply the total 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.

VI. 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 Figure 1 [14], the `barcode` preprocessor [14], and a circuit implementing a part of the `X.25` communications protocol [15]. All the RTL circuits were obtained by synthesizing them from control-flow intensive specifications using the SECONDS high-level synthesis system [4, 15].

We built glitching activity as well as power models for the components of the RTL library used by SECONDS. For the first phase of our power estimation tool (zero-delay RTL simulation), we used a test bench that was derived using the 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 for predicting glitching activity at various data path and control signals using the models presented in previous sections. Thereafter, zero-delay and glitch statistics were used to calculate a number for power consumption for each data path block and for the controller. We performed logic synthesis optimizations on the controller, mapped the controller and data path to a commercial 0.8 micron technology library, and estimated power consumption using the tool CSIM.

We performed experiments to demonstrate the following claims (i) the estimates obtained by our RTL power estimation tool compared favorably to estimates obtained using CSIM, and (ii) the importance of estimating glitching activity and using it to enhance the accuracy of power estimates for the various RTL circuit blocks. Table 2 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 control and data path signals (column **RTL Est. W/O Gl.**). In order to obtain RTL power estimates ignoring the effects of glitches, we used our tool but instructed it to not perform the second phase of glitching activity estimation. Note that the estimates thus obtained do include the effects of glitches within each RTL block. For the second and third cases, the table reports the power estimate as well as the percentage error with respect to CSIM.

Table 2: Experimental Results

Circuit	CSIM	RTL Est.		RTL Est. W/O Gl.	
		Pow.	%Err	Pow.	%Err
`GCD`	1.64mW	1.53mW	6.71	1.28mW	21.95
`Bar-code`	2.82mW	2.94mW	4.25	2.41mW	14.54
`X.25`	3.38mW	3.18mW	5.92	2.89mW	14.49

The results indicate that the proposed RTL switching activity and power estimation techniques result in power estimates that range from within 4.25% 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 data path signals during RTL power estimation.

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*, pp. 355–361, Oct. 1992.

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

[3] P. K. Jha and N. D. Dutt, "Rapid estimation for parameterized components in high-level synthesis," *IEEE Trans. VLSI Systems*, 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*, pp. 736–742, Nov. 1994.

[5] S. R. Powell and P. M. Chau, "Estimating power dissipation of VLSI signal processing chips: the PFA technique," in *Proc. VLSI Signal Processing IV*, pp. 250–259, 1990.

[6] P. Landman and J. M. Rabaey, "Architectural power analysis: the dual bit type method," *IEEE Trans. VLSI Systems*, vol. 3, pp. 173–187, June 1995.

[7] 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*, pp. 81–86, Apr. 1995.

[8] F. N. Najm, "Towards a high-level power estimation capability," in *Proc. Int. Symp. Low Power Design*, pp. 87–92, Apr. 1995.

[9] P. Landman and J. M. Rabaey, "Activity-sensitive architectural power analysis for the control path," in *Proc. Int. Symp. Low Power Design*, pp. 93–98, Apr. 1995.

[10] *CSIM Version 5 Users Manual.* Systems LSI Division, NEC Corp., 1993.

[11] S. Dey, A. Raghunathan, and N. K. Jha, "Register-trasfer level estimation techniques for switching activity and power consumption," Tech. Rep. 96-C017-4, NEC USA, Princeton, NJ, Apr. 1996.

[12] G. Casella and R. L. Berger, *Statistical Inference*. Duxbury Press, Belmont, CA, 1990.

[13] D. E. Knuth, *The Art of Computer Programming, Vol 2*. Addison-Wesley Publishing Co., Reading, MA, 1980.

[14] High-level synthesis benchmarks, CAD Benchmarking Laboratory, Research Triangle Park, NC.

[15] S. Bhattacharya, S. Dey, and F. Brglez, "Performance analysis and optimization of schedules for conditional and loop-intensive specifications," in *Proc. Design Automation Conf.*, pp. 491–496, June 1994.