Activity-Sensitive Architectural Power Analysis

Paul E. Landman, *Member, IEEE*, and Jan M. Rabaey, *Fellow, IEEE*

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 models don't adequately model activity, however, leading to inaccurate results. This paper describes an activity-sensitive power analysis strategy for datapath, memory, control path, and interconnect elements. Since datapath and memory modeling has been described in a previous publication, this paper focuses mainly on a new Activity-Based Control (ABC) model and on a hierarchical interconnect analysis strategy that enables estimates of chip area as well as power consumption. Architecture-level estimates are compared to switch-level measurements based on net lists extracted from the layouts of three chips: a digital filter, a global controller, and a microprocessor. The average power estimation error is about 9% with a standard deviation of 10%, and the area estimates err on average by 14% with a standard deviation of 6%.**

I. Introduction

CURRENTLY, the portable consumer electronics market is undergoing a period of rapid growth. With portability comes a new set of design requirements. In particular, the constraints of battery operation have forced designers to focus on power considerations as well as speed and area. Furthermore, the high-cost of packaging and cooling power-hungry devices has led to increasing efforts aimed at minimizing power consumption even in high performance, nonportable systems.

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 feedback about the impact of various design decisions on these three important parameters. Analysis tools such as SPICE [1] and PowerMill [2] can be useful in this capacity, but they both require a transistor-level netlist as input and, therefore, they can only be applied toward the end of the design flow when major changes are difficult and expensive to implement.

This paper describes techniques for estimating area and power consumption given an architecture-level description of a system. The paper builds on research presented in [3], which described how to estimate the power of individual architectural blocks in a datapath. Here we extend that work by introducing techniques for control path and interconnect analysis, and by presenting results obtained from several design examples.

Fig. 1 gives an overview of the power analysis strategy that we propose in this paper. The inputs from the user are a description of a candidate architecture at the register-transfer level and a set of data and instruction inputs for which a power analysis is desired. Rather than attempting to find a single model for the entire chip, we take the approach of identifying four basic classes of components: datapath, memory, control, and interconnect. The modeling of power consumption for each class is addressed separately. For datapath and memory analysis, Section III reviews a word-level data model known as the Dual Bit Type (or DBT) model. An architectural model for control path power consumption, the Activity-Based Control (ABC) model, is presented in Section IV. Section V describes techniques for estimating the physical capacitance of interconnect (while producing chip area estimates as a side effect). It also describes how to combine these physical capacitances with the appropriate activity measures to obtain estimates of control and data bus power consumption. Section VI brings together the four classes of models, describing how they can be integrated to enable architectural power analysis for entire chips. This section also describes how the complexity and activity parameters required by the DBT and ABC models can be derived. In Section VII, the models are verified using several realistic examples including a programmable microprocessor.

II. Previous Work

The majority of the available literature on power estimation deals with transistor- or gate-level modeling [2], [4]–[9]. As stated above, however, we are interested in tools that operate at the architecture level. In the terminology of this paper, architecture refers to the register-transfer level of abstraction, where the primitives are blocks such as multipliers, adders, memories, and finite state machines. Two principal strategies have been proposed for estimating power at this level.

The first technique is based on the concept of gate equivalents. In this method, gate equivalent counts are used to roughly describe the complexity of modules within a chip. The gate equivalent count specifies the approximate number of reference gates (e.g., two-input NAND's) that are required to implement a particular function (e.g., a 16-b counter). The power required for that function can then be estimated by multiplying the approximate number of gate equivalents required by the average power consumed per gate. This is the essence of the strategy used in the Chip Estimation System [10]. Svensson and Liu also model logic power consumption using gate equivalents, but attempt to improve overall

Manuscript received December 1, 1995. This work was supported by ARPA Grant J-FBI 93-153 and by a Fellowship from the National Science Foundation. This paper was recommended by Guest Editors M. Pedram and M. Fujita.

P. E. Landman is with the DSP R&D Center, Texas Instruments, Dallas, TX 75265 USA.

J. M. Rabaey is with the EECS Department, University of California, Berkeley, CA 94720 USA.

Publisher Item Identifier S 0278-0070(96)04856-7.

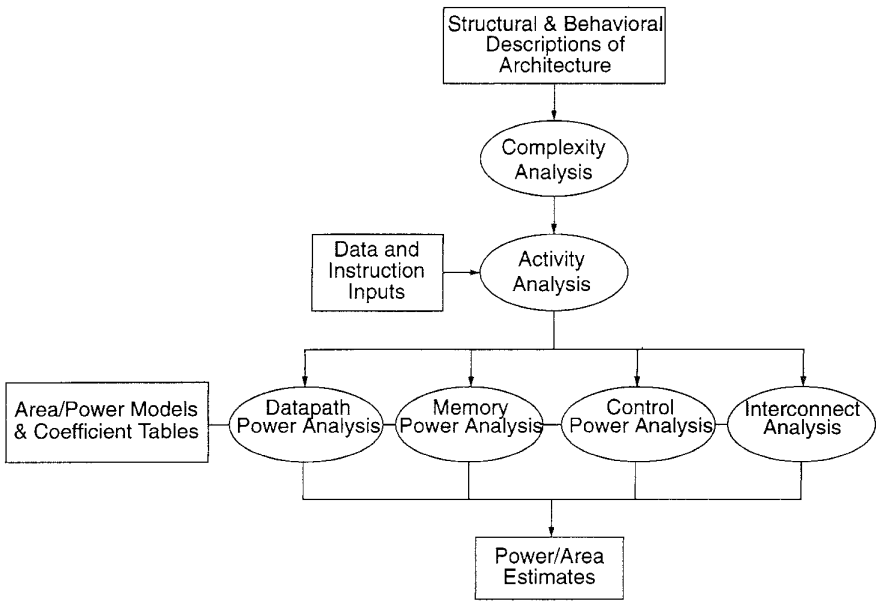


Fig. 1. Overview of architectural power analysis strategy.

accuracy by using custom models to handle memories [11]. Both approaches use derivatives of Rent's Rule to estimate interconnect length and capacitance [12].

The second principal modeling strategy can be classified as a precharacterized cell library approach. Under this scheme, instead of using a single gate-equivalent model for all "logic" blocks, a separate model is supplied for each block in the library: multipliers, adders, buffers, etc. These custom models better reflect how the complexity of a specific block influences its power. This technique was first proposed by Powell and Chau who termed it the Power Factor Approximation method [13].

The advantage of both these approaches is that they require minimal design information as input and that they are straightforward to apply. This simplicity, however, comes at the expense of accuracy. For, while both techniques do a satisfactory job of relating design complexity to power, they do not adequately account for the impact of activity on power. In both cases, power models are characterized assuming fixed activity levels—typically, corresponding to an assumption of random, uniformly distributed white noise (UWN) data. Often this assumption is not justified as demonstrated by Fig. 2. In this example, the divider power based on switch-level simulation of an extracted layout varies by more than a factor of two as input data statistics change. In the following sections, we propose architectural power analysis techniques that are sensitive to activity.

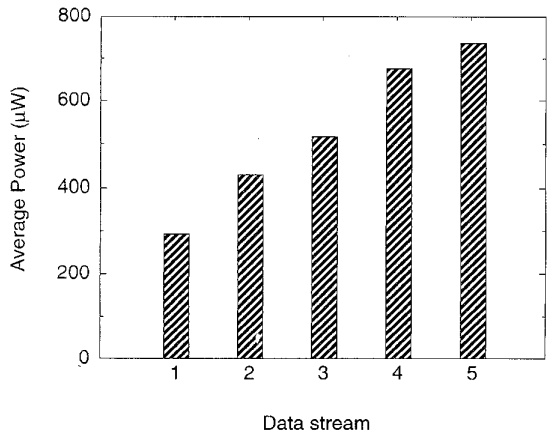


Fig. 2. Divider power from switch-level simulation of various data streams.

III. Datapath and Memory Modeling

Datapath and *data* memory components perform and store the results of the numerical computations required by an algorithm. In previous papers, we have described architectural power analysis techniques for these elements in detail [3], [14], [15]. This section will briefly review the principal results of those papers.

The idea is to produce a black-box model of the capacitance switched in each module for various types of input activity. If desired, these capacitance estimates can be converted to an equivalent energy, $E = CV^2$, or power, $P = CV^2f$. The models must take into account not only the complexity of the module being characterized, but also the switching activity within the module.

