Activity-Sensitive Architectural Power Analysis for the Control Path

Paul E. Landman
Integrated Systems Laboratory
Texas Instruments
Dallas, Texas

Jan M. Rabaey
EECS Department
University of California, Berkeley
Berkeley, California

Abstract

Prompted by demands for portability and low-cost packaging, the electronics industry has begun to view power consumption as a critical design criteria. As such there is a growing need for tools that can accurately predict power consumption early in the design process. Many high-level power analysis tools don't adequately model activity, however, leading to inaccurate results. In a previous publication, we introduced architecture-level power analysis techniques for datapath and memory modeling. This paper focuses on the control path, describing a novel power analysis strategy known as the Activity-Based Control (ABC) model. Architecture-level estimates are then compared to switch-level simulations of three chips: a divider, a speech recognition controller, and a microprocessor. The average error observed in the control path power estimates is 13% with a maximum error of 29%.

1 Introduction

Currently, the portable consumer electronics market is undergoing a period of rapid growth. The constraints of battery operation have forced designers to focus on power considerations as well as speed and area. Even for non-portable systems, the high-cost of packaging and cooling power-hungry devices has led to increasing efforts aimed at minimizing power.

This trend further complicates the design process as engineers must now consider joint optimization not only of area and speed, but also of power. CAD tools can help manage this complexity by providing the designer with feedback relating to these three important parameters. Analysis tools such as SPICE and PowerMill [1] can be useful in this capacity, but both require a transistor-level netlist and, therefore, they can only be applied towards the end of the design flow when changes become expensive to implement.

This paper describes techniques for estimating power consumption given an architecture-level description of a system. The paper builds on research described in [2] and [3], which describe how to estimate the power consumed by datapath and memory elements using the Dual Bit Type (DBT) model. Here we extend that work by introducing techniques for control path power analysis. This Activity-Based Control (ABC) model is described in Section 3. Section 4 discusses a chip-level power analysis tool, called SPA, that employs the ABC model and also describes how the complexity and activity parameters required by the ABC model can be derived. In Section 5, SPA's results are verified using several realistic examples including a programmable microprocessor.

2 Previous Work

Most power estimation research deals with transistor- or gate-level modeling as noted in [4]. Here, we are interested in tools that operate at a higher abstraction level, where the primitives are entire combinational logic blocks rather than individual gates.

To date, most high-level power estimation techniques suitable for the control path rely on the concept of gate equivalents [5][6]. Under this regime, module complexity is approximated by the number of reference gates (e.g. two-input NAND's) required to implement the function. Power is then estimated by multiplying the average power per gate by the gate equivalent count.

The advantage of this straightforward approach is that it requires minimal design information as input. The disadvantage is that the power per gate is characterized assuming fixed activity levels - typically, corresponding to independent white noise inputs. Often this assumption is not justified and, as later results will show, estimates based on white noise activity assumptions often err by a factor of two or more. In the following sections we propose architectural control path power analysis techniques that are sensitive to activity.

3 Control Path Power Modeling

Controllers direct the sequence of operations to be executed by the datapath, initiate memory accesses, and coordinate data transfers over interconnect. The behavior of a typical controller can be described by a *state transition graph* (STG) or, equivalently, by a *control table*, which for all present state and input values specifies next state and output values. Controllers are often implemented using the finite state machine (FSM) structure of Figure 1. The *implementation style* of the combinational logic can take many forms: e.g. ROM, PLA, or random logic (i.e. standard cells).

The task of architectural controller power analysis is to produce an estimate of the final implementation power given only the target implementation style and the state machine description, say, in the form of a control table. At the architecture level, accurately estimating power for the control path is more difficult than for the datapath. The circuits used by datapath elements (such as adders and multipliers) are often known a priori, in effect, fixing the physical capacitance of the modules. In contrast, the physical capacitance of a controller depends on the contents of the control table, which are not known until run time.

Since it is impractical to characterize each controller style for all possible control table contents a priori, we instead characterize for "random" tables. This results in a fixed physical capacitance somewhere between the extremes of an "empty" (all zero) table and a "full" (all one) table. Then, for each implementation style, prototype controllers of different complexities can be synthesized a priori and characterized for various activity levels.

The discussion of the ABC model will be divided into four parts. Section 3.1 will describe model parameters that influence power regardless of implementation style, while Section 3.2 will present target-specific examples of ABC models. Section 3.3 will discuss techniques for library characterization, and Section 3.4 will review the power analysis method being proposed here.

