VLSI DESIGN
1998, Vol. 7, No. 3, pp. 255–270
Reprints available directly from the publisher
Photocopying permitted by license only

Architectural Power Estimation Based on Behavior Level Profiling

SRINIVAS KATKOORI[a,†] and RANGA VEMURI[b,*]

[a]*University of South Florida, Department of Computer Science & Engineering, 4202 E. Fowler Avenue, ENB 118, Tampa FL 33620-5399;*
[b]*Laboratory for Digital Design Environments, Department of Electrical and Computer Engineering, 813 Rhodes Hall, Mail Location 30, University of Cincinnati, Cincinnati, Ohio 45221-0030*

High level synthesis is the process of generating register transfer (RT) level designs from behavioral specifications. High level synthesis systems have traditionally taken into account such constraints as area, clock period and throughput time. Many high level synthesis systems [1] permit generation of many alternative RT level designs meeting these constraints in a relatively short time. If it is possible to accurately estimate the power consumption of RT level designs, then a low power design from among these alternatives can be selected.

In this paper, we present an accurate power estimation technique for register transfer level designs generated by high level synthesis systems. The technique has four main aspects: (1) Each RT level component used in high level synthesis is characterized for average switched capacitance per input vector. This data is stored in the RT level component library. (2) Using user-specified stimuli, the given behavioral description is simulated and event activities of various operators and carriers are measured. Then, the behavioral specification is submitted to the synthesis system and a number of alternative RTL designs meeting speed, space and throughput rate constraints are generated. (3) Event activity of each component in an RT level design is estimated using the event activities measured at the time of behavior level profiling and the structure of the RTL design itself. (4) The event activities so obtained are then used to modulate the average switched capacitances of the respective RT level components to obtain an estimate the total switched capacitance of each component.

Detailed power estimation procedures for the three different parts of RTL designs, namely, data path, controller and interconnect are presented. Experimental results obtained from a variety of designs show that the power estimates are within 3% – 10% of the actual power measured by simulating the transistor level designs extracted from mask layouts.

Keywords: High level synthesis, power estimation, behavioral profiling, register transfer level designs, low power

*Corresponding author.
†This work was performed as part of the doctoral dissertation, when the author was at University of Cincinnati.

1. INTRODUCTION

Due to the increasing demand for portable applications and the rapidly growing complexity, power consumption has become one of the main issues in the realization of VLSI chips. There have been major efforts [2] to reduce the power consumption at all levels of abstraction in the design flow. Accurate power estimation techniques are the key to the success of these efforts. Although accurate power estimation is possible at the lower levels of abstraction, it is very time consuming. Hence, recently focus has shifted to the higher levels of abstraction including register transfer (RT) level and above [3]. In this paper, we present a power estimation technique for automatically synthesized RT level designs. This technique is based on behavior level profiling.

A high level synthesis system accepts a behavioral specification written in a hardware description language such as VHDL, a module library, and design constraints such as the area and delay constraints. The module library consists of RT level modules such as adders, multipliers, registers and multiplexors. The output of the synthesis system is a RT level design satisfying the user specified constraints. The synthesis time is usually quite small compared to logic synthesis or layout synthesis. Hence, it is possible to synthesize many constraint-satisfying RT level designs in a relatively short time.

RT level designs are composed of two interacting parts: *datapath* and *controller*. The datapath consists of execution units such as adders and multipliers, storage units such as registers and RAMs, and interconnect units such as multiplexors and buses. Since the structure of the design is known completely accurate power estimation is feasible. In addition, since the modules are at a sufficiently high level of abstraction such estimation should be time efficient. At the higher levels of abstraction such as the behavioral level, accurate power estimation is difficult due to the lack of sufficient implementation detail. On the other hand, at lower levels of abstraction, such as logic and layout levels, even though sufficient implementation detail is available, estimation time is discouraging. Hence, we are motivated to estimate power at the RT level of abstraction. For a given RT level design and for a given set of input vectors, we estimate the total capacitance switched in the design. We use "power" and "switched capacitance" synonymously. Our estimation technique is set in the context of a high level synthesis system known as the Profile-Driven Synthesis System (PDSS).

Our power estimation procedure requires the following inputs: (1) A module library characterized for the average intrinsic switched capacitance (ISC) per input vector. (2) Profile data for various carriers and operators in the data flow graph of the behavioral specification. This data is obtained by simulating the behavioral specification using user-specified stimuli. (3) A RT level design generated by the synthesis system. (4) Binding information of the operators and carriers in the data flow graph to the module instances in the RT level design.