Complexity is handled by allowing the library designer to specify appropriate complexity parameters for each module. These parameters can then be used in a capacitance model to describe precisely how the physical capacitance of each module should scale with its "size" or complexity. For instance, the capacitance model for a ripple-carry adder is given by

$$C_T = C_{eff}N \tag{1}$$

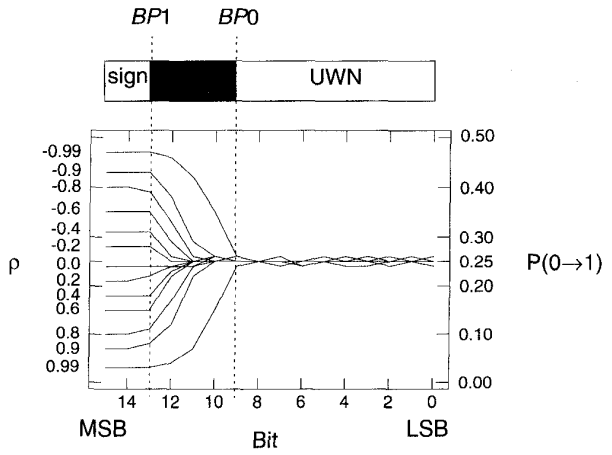


Fig. 3. Bit transition activity for data streams with varying temporal correlation.

where N is the word length and C_{eff} is a capacitive coefficient describing the effective (activity-dependent) capacitance switched per bit. If necessary the library designer can specify more complex expressions. For a typical SRAM

$$C_T = C_0 + C_1 W + C_2 N + C_3 WN \tag{2}$$

where W is the number of rows in the currently active block and N is the number of columns. The dominant $C_3 WN$ term stems from charging and discharging bit lines in the cell array. The remaining terms pertain to the row/column decoders, the word line drivers, and the sense amps. The equation can be expressed equally well in vector notation

$$C_T = \mathbf{C}_{eff} \cdot \mathbf{N}. \tag{3}$$

Here, the scalar capacitive coefficient and complexity parameters of (2) have simply been replaced by vectors

$$\mathbf{C}_{eff} = [C_0 \quad C_1 \quad C_2 \quad C_3]^T$$

and

$$\mathbf{N} = [1 \quad W \quad N \quad WN]^T.$$

The model also accounts for activity as well as complexity. Instead of having a single C_{eff} coefficient per module based on a white noise activity assumption, the model employs several capacitive coefficients for each module, corresponding to different input activity types. For fixed-point two's-complement data, we use a dual bit type (DBT) model that accounts for two classes of bits: sign and data. Fig. 3 confirms that two's-complement data is, indeed, characterized by two distinct activity regions. The data bits (LSB's) exhibit the activity of uniform white noise (UWN), while sign bit activity, in contrast, depends on the exact sequence of sign transitions. The likelihood of sign toggling is characterized by the temporal data stream correlation $\rho = \mathrm{cov}(X_{t-1}, X_t)/\sigma^2$.

The DBT method accounts for these differing activities by using a separate capacitive coefficient for each transition type. The data region uses a single UWN capacitive coefficient, C_{UU}. In contrast, characterizing all possible sign transitions requires several coefficients of the form, C_{SS}, where S can be positive or negative: C_{++}, C_{+-}, C_{-+}, and C_{--}. A module that operates on two or more input data streams requires additional capacitive coefficients. Each coefficient can be either a scalar or a vector depending on the number of terms in the capacitance model: for the above SRAM $\mathbf{C}_{\mathrm{UU}} = [C_{0,\mathrm{UU}} \quad C_{1,\mathrm{UU}} \quad C_{2,\mathrm{UU}} \quad C_{3,\mathrm{UU}}]^T$.

Capacitive coefficient values are derived for each module during a *library characterization* step. This is a one-time process, not required during power analysis, but instead performed whenever a new cell is added to the library. *Pattern generation* is the first step in the three-stage characterization process. During this phase, white noise and sign input patterns are generated. Next, *simulation* is used to measure the capacitance switched for these patterns. In order to characterize the influence of complexity, as well as activity, the module may be characterized for several complexity parameter values (e.g., word length, storage capacity, etc.). Finally, during *coefficient extraction*, the capacitance models are fit to the simulated capacitance data to produce a set of "best fit" capacitive coefficients.

The power analysis process itself consists of decomposing the modules into white noise and sign regions and then estimating the effective capacitance switched within each region. The size of the regions depends on the positions of the model breakpoints ($BP0$ and $BP1$) in Fig. 3. Analytical expressions were derived in [15] that express the breakpoints as a function of data stream statistics such as mean (μ), variance (σ^2), and correlation (ρ)

$$BP1 = \log_2(|\mu| + 3\sigma) \tag{4}$$

$$BP0 = \log_2 \sigma + \Delta BP0 \tag{5}$$

$$\Delta BP0 = \log_2 \left[\sqrt{1-\rho^2} + \frac{|\rho|}{8} \right]. \tag{6}$$

This information can be used to calculate the effective capacitance switched in each region during a typical module access

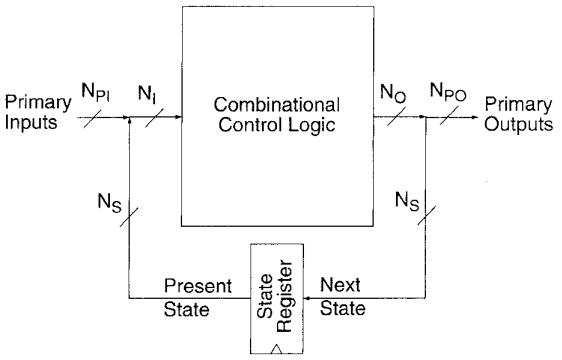
$$C_T = \frac{N_{\mathrm{U}}}{N} [\mathbf{C}_{\mathrm{UU}} \cdot \mathbf{N}] + \frac{N_{\mathrm{S}}}{N} \left[\sum_{\mathrm{SS} \in \binom{++,\ +-}{-+,\ --}} \mathrm{P}(\mathrm{SS}) \mathbf{C}_{\mathrm{SS}} \cdot \mathbf{N} \right] \tag{7}$$

where N_{U}/N and N_{S}/N are the fraction of UWN and sign bits, respectively, and $\mathrm{P}(\mathrm{SS})$ represents the probability of the various sign transitions. These probabilities along with the mean, variance, and correlation statistics can be measured during a functional simulation of the candidate architecture. Since this can be a register-transfer level (RTL) simulation, the required activity statistics for the entire chip can be gathered rapidly and efficiently.

Results have shown that this method can be used to produce RT-level power estimates within 15% of switch-level simulations based on circuits extracted from layout. Thus, it is possible to accurately model datapath and memory power consumption at the architecture level by accounting for the impact of both activity and complexity on power. The DBT

Input Table: Present State	IN	Output Table: Next State	OUT
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

(a)



(b)

Fig. 4. Behavioral (a) and structural (b) descriptions of T flip-flop FSM.

model is appropriate for fixed-point two's-complement data; other representations might require new activity models.

It should also be clarified that in this section "memory" has referred to storage elements containing numerical *data*. Of course, memory can also contain control path information, such as instructions. In that case, it is necessary to characterize the memories under a different activity model. In particular, the activity model described in the next section is more appropriate for memories containing instructions or control information.

IV. Control Path 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* as shown in Fig. 4(a). Controllers are often implemented using the finite state machine (FSM) structure of Fig. 4(b). This structure uses combinational logic to generate the next state and the primary outputs given the present state and the primary inputs. 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 a description of the state machine to be realized, say, in the form of a control table. It is also possible to envision power estimation based solely on the FSM behavior; however, behavioral power estimation [16]–[19], though useful for rough preliminary predictions, necessarily offers fairly limited accuracy.

Even at the architecture level, accurately estimating controller power is a nontrivial task. In the datapath case, the circuitry of library elements (such as adders and multipliers) is 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.

Since it is impractical to characterize each class of controller for all possible control table contents we, instead, characterize for random control tables. This results in a fixed, average 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. Since the method accounts explicitly for the effect of activity on power consumption we refer to it as the Activity-Based Control, or ABC, model.

The discussion of controller power modeling will be divided into two parts. Section IV-A will describe model parameters that influence power regardless of implementation style, while Section IV-B will present target-specific power models. This will be followed by Section IV-C which will discuss techniques for library characterization and Section IV-D which will review the controller power analysis method being proposed here.

A. Target-Independent Parameters

Two classes of parameters influence controller power regardless of the target implementation style: complexity parameters and activity parameters. First, the complexity (or size) of a controller directly influences its physical capacitance and, therefore, its power consumption. Fig. 4 suggests that combinational logic block complexity 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 input-plane complexity. Section VI-A describes techniques for estimating 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*

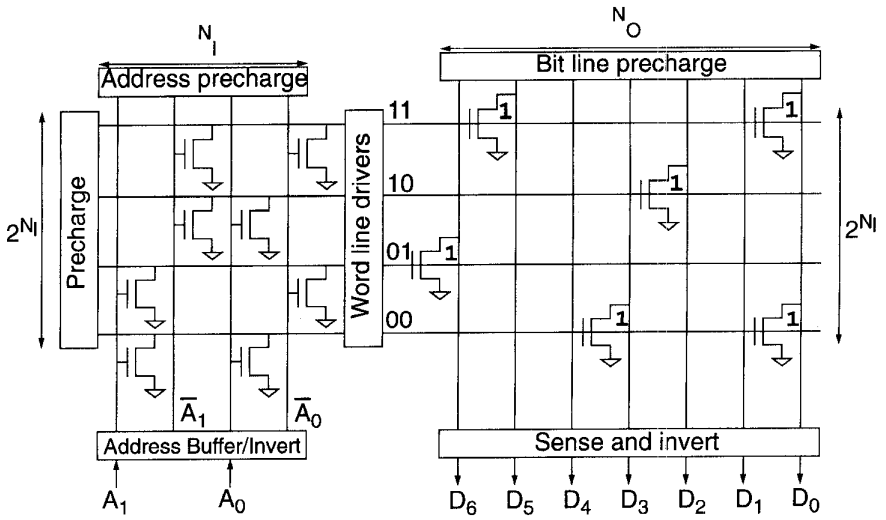


Fig. 5. Basic structure of prototype ROM (4 × 7 in this example).

activity: (α_O, P_O). Section VI-B will describe techniques for acquiring the necessary activity parameters through functional simulation.

B. Target-Specific Capacitance Models

The ABC method allows the user to specify a capacitance model that reflects how the effective capacitance of a particular controller class scales with changes in complexity and activity. This section illustrates how to construct a capacitance model using 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.