3.1 Target-Independent Parameters

Two classes of parameters influence power regardless of the target implementation style: complexity parameters and activity

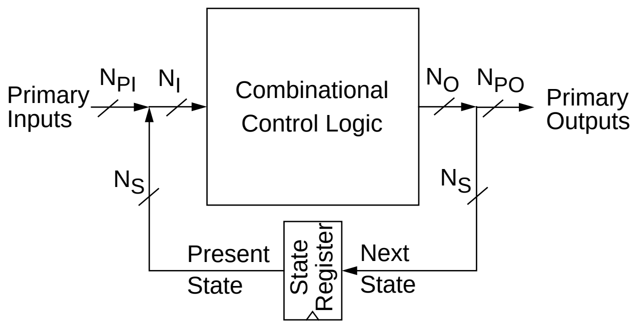

Figure 1. Typical FSM structure

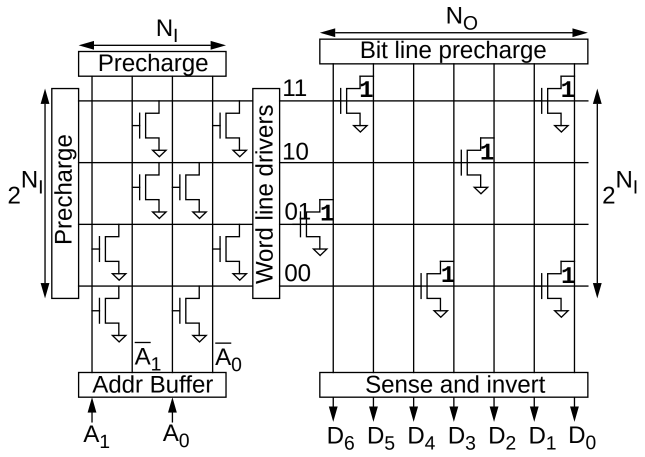

Figure 2. Sample ROM structure (in this case, 4x7)

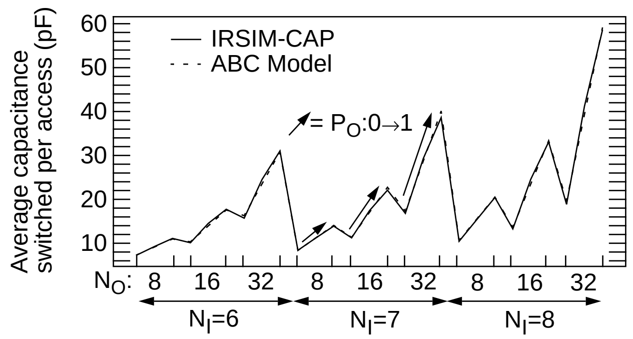

Figure 3. Verification of ROM capacitance model

parameters. The complexity (or size) of a controller directly influences its physical capacitance and, therefore, its power consumption. The complexity of the combinational logic block in Figure 1 can be measured to some extent by the number of inputs, N_I, and outputs, N_O. An increase in N_O=N_S+N_{PO} requires additional logic to generate the larger number of next state bits, N_S, and/or primary outputs, N_{PO}. Similarly, a larger N_I=N_S+N_{PI} means more input decoding due to an increase in the number of present state bits, N_S, and/or primary inputs, N_{PI}. Actually, N_I is only a good measure of input-plane complexity when exhaustive "address" decoding is used (as in a ROM). In other cases, the number of min-terms, N_M, in the logic-minimized control table is a better measure of complexity. Section 4.1 describes how to estimate N_I, N_M, and N_O.

Complexity gives us some indication of the physical capacitance contained in a controller, but if the capacitance is not switched, no power is consumed. Since at the architecture level we treat combinational logic blocks as black boxes, activity can best be described by external measures. The *input activity* tells us something about how much switching occurs in the input plane, or "address" decoding, portion of the combinational logic. For static logic, the transition activity, α_I, is the proper measure of circuit activity in the input plane. It is equal to the fraction of input bits (including state) that switch each cycle. For dynamic logic, the signal probabilities of the inputs - that is, the probability that an input bit is one (P_I) or zero (1-P_I) - determine circuit activity, since precharging to a known state each clock cycle negates the influence of the previous signal value on current power consumption. The input activity parameters (α_I, P_I) tell only half of the story, however - we also require a measure of *output activity*: (α_O, P_O). Section 4.2 will describe techniques for acquiring the necessary activity parameters through functional simulation.

3.2 Target-Specific Capacitance Models

The ABC method uses a library-based approach, allowing the user to define a unique capacitance model for each controller implementation style in the library. If desired, these capacitance estimates can be converted to an equivalent energy, $E=CV^2$, or power, $P=CV^2f$. This section illustrates how to construct an ABC capacitance model for three case studies: a ROM-based controller, a PLA-based controller, and a random logic controller. The same concepts are readily applicable to other implementation styles.