High level synthesis process introduces certain RT level module instances such as temporary registers and multiplexors for which the profile data is not known since these modules have no direct correspondence with the operators and carriers at the behavior level. Profile data for such data path units is derived using the profile data at their inputs which in turn is obtained from the profile data measured at the behavior level. The switched capacitance for each module instance is estimated as the product of its profile data and its intrinsic switched capacitance obtained from the module library. The total switched capacitance in the datapath is the sum of estimated switched capacitances over all instances.

The switched capacitance estimation for the controller, assumed to be implemented as a PLA, is as follows: A parameterized PLA characterization table for average switched capacitance per clock cycle is obtained as explained in Section 5. Given the controller size, the switched capacitance for the controller is estimated by determining the closest point in the PLA table.

The power estimated for the entire design is the sum of the estimated switched capacitances of the datapath and the controller. Experimental results show that the power estimated in the RT level datapaths and controllers is within 15% of the actual power measured at the layout level.

Section 2 presents a brief survey of the power estimation techniques at architectural and other levels of abstraction. Section 3 discusses various issues involved in power estimation. Section 4 discusses the concept of behavioral profiling. Section 5 discusses the module library characterization and the PLA characterization for the average switched capacitance per unit vector. Section 6 discusses the power estimation technique. Section 7 presents the results obtained for several examples. Section 8 discusses the results and presents concluding remarks.

2. PREVIOUS WORK

Powell *et al.* [4] suggested Power Factor Approximation (PFA) method for characterizing each module in a module library consisting of functional blocks. The method provides different gate equivalent models for blocks such as multipliers, adders, etc. Each functional block is associated with a PFA proportionality constant and a hardware complexity constant. The PFA constant captures the intrinsic internal activity of the module. Purely random inputs are applied when deriving the PFA constant. The power dissipation in a chip is the sum of the power dissipation in all blocks of the chip. The power contributed by a block in the chip is simply the product of the above two constants and the block's activity frequency. The activity frequency of a functional block is the frequency at which the function is performed. In Powell *et al.* [5] present an algorithm level power dissipation model for a class of DSP algorithms known as MA-based (Multiply-Add) DSP Algorithms. The major sources of power dissipation in MA-DSP systems were identified to be memory operations, computations and I/O operations. Impact of the number of available processing elements, complexity of processing elements, memory organization and type of arithmetic on power dissipation was discussed. This model relates power dissipation to high level algorithmic and architectural parameters.

Chandrakasan *et al.* [6, 7] described a high-level synthesis system, HYPER-LP, which uses a variety of architectural and computational transformations to minimize power consumption in application-specific datapath-intensive CMOS circuits.

Landman *et al.* [8] presented a methodology for low-power design-space exploration at the architectural level. Black-box power models for the architectural-level components were generated [9] and used to estimate power while preserving the accuracy of the gate or circuit level estimation. The power analysis tool was set in the context of HYPER [10], a high level synthesis system. The key differences between our approach and Landman's approach are (1) our synthesis system, known as PDSS (Profile Driven Synthesis System) [11], is targeted towards control-dominated ASIC applications. The behavioral specifications can contain complex control constructs such nested loops, conditional and subprograms. On the other hand, HYPER primarily targets mostly straight-line DSP-style specifications. (2) Our approach is based solely on the behavioral profiling. Landman's estimation is based on behavioral profiling or RT level profiling. For large designs, with large set of inputs, the latter approach is time consuming and hence design space exploration is difficult. (3) Our characterization of the module library is based on purely random inputs, that is, Uniform White Noise (UWN) model. Landman, on the other hand, proposed DBT (Dual Bit Type) model to take into account the input activity. Our power dissipation model based on UWN model is simpler compared to Landman's and yet yields reasonably accurate estimates.

Renu *et al.* [12] proposed a behavior level power estimation technique based on a combination of analytical and stochastic methods. Based

on this, a design space exploration tool is presented which is used to examine the effect of different design steps such as transformations and algorithms. These techniques have also been implemented in HYPER synthesis environment [10].

Anand *et al.* [14] present a behavioral synthesis system known as Genesis, for synthesizing low power datapath intensive CMOS circuits. During the allocation phase, (1) the physical capacitance is reduced by minimizing the number of functional modules, registers and multiplexors; and (2) the transition activity for a given module is reduced by selecting a proper sequence of operations for that module. The controller is optimized so as to generate control signals which will reduce the transition activity in the datapath. This is achieved by introducing don't-cares in the state table of the controller. If a datapath module is idle for a particular cycle, then the control signal driving that module is assigned a don't-care, thus avoiding unnecessary clocking of the module. In Anand *et al.* [15] present a simulation-based method to measure intra- and inter-iteration effects of hardware sharing on switched capacitance. During the simulation, information is gathered which is used to formulate allocation as an ILP problem with the total switched capacitance in the datapath as the objective function. The solution to the problem yields optimal allocation for the given model.