1) ROM-Based Controllers: The exact form of a ROM capacitance model can vary from one implementation to another. To address this issue, we use a library-based approach, allowing the user to define a unique capacitance model for each implementation in the library. If the ROM structure is not known, the model can be based on measurements taken from data books or previous implementations. In this example, we will use the ROM structure of Fig. 5. The average capacitance switched during a single access is given by

$$C_T = C_0 + C_1 N_I 2^{N_I} + C_2 P_O N_O 2^{N_I} + C_3 P_O N_O + C_4 N_O. \quad (8)$$

The terms in this expression relate to power consumed in the input plane (address decoding) and the output plane (bit lines). We first analyze the complexity and activity of the input plane. The complexity is proportional to the product of the number of columns and rows in the input plane: $N_I 2^{N_I}$. Since both true and complement address lines are present, the input-plane 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. Thus, no explicit activity factor is present in the $C_1 N_I 2^{N_I}$ term. It might seem that a separate N_I term would be needed to model the address buffers, but remember that the load being driven by these buffers is proportional to 2^{N_I}. Thus, the single $C_1 N_I 2^{N_I}$ term properly models both the address buffers and decoders.

In the output plane, the power consumption 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.

Combining terms yields the ROM capacitance model of (8), where C_0, C_1, C_2, C_3, and C_4 are capacitive coefficients dependent on the exact circuitry and technology used by the ROM. These coefficients are extracted through a library characterization process that will be described in Section IV-C.

As a means of validation, a comparison was made between a switch-level simulation of the ROM (using IRSIM-CAP) and the ABC model after characterization for a 1.2 μm technology using random control table contents. IRSIM-CAP [14] is a modified version of the switch-level simulator IRSIM [20] with improved capacitance measurement capabilities. The simulations were performed for random input streams with distributions chosen to exercise a variety of input and output activity levels. As with all IRSIM-CAP results described in this paper, the simulations were allowed to continue until the average capacitance switched satisfied a convergence criteria of 5%. This strategy allows us to handle sequential circuits with reasonable accuracy.

For this example, the results of the validation process are shown in Fig. 6. The ABC model exhibits an rms estimation error (relative to IRSIM-CAP) of about 2.5% and a maximum error of 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 (such as the ABC technique)

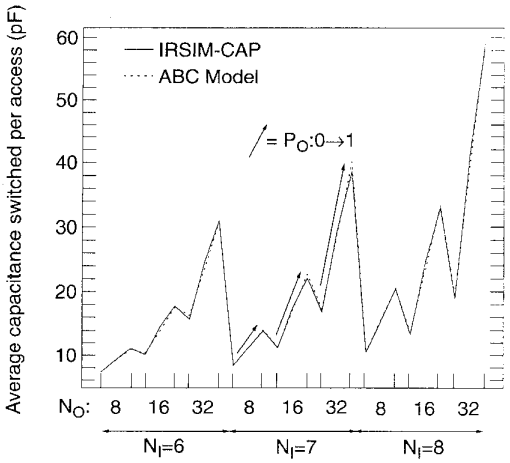


Fig. 6. ROM-based controller: IRSIM-CAP versus ABC model.

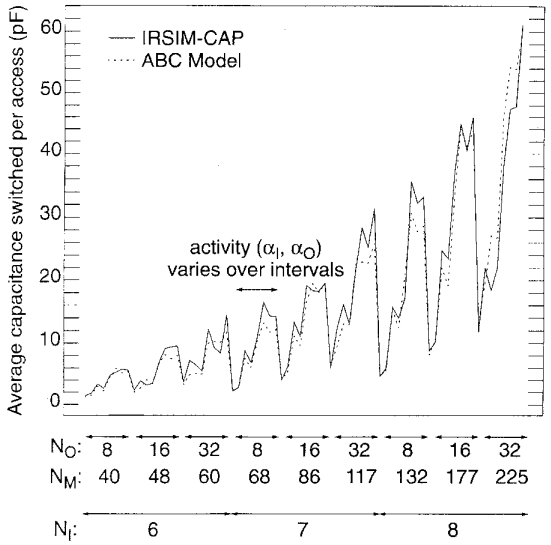


Fig. 7. Random logic controller: IRSIM-CAP versus ABC model.

which accounts for activity. It is also of interest to observe the magnitude of errors that can be expected for nonrandom control table contents. Section VII-B will show a controller with highly redundant (nonrandom) control table entries for which the power for three implementations differs on average by 12.6% from the ABC prediction and at most by 29%.

2) PLA-Based Controllers: Conceptually, the structure of a PLA is quite similar to that of a ROM. The principal 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 min-terms, 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

$$C_T = C_0\alpha_I N_I N_M + C_1 P_O N_O N_M + C_2 P_O N_O + C_3 N_O N_M + C_4 N_M. \quad (9)$$

We can compare this model to the ROM expression of (8). As expected, 2^{N_I} has given way to N_M. Also, since this PLA happens to use static input decoding, an α_I term has been added to model the effect of input transitions 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 gives rise to the activity-independent terms: $C_3 N_O N_M$ and $C_4 N_M$. The model resulting from characterization of the PLA (in a 1.2 μm technology) has an rms and maximum error of 2.5% and 6.3%, respectively.

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

$$C_T = C_0\alpha_I N_I N_M + C_1\alpha_O N_O N_M. \quad (10)$$

This expression contains two components—one relating to the input plane capacitance and the other relating to the output plane. Since this example is based on static logic, the appropriate activity measures are α_I and α_O, respectively. For the input plane, the complexity is given by the product of the number of inputs to that plane N_I and the number of outputs that 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. Since the exact equation depends on the synthesis tools being used, the library maintainer is free to tune the model as needed.

The two capacitive coefficients are derived during a characterization phase and will be a function of the standard cell library being used and, to some extent, the logic minimization, placement, and routing tools being applied. A comparison of the model to switch-level simulations for a 1.2 μm cell library is shown in Fig. 7. The results are for control tables with random contents that have been minimized using espresso [21] and synthesized using MIS [22] and the Lager IV silicon assembly system [23]. While the agreement is not as good as the ROM and PLA models, the rms error is still an acceptable 15.1%.

C. Characterization Method

Before the controller capacitance models can be used, circuit- and technology-dependent values of 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.

1) Pattern Generation: In the first phase, two distinct sets of data patterns are generated: the patterns stored in the control

TABLE I
"ADDRESS/DATA" PATTERNS FOR GENERATING DESIRED CONTROLLER ACTIVITIES

Label	Address A[7:2]	Map A[1:0]	Data
$A(0,0)$	000000	00	00000000
$A(0,\frac{1}{2})$		01	01010101
$A(0,1)$		11	11111111
$A(\frac{1}{2},0)$	010101	00	00000000
$A(\frac{1}{2},\frac{1}{2})$		01	01010101
$A(\frac{1}{2},1)$		11	11111111
$A(1,0)$	111111	00	00000000
$A(1,\frac{1}{2})$		01	01010101
$A(1,1)$		11	11111111

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 reasonable 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 then exploit the don't-cares by using logic minimization algorithms (such as espresso [21]).

The second phase of pattern generation entails producing input streams that exercise the controller over a full range of activities from 0–100%. Three evenly spaced activity levels (0, 1/2, and 1) can be realized by correct sequencing of three basic patterns: $00\cdots00$, $01\cdots01$, and $11\cdots11$. The input activity can be controlled by using the patterns as input "addresses" to the combinational logic block of the controller. 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 exemplified in Table I, 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. In summary, 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, input and output activities can be controlled fairly independently. For instance, desired input and output *signal probabilities* can be chosen from nine 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 of the form $(\alpha_I, \alpha_O) \in \{0, 1/2, 1\} \times \{0, 1/2, 1\}$ 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 $\alpha_O = |\alpha_O^0 - \alpha_O^1|$.

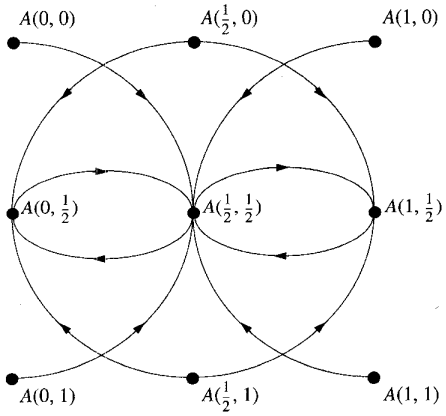


Fig. 8. Graphical representation of address sequences for (α_I, P_O) = (1/2, 1/2).

Desired address sequences can be represented graphically by associating the ordered activity pairs with a coordinate system as shown in Fig. 8, which shows address transitions corresponding to the activity pair (α_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 (9) from Section IV-B-2, input transition activity α_I, and output signal probability P_O are the activity parameters appropriate for the PLA-based controller. A full characterization of the PLA would include nine activity pairs$(\alpha_I, P_O) \in \{0, 1/2, 1\} \times \{0, 1/2, 1\}$. The standard cell controller requires nine as well, $(\alpha_I, \alpha_O) \in \{0, 1/2, 1\} \times \{0, 1/2, 1\}$, but 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 input transitions are averaged together to produce an aggregate switching capacitance for each I/O activity level and each implementation style.

Clearly, this strategy for selecting representative control table and input/output patterns is an approximation. It is difficult to precisely gauge the sensitivity of the model accuracy to pattern selection, since pathological cases can always be constructed. These cases would not say much, however, about what range of accuracies could be expected in practice. For this information, it is perhaps best to turn to the real controller designs that have been characterized with this method. These are discussed in Section VII where we will see, for example, that the maximum ABC prediction error for any of the implementations shown was 29%.