3.2.1 ROM-Based Controllers

A capacitance model should describe how the average capacitance switched during a single access scales with changes in complexity and activity. For the ROM structure of Figure 2 the appropriate capacitance model is given by:

$$C_T = C_0 + C_1 N_I 2^{N_I} + C_2 P_O N_O 2^{N_I} + C_3 P_O N_O + C_4 N_O \qquad (1)$$

The terms relate to power consumed in the input plane (address decoding) and the output plane (bit line charging). The input-plane complexity is proportional to the number of columns and rows in the plane: $N_I 2^{N_I}$. Since both true and complement address lines are present, the activity is independent of external address activity - i.e. during evaluation, half of the precharged lines will remain high and the other half will discharge, regardless of external input activity. Therefore, $C_1 N_I 2^{N_I}$ contains no explicit activity factor.

In the output plane, power is dominated by charging and discharging the bit lines whose lengths (and capacitances) are proportional to the number of rows in the array, 2^{N_I}. Initially high, the bit lines corresponding to **1** output bits (on average, $P_O N_O$ of them) discharge during an access and then precharge prior to the next read cycle. This explains the $C_2 P_O N_O 2^{N_I}$ term. The buffering circuitry for the N_O outputs gives rise to the $C_3 P_O N_O$ term, but there is also an activity-independent term, $C_4 N_O$, since the sense circuitry produces both true and complemented signals.

The circuit- and technology-dependent capacitive coefficients C_0, C_1, C_2, C_3, and C_4 are extracted through a library characterization process described in Section 3.3. Figure 3 contains a comparison between IRSIM-CAP and the ABC model after characterization for a 1.2 µm CMOS technology using random control table contents. IRSIM-CAP [2] is a modified version of the switch-level simulator IRSIM [7] with improved capacitance measurement capabilities. The rms ABC model error is about 2.5% and the maximum error is 4.5%. The arrows in the figure denote results for controllers of fixed complexity for which the output signal probability, P_O, varies from zero to one. The fact that power consumption varies significantly with P_O is a strong argument in favor of a modeling strategy which accounts for activity.

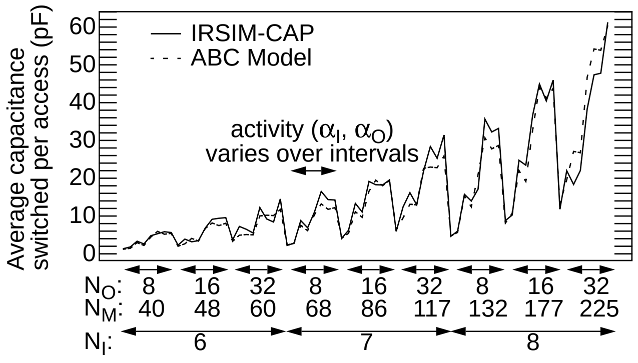

Figure 4. Verification of random logic ABC model

Label	**Address A[7:2]**	**Map A[1:0]**	**Data**
A(0,0)	`000000`	`00`	`00000000`
A(0,1/2)		`01`	`01010101`
A(0,1)		`11`	`11111111`
A(1/2,0)	`010101`	`00`	`00000000`
A(1/2,1/2)		`01`	`01010101`
A(1/2,1)		`11`	`11111111`
A(1,0)	`111111`	`00`	`00000000`
A(1,1/2)		`01`	`01010101`
A(1,1)		`11`	`11111111`

Table 1: Patterns for generating desired I/O activities

3.2.2 PLA-Based Controllers

The principle difference between a PLA and a ROM is that the input plane of the ROM performs a full decoding of all possible addresses, while a PLA uses logic minimization to reduce the amount of decoding required. As a result, the height of the input and output planes in a PLA is given by the number of unique minterms, N_M, which will usually be less than 2^{N_I}. For a sample PLA with a static input plane and a dynamic output plane, the capacitance model was found to follow:

$$C_T = C_0 \alpha_I N_I N_M + C_1 P_O N_O N_M + C_2 P_O N_O + C_3 N_O N_M + C_4 N_M \quad (2)$$

We can compare this model to the ROM expression of (1). As expected, 2^{N_I} has given way to N_M. Also, since this PLA uses static input decoding, an α_I term has been added to model the effect of input activity on power. In the output plane, since no differential signaling is used, the $C_4 N_O$ term disappears. Furthermore, this PLA uses a clocked virtual ground node in the output plane that charges to V_{dd} each cycle, regardless of the output signal values. This leads to the activity-independent terms: $C_3 N_O N_M$ and $C_4 N_M$. Characterization of the PLA (in a 1.2 µm technology) yields rms and maximum errors of 2.5% and 6.3%, respectively.

3.2.3 Random Logic Controllers

Since a random logic implementation is much less regular than a PLA or ROM, it is more difficult to come up with an accurate capacitance model, but an approximate model for a standard cell controller implemented in static logic might be given by:

$$C_T = C_0 \alpha_I N_I N_M + C_1 \alpha_O N_O N_M \quad (3)$$

Since this example is based on static logic, the proper activity measures are α_I and α_O. The input-plane complexity is given by the product of the number of inputs, N_I, and the number of outputs that the plane produces, N_M. The same is true for the output plane, except in this case there are N_M inputs to the plane and N_O outputs.

The exact model and its capacitive coefficients will be a function of the standard cell library being used and, to some extent, the logic synthesis, placement, and routing tools being applied. A comparison of this model to switch-level simulations for a 1.2 µm cell library is shown in Figure 4. The results are for control tables with random contents that have been minimized using espresso and synthesized using MIS [8] and the Lager IV silicon assembly system [9]. While the agreement is not as good as the ROM and PLA models, the rms error is still an acceptable 15.1%.

3.3 Characterization Method

Before the controller capacitance models can be used, circuit- and technology-dependent values for the capacitive coefficients must be measured through a *characterization* process. For a given class of controller, the idea is to actually measure the capacitance switched within implementations of varying complexities for different input and output activities. The observations are then used to find capacitive coefficients that give the best fit to the measured data. The characterization process occurs in three phases: pattern generation, simulation, and coefficient extraction.

3.3.1 Pattern Generation

In the first phase, two distinct sets of data patterns are generated: the patterns stored in the control table that the FSM implements and the patterns applied as input to the controller during simulation. Since the control table patterns can affect the physical capacitance of the controller and since it would be impossible to characterize for all possible control tables, we instead generate patterns that result in some sort of average physical capacitance. A good approximation is to use a random output table with a uniform distributions of zeros and ones. In real controllers, however, not all outputs affect system behavior in all states. Setting a fraction (say, half) of the output table entries to don't-cares models this effect. PLA and random logic synthesizers can exploit the don't-cares by using logic minimization algorithms (such as espresso [8]).

The second phase of pattern generation entails producing input streams that exercise the controller over a full range of activities from 0% to 100%. Three evenly spaced activity levels (0, 1/2, and 1) can be realized by correct sequencing of three basic patterns: **`00`**...**`00`**, **`01`**...**`01`**, and **`11`**...**`11`**. The input activity can be controlled by using the patterns as input "addresses" to the combinational logic block. The output activity can be independently controlled by storing the same three data patterns at each of these three "addresses." While three different values cannot be referenced by a single address, they can, however, be referenced by "similar" addresses. As shown in Table 1, the two least significant address bits can be used to map the three different data patterns to similar, but unique, addresses. The left-most column provides a convenient label for each (input, output) activity pair. So while the output table is still filled primarily with random data, it also contains nine deterministic values that allow precise control of input and output activities during characterization.

Using this strategy, desired input and output *signal probabilities* can be chosen from any one of nine distinct possibilities, $(P_I, P_O) \in \{0, 1/2, 1\} \times \{0, 1/2, 1\}$, simply by accessing address $A(P_I, P_O)$. It is also possible to generate any of nine different *transition* activities, $(\alpha_I, \alpha_O) \in \{0, 1/2, 1\} \times \{0, 1/2, 1\}$, by accessing an address sequence $A(\alpha_I^0, \alpha_O^0) \rightarrow A(\alpha_I^1, \alpha_O^1)$ that satisfies $\alpha_I = |\alpha_I^0 - \alpha_I^1|$ and $\alpha_O = |\alpha_O^0 - \alpha_O^1|$. Desired address sequences can be represented graphically by associating the ordered activity pairs with a coordinate system as shown in Figure

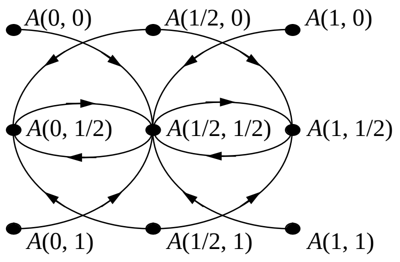

Figure 5. Address sequences for $(\alpha_I, P_O) = (1/2, 1/2)$

5, which shows address transitions corresponding to the activity pair $(\alpha_I, P_O) = (1/2, 1/2)$. As the figure demonstrates, the input transition activity determines how many columns each edge traverses, and the output signal probability determines at which row each edge terminates.