2) Simulation and Coefficient Extraction: During the next phase of characterization, a module corresponding to a given set of complexity parameters (e.g., number of inputs, minterms, and outputs) is synthesized and simulated for the data patterns generated by the aforementioned procedure. The simulation can be performed with either a circuit- or gate-level tool (e.g., SPICE [1], PowerMill [2], IRSIM [20]) depending on the time the designer wishes to allow for characterization and the accuracy desired. As mentioned above, the simulations described in this paper were all performed with IRSIM-CAP using a convergence criteria of 5%. Simulating to convergence,

rather than for a fixed number of steps, allows the method to handle sequential circuits with reasonable accuracy. The output of the simulation 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 $\mathbf{C}_{eff}$

$$\mathbf{C}_{sim} = \mathbf{P}\mathbf{C}_{eff} + \mathbf{e} \tag{11}$$

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 modeling error.

D. 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

$$C_T^{CL} = \mathbf{C}_{eff}^{CL} \cdot \mathbf{N} \tag{12}$$

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 module. 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

$$C_T^{reg} = \alpha_S C_0 N_S \tag{13}$$

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: $C_T|_{multi\text{-}cycle} = N_{CT} \cdot C_T|_{single\text{-}cycle}$.

E. Control Path Summary

The ABC model presented here explicitly accounts for activity and provides a general framework for modeling different controller structures. Three specific examples—based on ROM's, PLA's, and random logic—were presented in this section, but the same general techniques could be used to model other implementation styles. 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. The result is an accurate architecture-level estimate that reflects both the physical capacitance and the circuit activity of the controller being analyzed.

V. Interconnect Modeling

The final class of component in a typical chip is interconnect. This section addresses the problem of estimating the power consumed in charging and discharging interconnect capacitance.

A. Interconnect Activity

The activity of a wire depends on the type of signal that wire carries: data or control. The activity of a data signal can be described by the DBT model. Using this model, the total capacitance switched during a series of N_{DT} data transitions on a bus driven by static logic is given by

$$\text{static: } C_{DBT} = N_{DT}[\tfrac{1}{4}C_w N_{\text{U}} + P(+-)C_w N_{\text{S}}] \tag{14}$$

where 1/4 is the probability that a UWN bit transitions from zero to one, C_w is the physical capacitance of the wires, N_{U} is the number of UWN bits in the data model, N_{S} is the number of sign bits, and $P(+-)$ is the probability of a positive to negative transition between two successive samples in the data stream. For a control wire driven by static logic, the appropriate activity is given by the ABC parameter α, the fraction of bits in the control signal that transition. The total bus capacitance switched during N_{CT} control word transitions is

$$\text{static: } C_{ABC} = N_{CT}[\tfrac{1}{2}\alpha C_w N] \tag{15}$$

where α is the probability that a bit in the control word makes a transition, $1/2\alpha$ is the probability of a zero-to-one transition, and N is the number of bits required to represent all values that the control word can assume. The potentially strong correlation between the control signals making up the control word is handled by the fact that activities are estimated using a simulation-based approach as described in Section VI-B.

Equations (14) and (15) apply to static buses. Some buses, however, use precharged logic. In other words, each clock cycle they are precharged to V_{dd} and then the 0 bits discharge during an evaluation phase. For these precharged buses, the appropriate DBT effective capacitance equation is

$$\text{dynamic: } C_{DBT} = N_{clk}[\tfrac{1}{2}C_w N_{\text{U}} + P(+)C_w N_{\text{S}}] \tag{16}$$

where N_{clk} is the number of clock cycles being evaluated and $P(+)$ is the probability that a data sample is positive. Similarly, the ABC capacitance model becomes

$$\text{dynamic: } C_{ABC} = N_{clk}[(1-P)C_w N] \tag{17}$$

where P is the ABC signal probability parameter.

These expressions rely on the availability of DBT/ABC activity parameters and the physical wire capacitance. Section VI-B will describe how to derive activity parameters. The remainder of this section will cover physical capacitance estimation.

B. Physical Capacitance

In the ideal case, the physical capacitance of the interconnect network is known and can be used directly in architectural

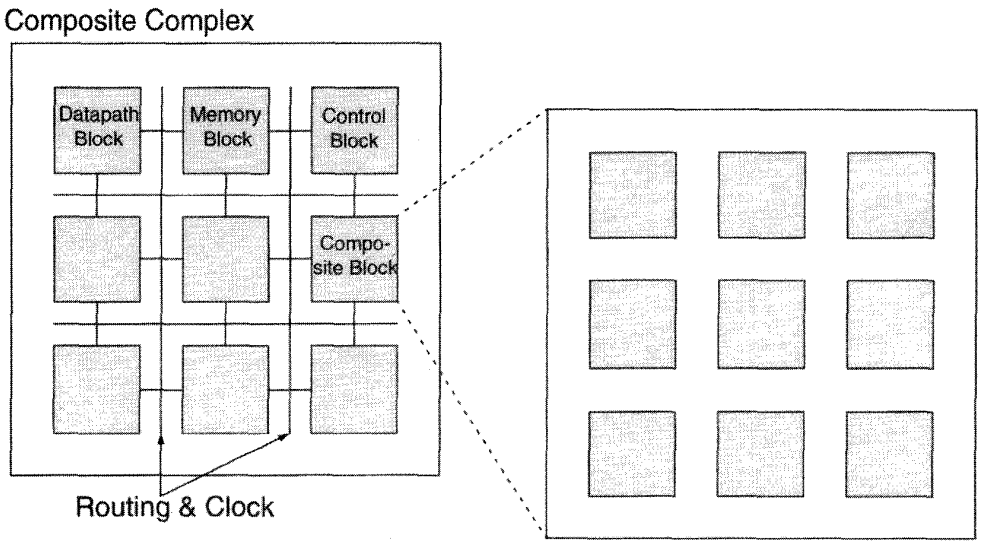


Fig. 9. Hierarchical structure of chip.

power estimation. More typically, however, analysis occurs prior to layout and wire capacitances must be estimated. Given a process technology and a set of design rules, it is possible to make fairly accurate estimates of average interconnect capacitance per unit length. Estimating interconnect length is difficult, however, and depends intimately on design-specific layout considerations.

The topic has received a good deal of attention in the past. Feuer [24] and Donath [25] have presented derivations based on Rent's Rule [12], which give rise to expressions for average lengths proportional to a power ($p < 1$) of design area. Sorkin [26] confirms this general rule noting that empirical data support average lengths proportional to the square root of the chip area (i.e., $p = 1/2$). Donath also makes use of hierarchical partitioning and placement. Our techniques rely heavily on this notion, as well as the empirical observations of Sorkin resulting in an intuitive, yet reasonably accurate, analysis strategy.

C. Wire Length Estimation Strategy

The input to this process is a hierarchical RTL description of the chip being analyzed, which may contain four classes of blocks (see Fig. 9). A *composite* block is used to introduce further hierarchy into the design. The other three block types are named for the type of structures within the block: datapath, memory, and control.

Estimating wire length at a given level of hierarchy requires area estimates for each component block. This suggests a recursive analysis scheme using a depth-first traversal strategy. Datapath, memory, and control blocks are treated as leaf cells and are handled by dedicated analysis routines. This hierarchical approach preserves partitioning clues supplied by the designer. This enables block-by-block estimates of wire length as opposed to a single chip-wide average. Sections V-D–G describe the analysis strategies for each of the four block types: composite, datapath, memory, and control.

D. Composite Blocks

Since we are interested in early estimates of wire length, precise placement and routing information is not available. Instead, we must predict what the average length is likely to be for a "good" placement. One option is to perform an early floorplanning step, but if this process is deemed too expensive there are other reasonable alternatives.

The approach advocated here relies on the empirical observation that the quality of a "good" placement often differs from a random placement by a constant factor, k [25], [26]. Since the average wire length for a random placement on a square array is 1/3 the side-length of the complex [27], the average length for a "good" placement is approximately

$$L = k\frac{\sqrt{A}}{3}. \tag{18}$$

The exact value of the k factor will depend on the characteristics of the placement and routing tools being used; however, an oft-quoted conversion factor is approximately 3/5. Since this value results in the formula $L = \sqrt{A}/5$, it has been called the "1/5 rule" in the literature [26].

The area of a composite complex is the sum the areas of the component blocks A_B and the area occupied by wires A_w

$$\begin{aligned} A &= A_w + A_B \\ &= A_w + \sum_{i \in \{\text{Blocks}\}} A_i. \end{aligned} \tag{19}$$

The component of area related to routing depends on the total number of wires in the complex (N_w), the average pitch at which they are routed (W_p), and their average length (L)

$$A_w = N_w W_p L. \tag{20}$$

Taken simultaneously, (18)–(20) result in a quadratic expression that can be solved to yield

$$L = \frac{k^2 N_w W_p + \sqrt{(k^2 N_w W_p)^2 + 36k^2 A_B}}{18}. \tag{21}$$

The above expressions can be used to recursively estimate the area and average interconnect length of composite complexes. For a parallel discussion of clock length estimation the reader is referred to [15].

E. Datapath Blocks

Since signals often flow through a datapath in a fairly linear fashion, datapath blocks often employ a one- rather than two-dimensional placement of modules, where the modules are formed by tiling up N individual bit-slices. Thus, interconnect length is proportional to the length, and not the area, of the datapath

$$L_x = k\frac{L_{DP}}{3}. \tag{22}$$

As before, we can use a k factor (specific to the datapath tiler) to convert from a random placement to a "good" one. The length of the datapath L_{DP} in turn is determined by module lengths and by the routing channels separating the modules

$$L_{DP} = L_R + \sum_{i\in\{\text{Blocks}\}} L_i. \tag{23}$$

Module lengths can be read from a library database, but routing channel lengths must be estimated from wiring pitch W_p and the number of vertical wiring tracks between modules, which is close to the total number of I/O terminals on all the modules

$$L_R = W_p \sum_{i\in\{\text{Modules}\}} N_{IO_i}. \tag{24}$$

In order to connect the terminals of two modules in the datapath, the wire must first be routed vertically from the terminals to the selected feedthrough. The length of these vertical wiring components is related to the width W_{BS} of a datapath bit slice

$$L_y = 2k\frac{W_{BS}}{3}. \tag{25}$$

Combining L_x and L_y gives us our average interconnect length $L = L_x + L_y$.