Recalling (2) from Section 3.2.2, input transition activity, α_I, and output signal probability, P_O, are the activity parameters used for the PLA-based controller. Full characterization would include nine activity pairs: $(\alpha_I, P_O) \in \{0, 1/2, 1\} \times \{0, 1/2, 1\}$. While, the standard cell controller requires nine as well, $(\alpha_I, \alpha_O) \in \{0, 1/2, 1\} \times \{0, 1/2, 1\}$, the ROM capacitance model requires only three activity values, $P_O \in \{0, 1/2, 1\}$. In each case, the simulated capacitance observations for individual transitions are averaged together to produce a characteristic capacitance for each I/O activity level and implementation style.

3.3.2 Simulation and Coefficient Extraction

During simulation, a module corresponding to a given set of complexity parameters (e.g. number of inputs, min-terms, and outputs) is synthesized and simulated for the data patterns generated above. The simulation can use either a circuit- or gate-level tool (e.g. SPICE, PowerMill [1], IRSIM [7]) depending on the time allotted for characterization and the accuracy desired. The output of this process is a number of capacitance observations - one for each controller style, complexity, and activity level.

Coefficient extraction refers to the process of deriving best-fit model coefficients from the raw simulation data. This can be achieved using techniques such as least-squares regression, which minimizes the mean-squared error of the model. In vector notation, this amounts to solving the following matrix equation for the capacitive coefficient vector, $\mathbf{C}_{eff} = [C_0 \;\; C_1 \;\; \ldots]^{\mathrm{T}}$:

$$\mathbf{C}_{sim} = \mathbf{P}\mathbf{C}_{eff} + \mathbf{e} \tag{4}$$

where $\mathbf{C}_{sim}$ is a vector of simulated capacitance observations, $\mathbf{P}$ is a matrix of complexity and activity parameter values corresponding to the observations, and $\mathbf{e}$ is the minimized error.

3.4 ABC Power Analysis Method

Analyzing controller power requires a capacitance model, a set of capacitive coefficients, and the appropriate complexity and activity parameters. Given these inputs, the combinational logic capacitance model can be evaluated (in vector form) as:

$$C_T^{CL} = \mathbf{C}_{eff}^{CL} \cdot \mathbf{N} \tag{5}$$

where $\mathbf{C}_{eff}^{CL}$ is a vector of capacitive coefficients and $\mathbf{N}$ is a vector of complexity and activity parameters for a given implementation style. Aside from the combinational logic block, the state register also contributes to the overall power consumption of the controller. Since N_S state bits must be stored, the capacitance model for the state register will have the following basic form:

$$C_T^{reg} = C_0 \alpha_S N_S \tag{6}$$

where α_S is the activity of the state bits. The total controller capacitance per access, then, is: $C_T = C_T^{CL} + C_T^{reg}$. If many accesses are involved, the total capacitance switched over a number of input control transitions, N_{CT}, can be computed using the following expression: $C_T\big|_{\text{multi-cycle}} = N_{CT} \cdot C_T\big|_{\text{single-cycle}}$

In summary, the Activity-Based Control (ABC) model presented here explicitly accounts for activity and provides a general framework for modeling different controller structures. Using the ABC method, analyzing controller power amounts to plugging the appropriate activity and complexity parameters into an equation that weights these parameters by technology- and implementation-dependent capacitive coefficients.

4 SPA: An Architectural Power/Area Analyzer

This control path analysis strategy has been combined with datapath, memory, and interconnect analysis techniques in an architectural power/area analysis tool called SPA. The primary input to SPA is a hierarchical RT-level description of the architecture under consideration. Currently, SPA uses a textual architectural description language (ADL) for this purpose; however, the same information could be provided by a graphical schematic capture interface. The control path is described using a control description language (CDL) that employs control tables, which specify how the next state and outputs of each control module relate to the present state and inputs. In order to maintain a relatively high level of abstraction, CDL allows the user to specify control signals and states as enumerated (symbolic) types rather than bit vectors. With the structure and behavior of the architecture fully defined, the next step in the process is to derive the complexity and activity parameters required by the power and area analysis models. Since this paper deals with control path power analysis, we focus on the parameters required by the ABC model.

4.1 Complexity Analysis

ABC complexity analysis reduces to ascertaining the “size” of control buses and modules. For example, if a control bus carries the function input to an ALU, the type of the bus might be defined as: $\text{ALU\_TYPE} \equiv \{\text{ADD, SUB, SHIFT, CMP}\}$. The word length, N, (in bits) required to represent an enumerated type, T, which can take on $|T|$ different values is:

$$N = \lceil \log_2 |T| \rceil \tag{7}$$

For instance, two bits are required for a binary encoding of an ALU_TYPE control bus. For a “one-hot” encoding strategy, the appropriate complexity formula would simply be $N = |T|$. One way to handle multiple encoding schemes is to have the user associate an *encoding style attribute* with each data type or entity.

For a control module, two important complexity parameters are the number of input bits, N_I, and the number of output bits, N_O. Since the module may take many control buses as input and may feed multiple control buses at its output, these parameters are calculated by summing individual control bus widths:

$$N_{I/O} = \sum_{i \in \{\text{I/Os}\}} N_{I/O_i} = \sum_{i \in \{\text{I/Os}\}} \lceil \log_2 |\text{I/O\_TYPE}_i| \rceil \tag{8}$$

The number of min-terms, N_M, in the minimized control table is another important complexity parameter. This is more difficult to estimate since the amount of minimization that can be performed depends on the binary encoding of the control table, which is usually not available at this stage of design. We can approximate N_M by assuming some binary mapping (e.g. random), performing logic minimization (e.g. espresso [8]), and using the number of min-terms in the minimized table as an estimate of N_M.

4.2 Activity Analysis

The next step in the process is to derive the ABC activity statistics through a functional simulation of the architecture. Many RT-level simulators would be suitable for this task, but currently SPA uses a VHDL simulator provided by Synopsys. A code generator is used to produce structural VHDL for each ADL block and behavioral VHDL for each CDL block. A collection of activity monitors attached to buses and modules accumulate activity statistics during simulation. If the chip requires data/instruction inputs, these must be supplied by the user. Since functional simulation is quite fast, run time is usually on the order of seconds or minutes.

The first ABC activity parameter for a control bus is the control word transition count, N_{CT}. Signal probability P and transition activity α are the two other activity statistics. If the binary encoding of symbols is known, exact values for P and α can be calculated during simulation. Otherwise, we can assume a random binary encoding (i.e. $P = 1/2$ and $\alpha = 1/2$).

Control modules require two sets of (α, P) activity statistics: one for the logic block input and one for the output. The input and output may consist of several control buses bundled together. N_{CT} counts transitions of the bundled input word, which are defined to occur when *any* of the component input signals transition. If we assume a random encoding, we again expect half of the input and output bits to be one, so $P_I = P_O = 1/2$. As for the transition activities, the following expression applies:

$$\alpha_{I/O} = \frac{\sum_{i \in \{\text{I/Os}\}} 0.5 N_{I/O_i} N_{CT_i}}{N_{I/O} N_{CT}} \qquad (9)$$

where i refers to the individual control buses that make up the input/output bundles. For the input activity case, this equation can be explained as follows: $0.5N_{I_i}$ is the number of bits that switch on average when input i makes a transition. This is then weighted by the total number of transitions that input i makes, N_{CT_i}, to yield the number of bits within input i that toggle during the entire simulation. This is then summed across all of the individual inputs to yield the total number of bits that switch in the bundled input word. Dividing this by the number of input transitions, N_{CT}, and the number of input bits, N_I, gives the fraction of input bits that toggle during a typical input transition - that is, α_I.

4.3 Power/Area Analysis

Finally, the complexity and activity parameters are fed to the core area and power analysis routines. SPA then steps through each datapath, memory, control, and interconnect entity in the design, applying the appropriate power and area analysis models. Finally, SPA dumps its results, categorizing area and power consumption by type of component. This points the designer to the most power-intensive portions of the implementation and provides useful guidance for further optimizations.

5 Results

This section presents results gathered using SPA. The first example shows SPA's applicability to datapath-intensive applications, while the second example is a control-intensive design. The final case study is a programmable instruction set processor that demonstrates SPA's ability to handle a real-world example containing significant datapath and control components. As the microprocessor employs on-chip instruction and data memories, this example also serves as a test case for memory-intensive designs.