Finally, the area of the datapath, required for analyzing the next level up in the hierarchy, can be approximated as

$$\begin{aligned} A &= W_{DP}L_{DP} \\ &= NW_{BS}L_{DP}. \end{aligned} \tag{26}$$

All these calculations are based on the availability of raw area data for primitive modules (e.g., adders, shifters, etc.). Unlike power, however, the area of primitive modules is a deterministic function of their complexity parameters [15]. Thus, the area data can be entered into the hardware database from direct measurement of layout dimensions.

F. Memory Blocks

Since the power models for memories already account for internal wiring, interconnect analysis is not required, per se, for these modules. Area estimates are required, however, to analyze parent blocks in the hierarchy. As with datapath modules, deterministic formulas can be derived relating complexity parameters (i.e., storage capacity) to expected layout area [15].

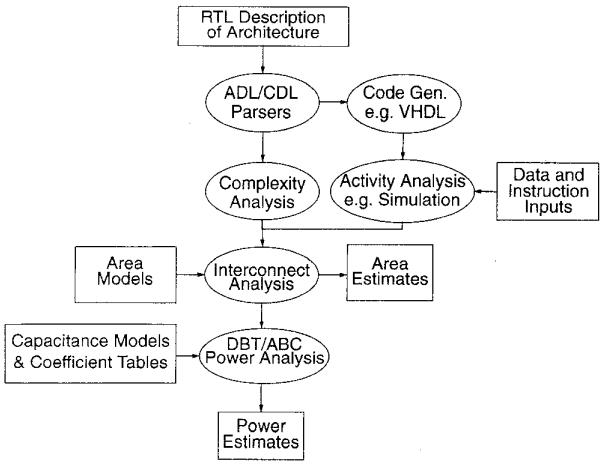


Fig. 10. Process flow for architectural power/area analysis.

G. Control Blocks

As for memory blocks, internal wiring is already included in the power models for control blocks when these blocks are characterized. So, once again, all that is required are area models for the three basic classes of controllers. The same model fitting techniques that were used for power can be abstracted to area modeling; however, since activity does not come into play, the area models will, in general, be more straightforward than the power models. Details are available in [15].

H. Interconnect Summary

To review, we propose a hierarchical approach to interconnect analysis that takes advantage of partitioning clues provided by the designer. This allows a distinction to be made between local and global interconnects. At each level of hierarchy, the length of the wires depended largely on the area of the block being analyzed. Once ascertained, the wire length estimates can be combined with DBT/ABC activity models to predict the power consumed in driving interconnect and clock networks.

The accuracy of these area and interconnect estimation techniques will be discussed more fully in Section VII. As a brief preview, the area estimate for the microprocessor example presented there is within 8% of the actual area. Moreover, the average interconnect and clock length estimates are off by 7% and 22%, respectively. Finally, the datapath-specific interconnect estimate errs by 17%.

VI. SPA: An Architectural Power/Area Analyzer

The previous sections have presented models and methods for analyzing the power consumption of the four basic classes of components: datapath, memory, control, and interconnect. These techniques have been integrated into an architectural power/area analysis environment called SPA. The SPA analysis flow consists of several phases as shown in Fig. 10. The primary input to SPA is a hierarchical RT-level description of the architecture under consideration. Currently, SPA uses

```
(inputs  (State_Type      state))
(outputs (Counter_Fn_Type a_reg)
         ...
         (State_Type next_state))

(output-table
;  state   | a_reg b_reg mux0 xi_reg yi_reg mux1 next_state
;  -----   | ----- ----- ---- ------ ------ ---- ----------
  (LDB    ) (NOP   LD    SEL0 NOP    NOP    SEL1 INV)
  (INV    ) (LD    NOP   SEL0 LD     NOP    SEL1 DIV0a)
  (DIV0a  ) (NOP   NOP   SEL1 NOP    LD     SEL0 DIV0b)
  (DIV0b  ) (NOP   NOP   SEL1 LD     NOP    SEL2 DIV1a)
  (DIV1a  ) (NOP   NOP   SEL1 NOP    LD     SEL0 DIV1b)
  (DIV1b  ) (NOP   NOP   SEL1 LD     NOP    SEL2 DIV2a)
  (DIV2a  ) (NOP   NOP   SEL1 NOP    LD     SEL0 DIV2b)
  (DIV2b  ) (NOP   NOP   SEL1 LD     NOP    SEL2 DIV3a)
  (DIV3a  ) (NOP   NOP   SEL1 NOP    LD     SEL0 DIV3b)
  (DIV3b  ) (NOP   NOP   SEL1 LD     NOP    SEL2 MULT)
  (MULT   ) (NOP   NOP   SEL1 LD     NOP    SEL1 LDB)
)
```

Fig. 11. Excerpt from CDL description of iterative hardware divider architecture.

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 (see Fig. 11). 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.

A. Complexity Analysis

The exact complexity parameters required and the method for calculating them differs depending on whether the entity falls under the DBT or ABC models. DBT complexity parameters, such as word length N, specify the "size" of datapath and memory elements. DBT complexity analysis consists of stepping through the various entities in the structural description and reading off the values specified by the user in the ADL "parameter" list of each module.

ABC complexity analysis reduces to ascertaining the "size" of control buses and modules. In order to maintain a high level of abstraction, all control signals are specified as enumerated types that take on symbolic, rather than binary, values. For example, if a control bus carries the function input to an ALU, the type of the bus might be defined as

$$\text{ALU\_TYPE} \equiv \{\text{ADD, SUB, SHIFT, CMP}\}. \tag{27}$$

The word length N (in bits) required to represent an enumerated type T which can take on $|T|$ different values is

$$N = \lceil \log_2 |T| \rceil. \tag{28}$$

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

Two important complexity parameters for a control module are the number of input bits N_I and the number of output bits N_O for the combinational logic block of each controller. This block may take many control buses as input and may feed multiple control buses at its output. The total input and output widths are calculated by summing individual control field widths

$$\begin{aligned} N_{\text{I/O}} &= \sum_{i \in \{\text{I/Os}\}} N_{\text{I/O}_i} \\ &= \sum_{i \in \{\text{I/Os}\}} \lceil \log_2 |\text{I/O\_TYPE}_i| \rceil. \end{aligned} \tag{29}$$

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 probably not available at this stage of design. It is more likely that the control table is specified in terms of symbolic values as shown in Fig. 11. We can approximate N_M by assuming some binary mapping (e.g., random), performing logic minimization (e.g., using espresso [21]), and using the number of min-terms in the minimized table as an estimate of N_M.

In summary, the complexity parameters can be determined from the architectural description of the design in question. The DBT parameters can be read directly from user specified parameter lists and the ABC parameters can be computed using a few simple expressions. Finally, the user is free to add custom complexity parameters to individual modules when appropriate as long as their values are specified when instantiating the module.

B. Activity Analysis

The next step in the process is to derive the ABC activity statistics for the design. This is accomplished by 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 the buses and modules in the VHDL description accumulate activity statistics during simulation. If the chip requires data or instruction inputs, these must be supplied by the user who is, therefore, able to characterize the power consumption for various input patterns and operating conditions. Since functional simulation is quite fast, run time is usually not an important concern and is often on the order of seconds or minutes.

The activity parameters for the DBT model are the word-level statistics discussed in Section III. These statistics include: mean (μ), variance (σ^2), correlation (ρ), sign transition probabilities, and access count (N_{DT}). For a synchronous system, this access count may be less than or equal to the number of clock cycles simulated; or, in the presence of block-level glitching, it may actually be higher.

ABC activity parameters can pertain either to a control bus or to a control module. For a control bus we must monitor the control word transition count, N_{CT}. We must also determine two other ABC activity parameters: signal probability P and transition activity α. If the binary encoding of symbols is known, exact values for P and α can be calculated during simulation by monitoring control word transitions. Otherwise, we can assume a random encoding (i.e., P = 1/2 and α = 1/2).

Control modules require two sets of (α, P) activity statistics: one for the combinational logic block input and one for the output. The input and output, in turn, may consist of several control buses bundled together. A transition of the controller input word leads to an increment of N_{CT} and is defined to occur when *any* of the component input signals transition, since this will initiate a reevaluation of the combinational logic block. 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

$$\alpha_{\mathrm{I/O}} = \frac{\sum_{i \in \{\mathrm{I/Os}\}} \frac{1}{2} N_{\mathrm{I/O}_i} N_{CT_i}}{N_{\mathrm{I/O}} N_{CT}} \tag{30}$$

where i refers to the individual control buses that make up the input/output words. For the case of input activity, this equation can be explained as follows: $1/2N_{I_i}$ 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}, to yield the number of bits within input i that toggle over the entire simulation. This is then summed across all of the individual inputs to yield the total number of bits that switch in the full input word during the simulation. Dividing this by the number of controller input transitions, N_{CT}, and the total number of input bits, N_I, results in the fraction of input bits that toggle during a typical input transition—that is, α_I.

To review, the ABC activity parameters depend on the encoding of the control words as binary values. If the user specifies the actual mapping, then simulated activities can be quite accurate; however, more likely the user will specify abstract symbolic types for the control signals and state machine control tables. In this case, a random assignment can be assumed in order to allow estimates of the control signal activities to be formed.