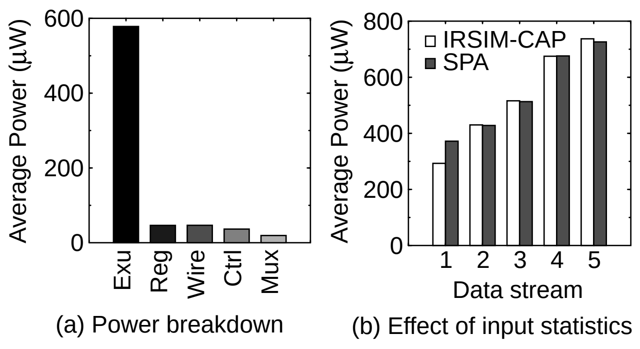

Figure 6. SPA power analysis results for divider

5.1 Datapath Intensive: A Newton-Raphson Divider

The first example is that of a hardware divider based on the Newton-Raphson iteration. Such a divider might be used to implement floating-point division in a microprocessor. For instance, Texas Instruments employed a similar iterative division algorithm in their TI 8847 processor. The architecture contains an eight-word register file that holds the 1/*b* lookup table, as well as several other registers that store the dividend, the divisor, and intermediate results. The dominant feature of the datapath is a 16x16 carry-save array multiplier. The power predictions of Figure 6a are for a voltage supply of 1.5V and a clock rate of 5 MHz.

To verify the accuracy of these predictions, the divider has been implemented down to layout. The total area of the final implementation is 4.40 mm^2. SPA's estimate of 4.68 mm^2 is within 6% of the actual area. The layout has also been extracted and simulated at the switch level using IRSIM-CAP. Figure 6b shows a comparison of the IRSIM-CAP and SPA results for several different data streams each of which leads to a different level of hardware activity. The error in the SPA power estimates varies from as little as 0.2% to at most 27% with an average error of 6%.

Now consider the time required to obtain architecture-level versus switch-level estimates of power. Specifying the design in ADL and CDL required approximately 15 minutes, and the time required to execute SPA including simulation and estimation was about 20 seconds on a Sun SPARCstation 10. In contrast, even with extensive use of existing hardware libraries, the layout-level implementation of the divider required about 6 hours. Switch-level simulation took another 10 minutes. So, for a relatively small sacrifice in accuracy, SPA allowed a 24x speedup in the design flow.

5.2 Control Intensive: A Speech Recognition FSM

This example deals with a global finite state machine for the front-end of a speech recognition system currently being developed by S. Stoiber at U. C. Berkeley. Since the chip-set is targeted at mobile applications, minimizing power consumption is a significant consideration. The state machine contains over 100 states, 10 inputs, and 25 outputs. The majority of the control table entries are redundant, making the FSM a good test of how well the ABC model can handle non-random control table contents.

Figure 7 shows the estimates provided by SPA for three possible implementation styles: ROM, PLA, and standard cell. The estimates are for a system clock of 3.3 MHz and a supply voltage of 1.5V. In order to verify that these predictions are reliable, all three implementations have been laid out, extracted, and simulated. The results from these physical designs have been included in Figure 7 for comparison. The average error in the area estimates is 17.3% and the maximum error is 22%. The average error in the power estimates is 12.6%, while the maximum error is 29% (standard

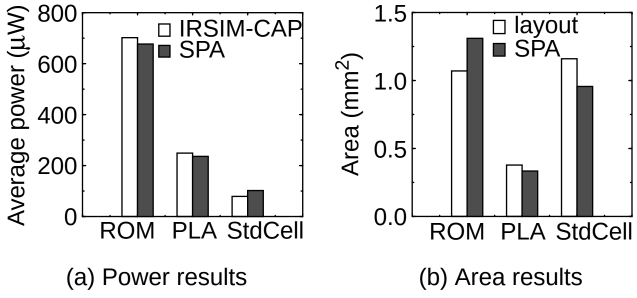

Figure 7. Results for three controller implementations

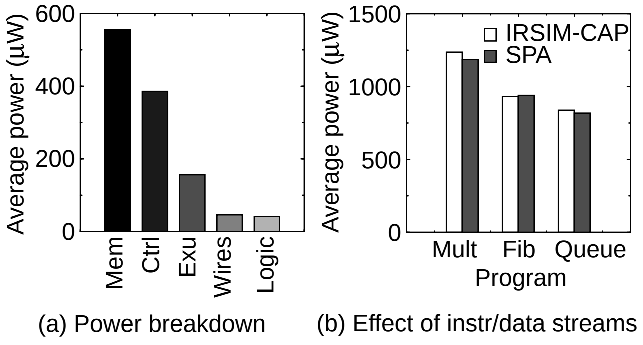

Figure 8. SPA power results for microprocessor

cell case). The standard cell case is more difficult since random logic is less regular and is more influenced by the binary contents of the particular control table being implemented.

For this example, on a Sun SPARCstation 10, SPA analyzed the area and power of the three controller implementations in about 2 minutes. In contrast, generating and extracting the layout for the three controllers took about 5 hours. Switch-level simulation required another 30 minutes.

5.3 Memory Intensive: A Microprocessor

This example considers a programmable instruction set processor with on-chip instruction and data memories. This case study demonstrates SPA's ability to handle memory-intensive, as well as datapath- and control-intensive, designs. The simple 16-bit microcoded processor can execute the following nine basic instructions: NOP, LDA, STA, ADD, SHR, CMP, JMP, BLT, and BGE. Figure 8a contains the power breakdowns as predicted by SPA for a multiplication program running on the processor with a 1.5V supply and a 10 MHz clock rate. The instruction and data memory accesses account for 47% of the total power consumption.

SPA can also be used to analyze the influence of instruction stream on power consumption. For example, Figure 8b shows the average power when running three different programs: a multiplication program (Mult), a Fibonacci sequence generator (Fib), and a circular queue (Queue). The power varies by as much as 38%, underscoring the need for prediction tools which account for the influence of computing activity on power consumption.

In order to confirm the accuracy of these predictions, the processor has been implemented down to the layout level. The extracted layout was simulated at the switch level for the three programs and the results are included in Figure 8b. The errors in the SPA power estimates are -4.1%, 0.88%, and -2.4%, respectively. The estimated area is within 8% of the actual area.

For this example, the architecture description took about two hours to produce, whereas layout required more than 25 hours. SPA was able to analyze the chip power consumption in about one minute (on a Sun SPARCstation 10), including parsing, simulation, and estimation times. In contrast, extraction and simulation of the layout required 45 minutes. Here again we see that the time savings offered by high level analysis tools is significant.

6 Conclusions

The Activity-Based Control (ABC) model was introduced for control path power analysis at the architecture level. The model improves upon contemporary architectural power analysis techniques (such as the gate-equivalent method) by accurately reflecting the effect of activity, as well as complexity, on power consumption. The ABC model has been incorporated into the SPA power/area analysis tool. SPA uses the ABC model to analyze the control path, the DBT model [2][3] to analyze the datapath and memories, and a strategy described in [10] to analyze interconnect and area. In this paper, SPA was used to gather results for several realistic design examples. Relative to switch-level power simulations of extracted layouts, SPA exhibited an average error rate of 7%. The average error for the control (ABC) portion was 13% with a maximum error of 29%. Area estimation errors were of similar magnitude averaging 14%. In conclusion, activity-sensitive architectural power modeling offers an attractive combination of speed and accuracy that will enable designers to evaluate and optimize power from the earliest stages of implementation.

Acknowledgments

The authors would like to thank Arthur Abnous for his significant contributions to the SPA project - in particular, his implementation of the ADL and CDL parsers and of the VHDL code generator. This research was funded by ARPA grant J-FBI 93-153 and by a fellowship from the National Science Foundation.

References

[1] A. Deng, “Power Analysis for CMOS/BiCMOS Circuits,” *1994 International Workshop on Low Power Design,* Napa Valley, CA, pp. 3-8, April 1994.

[2] P. Landman and J. Rabaey, “Black-Box Capacitance Models for Architectural Power Analysis,” *1994 International Workshop on Low Power Design*, pp. 165-170, April 1994.

[3] P. Landman and J. Rabaey, “Architectural Power Analysis: The Dual Bit Type Method,” accepted to *Transactions on VLSI Systems*, June 1995.

[4] F. Najm, “A Survey of Power Estimation Techniques in VLSI Circuits,” Transactions on VLSI Systems, vol. 2, no. 4, pp. 446-455, Dec. 1994.

[5] K. Müller-Glaser, K. Kirsch, and K. Neusinger, “Estimating Essential Design Characteristics to Support Project Planning for ASIC Design Management,” *IEEE International Conference on Computer-Aided Design ‘91*, pp. 148-151, Nov. 1991.

[6] C. Svensson and D. Liu, “A Power Estimation Tool and Prospects of Power Savings in CMOS VLSI Chips,” *1994 International Workshop on Low-Power Design*,p. 171-176, Apr. 1994.

[7] A. Salz and M. Horowitz, “IRSIM: An Incremental MOS Switch-Level Simulator,” *Proceedings of the 26th Design Automation Conference*, pp. 173-178, 1989.

[8] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[9] R. W. Brodersen ed., *Anatomy of a Silicon Compiler*, Kluwer Academic Publishers, 1992.

[10]P. Landman, *Low-Power Architectural Design Methodologies*, Ph.D. Dissertation, UC Berkeley, August 1994.