C. Power/Area Analysis

The complexity and activity parameters are then fed to the core area and power analysis routines along with the parsed architectural description. 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 in terms of the four classes of components. This points the designer to the most power-intensive portions of the implementation and provides useful guidance for further optimizations.

VII. Results

In this section, we present results gathered using the SPA power/area analysis tool. The first example is a Quadrature Mirror Filter which shows SPA's applicability to datapath-intensive applications. The second example is a control-intensive design—specifically, a finite-state machine that implements the global control function for a speech recognition front-end. 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 will also serve as a verification of SPA's efficacy for memory-intensive designs.

A. Datapath Intensive: A Quadrature Mirror Filter

SPA allows the designer to efficiently explore the design space, searching for low-power solutions. This example will demonstrate a design flow that employs SPA to minimize the power consumed by a Quadrature Mirror Filter (QMF) such as might be used in a subband coding algorithm [28]. A quadrature mirror filter takes an input signal and splits it into two bands: a low-pass band, $H_{LP}(\omega)$, and a high-pass band, $H_{HP}(\omega)$. The sample rate chosen for the filter would, in general, depend on the application. For the purposes of this example, we select a sample period of 0.3 μs or about 3.33 MHz. Four candidate architectures were explored using SPA.

The first version is a direct, naive implementation of the algorithm. The power and area predictions provided by SPA are shown in the "Initial" column of Table II. Four costly array multipliers are required to meet the throughput requirements of the algorithm at 5 V and this leads to a large die size of 95.9 mm^2.

In the second version of the design, the expensive array multipliers are replaced by shift-add operations, reducing the chip area to a more reasonable 17.2 mm^2. SPA reveals that this

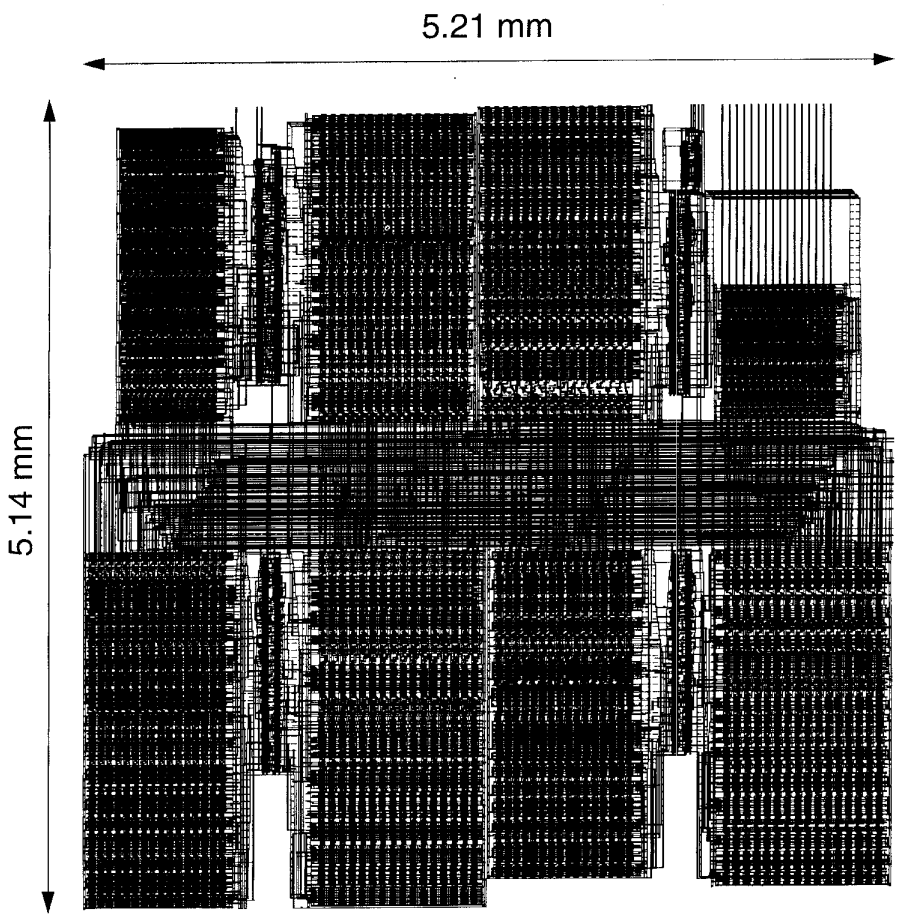


Fig. 12. Layout of retimed QMF filter design.

TABLE II
SPA POWER/AREA PREDICTIONS FOR QMF EXAMPLE

	Initial	Shift-Add	Retimed	Pipelined
V_{dd} (V)	5	5	1.5	1.25
Power (mW)	348.8	149.5	21.0	7.0
Area (mm^2)	95.9	17.2	24.4	115.4

version of the chip consumes 57% less power than the initial version, while at the same time occupying 82% less area.

A third implementation of the QMF example was generated by applying retiming, which reduces the critical path to 60 ns at 5 V. This allows us to lower the supply voltage of the implementation to about 1.5 V while still meeting the sampling rate constraint. Analysis using SPA shows a 7.1× reduction in power. Additional hardware requirements, however, increase the implementation area from 17.2 mm^2 to 24.4 mm^2.

A fourth version of the filter can be generated by pipelining the algorithm enabling a fully parallel implementation. This further reduces the critical path and allows the voltage supply to be reduced to 1.25 V. SPA confirms an additional power reduction of 3× for an overall reduction (from version one to version four) of 50×. Interestingly enough voltage reduction accounts for only 46% of the power saved by going from the retimed to the pipelined design. Fully 54% of the power saved by pipelining can be attributed to a distributed architecture which preserves signal correlations and, thus, minimizes switching activity. SPA is able to model these effects, but traditional estimators based on white-noise activity models are not.

While the pipelined example at 7 mW consumes less power than the 21 mW retimed design, it is at the cost of a 4.7× area increase. As a result, the retimed example, which is still 17× lower power than the initial solution and requires only 24.4 mm^2 is probably a more desirable solution.

To verify that SPA provided accurate area and power estimates, this version of the filter has been synthesized (down to layout), extracted, and simulated with actual speech samples as inputs. The chip plot is shown in Fig. 12. The predicted area of 24.4 mm^2 is within 9% of the actual 26.8 mm^2 area. A comparison of the SPA power predictions to switch-level simulation using IRSIM-CAP is given in Fig. 13. The figure shows the average power consumed by the chip for data streams corresponding to increasing input signal powers. SPA's estimates are within 5% to 14% of IRSIM-CAP for all data streams. Estimates based on the white-noise model are also included. The white-noise estimates do not track signal statistics and, therefore, err by as much as 71% for some of the data streams.

By using SPA to compare four candidate architectures we were able to significantly reduce design time. The four filter

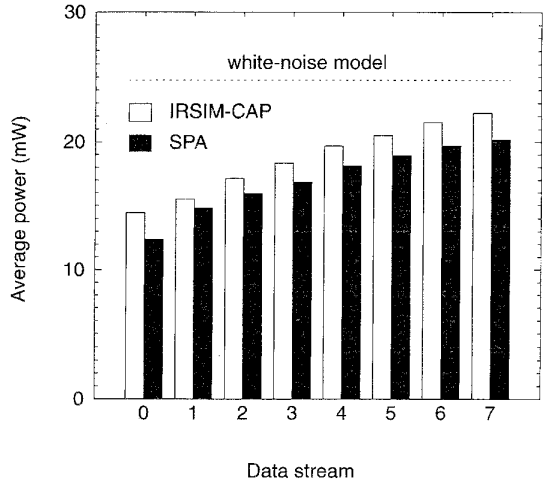


Fig. 13. Comparison of SPA to IRSIM-CAP for retimed QMF.

versions described here were synthesized using the HYPER high-level synthesis system [29] and analyzed with SPA in 5 min on a Sun SPARCstation 10. In contrast, laying out and simulating the retimed version took 3.2 h. Laying out and analyzing all four designs using low-level power analysis tools would have required 13 h or more.

B. Control Intensive: A Speech Recognition FSM

This example will demonstrate the use of SPA to aid in the design of a global finite state machine taken from the front-end of a speech recognition system [30]. Since this particular speech recognition 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 estimation models can handle nonrandom control table contents.

Fig. 14 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 1.5 V. The results show the area and average power for each candidate implementation. 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 Fig. 14 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 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. SPA still provides reasonable results, however, and more importantly it correctly tracks the influence of implementation style on area and power.

Based on the SPA results the designer could immediately eliminate the ROM-based solution. It consumes by far the most power and is also very large since it does not take advantage of redundancies in the control table. Both the PLA and standard cell implementations use logic minimization to reduce the

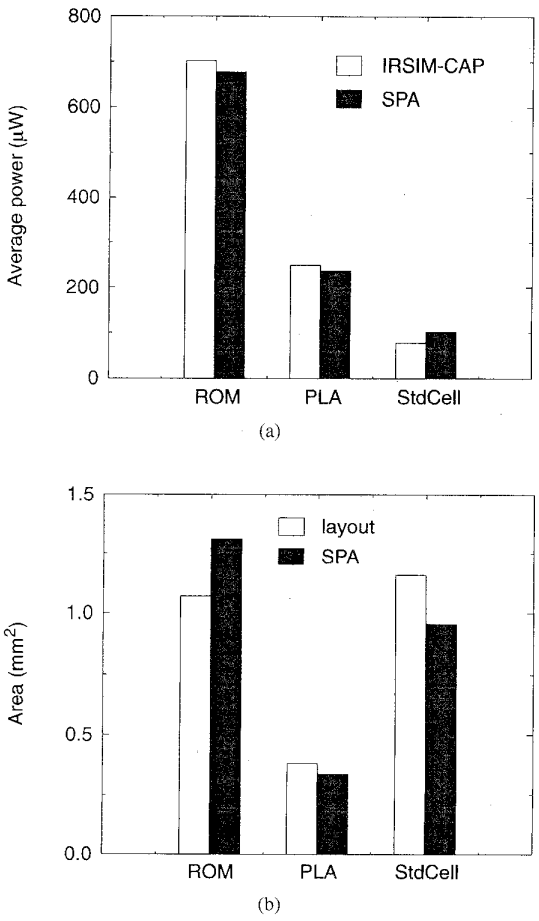


Fig. 14. Power and area results for three possible controller implementations. (a) Predicted versus actual power. (b) Predicted versus actual area.

amount of input decoding required. The higher regularity of the PLA allows it to be significantly smaller than the standard cell design, but the precharged nature of the unit leads to a higher activity (and power consumption) than the static logic standard cell implementation.

The time that can be saved by exploring these kinds of trade-offs at the architecture level is significant. For this example, on a Sun SPARCstation 10, SPA analyzed the area and power of the three controller implementations in about 2 min. In contrast, generating and extracting the layout for the three controllers took about 5 h. Simulation required another 30 min. Without high-level analysis tools it would not be practical to thoroughly explore all implementation possibilities.

C. Memory Intensive: A Programmable Microprocessor

The previous two examples have demonstrated SPA's accuracy and flexibility by showing how it can be used to analyze both datapath- and control-intensive applications. In this example, we tackle a programmable instruction set processor with on-chip instruction and data memories. Therefore, this case study demonstrates SPA's ability to handle memory-intensive, as well as datapath- and control-intensive, designs.

Fig. 15 depicts the architecture of the simple microcoded processor under consideration. The processor can execute the

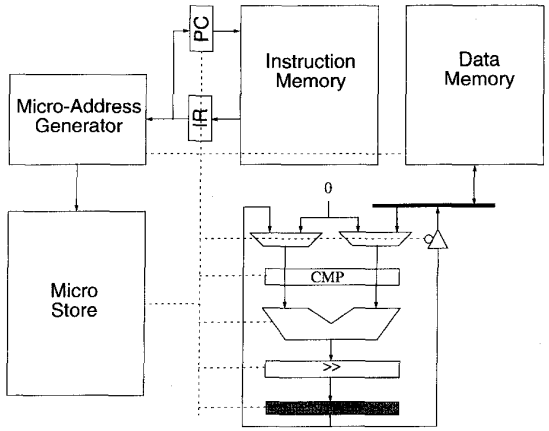


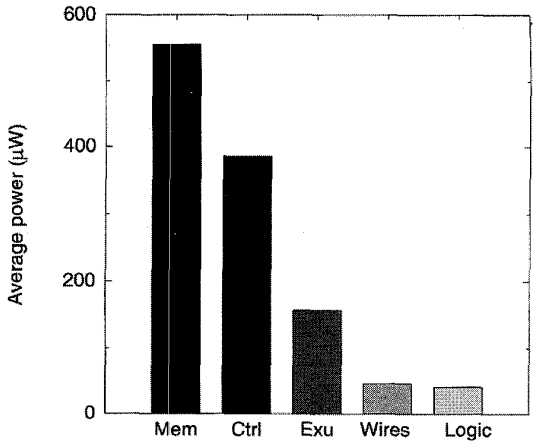
Fig. 15. Architecture of microcoded instruction set processor.

following nine basic instructions: NOP, LDA, STA, ADD, SHR, CMP, JMP, BLT, and BGE. Given the ADL and CDL descriptions of the architecture, SPA is able to make predictions regarding the area and power consumed by the proposed design while executing a given program on a given data stream. Fig. 16(a) contains the power breakdowns as predicted by SPA for a multiplication program running on the processor with a 1.5 V supply and a 10 MHz clock rate. The instruction and data memory accesses account for 47% of the total power consumption. This information could be used by the designer to select an appropriate focus for optimization efforts.

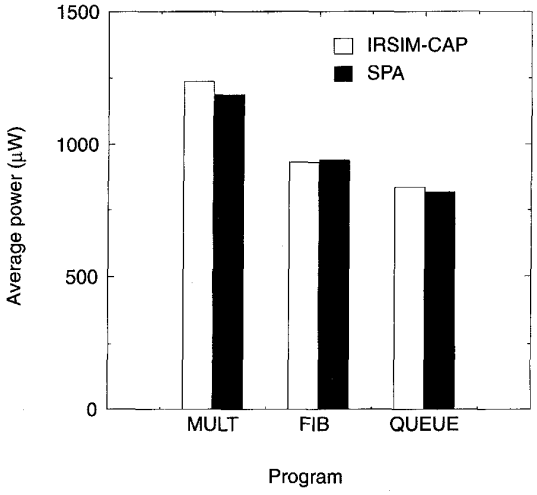
SPA can also be used to analyze the influence of instruction stream on average power consumption. This is made possible by the activity profiling approach used by SPA. As described in Section VI-B, SPA actually runs RT-level simulations profiling hardware activity for all instruction streams provided by the user. For example, Fig. 16(b) shows the average power while running three different programs: a multiplication program (MULT), a fibonacci sequence generator (FIB), and a circular queue (QUEUE). The power differs between these programs by as much as 38% making a prediction tool which can account for the influence of instruction statistics on power quite valuable.

In order to confirm the accuracy of these predictions, the processor has been implemented down to the layout level. The chip plot is shown in Fig. 17. The extracted layout was simulated at the switch level for the three programs and the results are included in Fig. 16(b). The error in the SPA estimates are: −4.1%, 0.88%, and −2.4%, respectively.

We can also use the layout data to verify the interconnect/area analysis strategy. Table III gives the area analysis for the design, showing that the estimated area is within 8% of the actual area. The estimated areas of the various blocks typically err by less than 20%. The micro sequencer is a notable exception. In the architectural description, this unit was defined as three distinct standard cell control blocks, while in the final implementation they were merged into a single block. This explains the overestimate and demonstrates the important point that architecture-level estimates are always subject to limitations imposed by an imperfect knowledge of the final implementation details.



(a)



(b)

Fig. 16. SPA power results for microprocessor. (a) Power breakdown by hardware class. (b) Power for different instruction/data streams.

TABLE III
PREDICTED VERSUS ACTUAL AREA FOR SIMPLE MICROPROCESSOR CHIP

Block	Actual (mm^2)	Predicted (mm^2)	Error (%)
Instruction memory	0.16	0.16	0%
Instruction regs	0.05	0.04	-20%
Micro sequencer	0.08	0.14	+75%
Micro store	0.13	0.11	-15%
Data memory	0.25	0.25	0%
Datapath	0.57	0.52	-9%
Wiring/overhead	0.90	0.74	-18%
Total	2.14	1.96	-8%

Based on the area estimates, the average interconnect and clock length are estimated at 554 μm and 2,174 μm. The actual lengths are 595 μm and 2,796 μm, implying estimation errors of −7% and −22%. To demonstrate that the interconnect

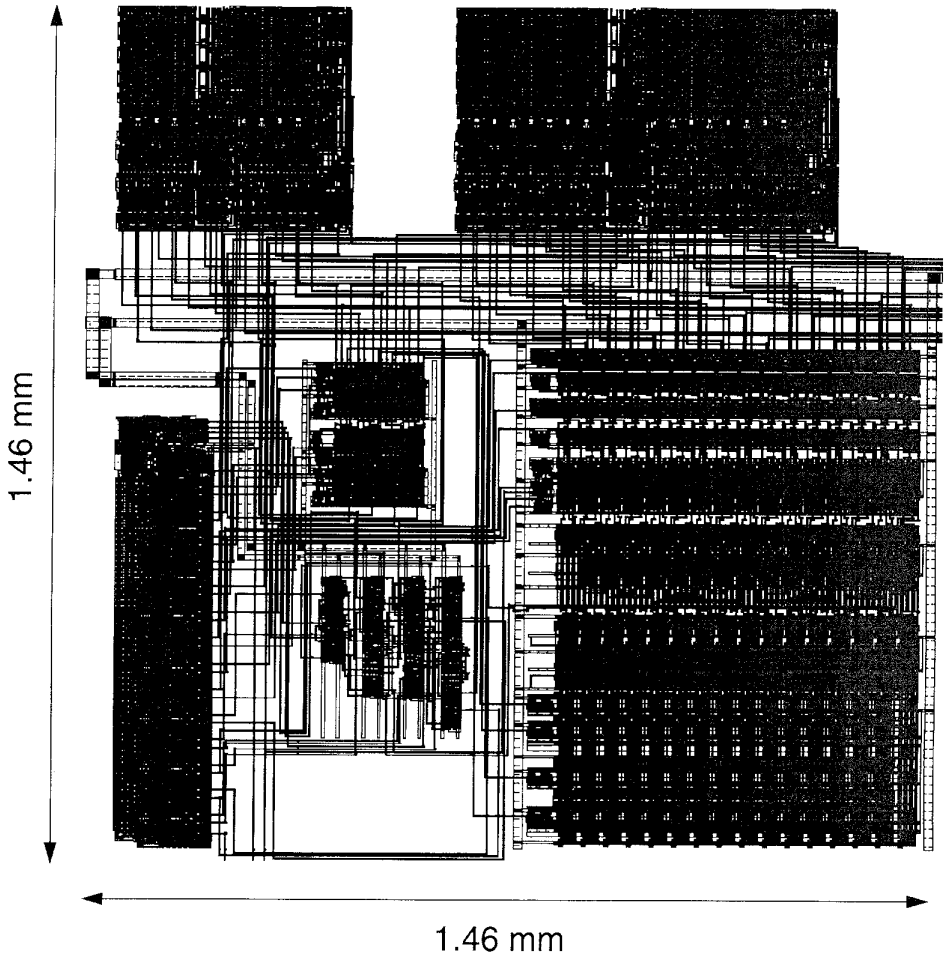


Fig. 17. Implementation of programmable microprocessor in 1.2 μm CMOS.

models are valid for other design examples, a hardware divider was analyzed as well. The predicted area was 4.68 mm^2 versus an actual implementation area of 4.40 mm^2—an error of 6%. The measured interconnect and clock wire lengths were 858 μm and 2,124 μm, while the predicted lengths were 856 μm and 2,861 μm for errors of -0.2% and 35%, respectively.

SPA's main advantage is that it requires only a high-level description of an architecture. For this example, the description took only about 2 h to produce, whereas layout required more than 25 h. Furthermore, SPA analyzed the chip power consumption in about 1 min (on a Sun SPARCstation 10), including parsing, simulation, and estimation times. In contrast, extraction and simulation of the layout consumed about 45 min.

VIII. Conclusion

This paper introduced a collection of techniques for analyzing the power consumption of chips at the architecture or RT level of abstraction. Power analysis for datapath and memory components was achieved using the Dual Bit Type (DBT) model. A new technique called the Activity-Based Control (ABC) model was introduced to handle power estimation for the control path. Both the DBT and ABC models improve upon contemporary architectural power analysis techniques such as gate-equivalent estimation by accurately reflecting the effect of activity, as well as complexity, on power consumption. Finally, a strategy for analyzing interconnect power consumption was presented. A useful side-effect of this analysis was a hierarchical area breakdown for the targeted design.

The above power and area analysis models have been implemented in a tool called SPA, which was used to gather results for several fairly realistic design examples. The three examples were constructed to fully exercise the power models and included datapath, control, and memory intensive designs as represented by a quadrature mirror filter, a speech recognition controller, and a microprocessor, respectively. Relative to switch-level power simulations of extracted layouts, these case studies revealed an average error of 9% with a standard deviation of 10%. Area estimation errors were of similar magnitude averaging 14% with a standard deviation of 6%.

Operating at the architecture level allows designers to obtain results much earlier in the design process and orders of magnitude faster than can be obtained using lower level analysis tools. This should allow designers to explore the area-speed-

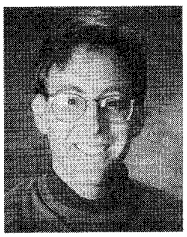
power trade-offs of several alternative architectures, while maintaining acceptable design times. Ideally, architecture-level analysis tools could be combined with existing gate- and circuit-level tools to form a seamless, integrated design environment that allows the user to consider optimizations at all levels ranging from system to circuit.

ACKNOWLEDGMENT

The authors wish to thank A. 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.

REFERENCES

[1] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Univ. California, Berkeley, Tech. Rep. ERL-M520, 1975.
[2] C. Huang, B. Zhang, A. Deng, and B. Swirski, "The design and implementation of PowerMill," in *Proc. Int. Symp. Low Power Design,* Dana Point, CA, Apr. 1995, pp. 105–110.
[3] P. Landman and J. Rabaey, "Architectural power analysis: The dual bit type method," *IEEE Trans. VLSI Syst.,* pp. 173–187, June 1995.
[4] A. Deng, Y. Shiau, and K. Loh, "Time domain current waveform simulation of CMOS circuits," in *Proc. Int. Conf. Computer-Aided Design,* 1988, pp. 208–211.
[5] M. A. Cirit, "Estimating dynamic power consumption of CMOS circuits," in *Proc. IEEE Int. Conf. Computer Aided Design,* Nov. 1987, pp. 534–537.
[6] F. Najm, I. Hajj, and P. Yang, "Probabilistic simulation for reliability analysis of CMOS VLSI circuits," *IEEE Trans. Computer-Aided Design,* vol. 9, pp. 439–450, Apr. 1990.
[7] F. Najm, "Transition density, a stochastic measure of activity in digital circuits," in *Proc. 28th Design Automation Conf.,* June 1991, pp. 644–649.
[8] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits," in *Proc. 29th Design Automation Conf.,* June 1992, pp. 253–259.
[9] C.-Y. Tsui, M. Pedram, and A. Despain, "Efficient estimation of dynamic power consumption under a real delay model," in *Proc. Int. Conf. Computer-Aided Design,* 1993, pp. 224–228.
[10] K. Muller-Glaser, K. Kirsch, and K. Neusinger, "Estimating essential design characteristics to support project planning for ASIC design management," in *Proc. IEEE Int. Conf. Computer-Aided Design,* Los Alamitos, CA, Nov. 1991, pp. 148–151.
[11] C. Svensson and D. Liu, "A power estimation tool and prospects of power savings in CMOS VLSI chips," in *Proc. Int. Workshop Low-Power Design,* Napa Valley, CA, Apr. 1994, pp. 171–176.
[12] B. Landman and R. Russo, "On a pin versus block relationship for partitions of logic graphs," *IEEE Trans. Computing,* vol. C-20, pp. 1469–1479, Dec. 1971.
[13] S. R. Powell and P. M. Chau, "Estimating power dissipation of VLSI signal processing chips: The PFA technique," *VLSI Signal Processing IV,* pp. 250–259, 1990.
[14] P. Landman and J. Rabaey, "Black-box capacitance models for architectural power analysis," in *Proc. Int. Workshop Low Power Design,* Napa Valley, CA, Apr. 1994, pp. 165–170.
[15] P. Landman, "Low-power architectural design methodologies," Ph.D. dissertation, Univ. of California, Berkeley, Aug. 1994.
[16] J. Ward *et al.,* "Figures of merit for VLSI implementations of digital signal processing algorithms," *Proc. IEE,* vol. 131, Part F, pp. 64–70, Feb. 1984.
[17] S. Powell and P. Chau, "A model for estimating power dissipation in a class of DSP VLSI chips," *IEEE Trans. Circuits Syst.,* vol. 38, pp. 646–650, June 1991.
[18] R. Mehra, "High-level power estimation and exploration," in *Proc. Int. Workshop Low Power Design,* Apr. 1994, pp. 197–202.
[19] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. W. Brodersen, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design,* vol. 14, pp. 12–31, Jan. 1995.
[20] A. Salz and M. Horowitz, "IRSIM: An incremental MOS switch-level simulator," in *Proc. 26th Design Automation Conf.,* 1989, pp. 173–178.
[21] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis.* Boston, MA: Kluwer Academic, 1984.
[22] L. Lavagno, S. Malik, R. Brayton, and A. Sangiovanni-Vincentelli, "MIS-MV: Optimization of multi-level logic with multiple-valued inputs," in *Proc. IEEE Int. Conf. Computer-Aided Design,* Santa Clara, CA, Nov. 1990, pp. 560–563.
[23] R. W. Brodersen, Ed., *Anatomy of a Silicon Compiler.* Kluwer Academic, 1992.
[24] M. Feuer, "Connectivity of random logic," *IEEE Trans. Comput.,* vol. C-31, pp. 29–33, Jan. 1982.
[25] W. Donath, "Placement and average interconnection lengths of computer logic," *IEEE Trans. Circuits Syst.* vol. CAS-26, pp. 272–277, Apr. 1979.
[26] G. Sorkin, "Asymptotically perfect trivial global routing: A stochastic analysis," *IEEE Trans. Computer-Aided Design,* vol. CAD-6, p. 820, 1987.
[27] B. Preas and M. Lorenzetti, Eds., *Physical Design Automation of VLSI Systems.* Menlo Park, CA: Benjamin/Cummings, 1988.
[28] N. Jayant and P. Noll, *Digital Coding of Waveforms.* Englewood Cliffs, NJ: Prentice-Hall (Signal Processing Series), 1984.
[29] J. M. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of datapath-intensive architectures," *IEEE Design Test of Comput.,* pp. 40–51, June 1991.
[30] S. Stoiber, "Low-power digital signal processing for speech recognition," Master's degree thesis, Univ. of California, Berkeley, Dec. 1994.



Paul E. Landman (S'92–M'95) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1989, 1991, and 1994, respectively. His research focused on low-power digital design techniques and tools with an emphasis on DSP applications.

After completing his dissertation, he joined the low-power design branch of the DSP R&D Center of Texas Instruments, Dallas. His initial research there focused on low-power algorithms, architectures, and circuits for portable video applications. Currently, he is participating in the development of a low-power programmable DSP targeted at wireless communication applications.

Dr. Landman is a National Science Foundation Fellowship recipient and is a member of Tau Beta Pi and Eta Kappa Nu.

Jan M. Rabaey (S'80–M'83–SM'92–F'95) received the E.E. and Ph.D. degrees in applied science from the Katholieke Universiteit Leuven, Belgium, in 1978 and 1983, respectively.

From 1983 to 1985, he was with the University of California, Berkeley, as a Visiting Research Engineer. From 1985 to 1987, he was a Research Manager with IMEC, Belgium, where he pioneered the development of the CATHEDRAL II synthesis system for digital signal processing. In 1987, he joined the faculty of the Electrical Engineering and Computer Science Department, University of California, where he is now a Professor. He has authored or coauthored more than 100 papers in the area of signal processing and design automation. His research interests include the exploration of architectures and algorithms for digital signal processing systems and their interaction. He is also active in various aspects of portable, distributed communications and computation systems, including low-power design, networking, and design applications.