A detailed discussion about power consumption in CMOS digital designs can be found in [16]. Techniques for low power operation are presented which use the lowest possible supply voltage coupled with architectural, logic, circuit, and technology optimizations. An excellent literature survey on the power estimation techniques at the logic and lower levels of abstraction can be found in [17].

In [11, 18], we have proposed a behavior level profiling based technique to estimate switching activity and switching capacitance in a design. The estimation is carried out in the scheduling and performance analysis phase of the synthesis. For a given input specification, various schedules can be generated satisfying the user given constraints. The schedule with least estimated switching capacitance is further synthesized. The estimation technique adopts analytical approach at the design level and statistical approach for the module library characterization. One of the drawbacks of the approach is that the interconnect estimation is somewhat inaccurate at the scheduling level, resulting in inaccurate power estimation. In the present work, the estimation is at the RT Level and is based on the behavior level profiling of the input specification. Since the interconnect structure is known completely, power estimation in the interconnect is more accurate compared to that obtained at the end of the scheduling step. In the present approach, the error in power estimator is in the range of 3% – 10%.

3. ISSUES IN POWER ESTIMATION

In a CMOS digital circuit, the power consumed is given by the following equations [19, 16]:

$$P_{\text{consumed}} = P_{\text{switching}} + P_{\text{sc}} + P_{\text{leakage}}$$

$$P_{\text{switching}} = \sum_i \frac{1}{2} * C_i * V_{\text{supply}}^2 * f_i$$

$$P_{\text{sc}} = I_{\text{sc}} * V_{\text{supply}}$$

$$P_{\text{leakage}} = I_{\text{leakage}} * V_{\text{supply}}$$

$P_{\text{switching}}$ is known as the switching component of power consumption which arises due to charging a node i with a load capacitance of C_i and which is clocked at a frequency, f_i. P_{sc}, the short-circuit component arises when the PMOS and NMOS transistors are switching simultaneously resulting in a short-circuit path from the voltage supply to ground. For a very short period of time, current is drawn from the voltage supply to the ground which results in power dissipation. P_{leakage} is due to the leakage current, I_{leakage}, which arises due to substrate injection and subthreshold effects.

The dominant term is $P_{\text{switching}}$. This term is dependent on the architectural parameters and is relatively amenable for estimation at higher levels

of abstraction. It is well-known that the static power consumption in digital CMOS circuits is negligible compared to the dynamic power consumption. Hence $P_{leakage}$, which is static in nature is negligible. P_{sc} can be kept within 15–20% of $P_{switching}$ [20] by proper design methodology. Thus, it is sufficient to estimate $P_{switching}$ to estimate the average power consumed by a design.

Dynamic power consumption is strongly dependent on the stream of inputs applied to the circuit [17]. Without any information about the input stream, it is impossible to accurately estimate the power consumption of a design. Thus, for a power estimation technique it is necessary to provide actual or statistical information about the input behavior.

Different power estimation techniques make different assumptions about the input vectors. These techniques are based on statistical, stochastic, probabilistic, or analytical approaches. For any technique two broad steps can be identified: (1) Characterization of the circuit components for power and storing relevant information about the components in the form of statistical models, parameterized tables, equations, etc. This is usually done only once for all the components used in the circuit. (2) Estimation of average power for a given design by combining the input behavior information specific to a design with the module library information using a statistical, stochastic probabilistic or analytical approach or a mix of these approaches.

In our approach for power estimation at RT level, the input vector behavior is indirectly specified by the user by providing a sequence of *typical* input vectors, known as the *profiling stimuli.* These vectors denote typical usage of the digital system being synthesized. These vectors are used to simulate the behavior level specification during which event activities of various behavior level operators and carriers are monitored and recorded. Collectively this information is known as the *profile data.*

For a given set of inputs to a digital circuit, the capacitance switched in the circuit is a measure of the power consumed by the circuit. We adopt this indirect approach for power estimation. Thus, in this paper, we use "power" and "switched capacitance" synonymously.

The module library is precharacterized for average switched capacitance per input vector as explained in detail in Section 5. RT level designs contain three subunits: datapath, controller, and interconnect. Detailed procedures to estimate the switched capacitances in each of these units are presented in Section 6.

4. BEHAVIORAL PROFILING

The concept of profiling a given program to gather various statistics is not new. A well-known technique for measuring program performance is to insert monitoring code into the program and execute the modified program. Program profiling counts the number of times each basic block is executed and the number of times each control-flow path is traversed. Profiling is widely used to measure instruction set utilization, identify program bottlenecks and estimate program execution times for code optimization [25, 26, 27, 28, 29]. Techniques to insert monitoring code to optimally and efficiently profile programs exist in the literature [30, 31, 32].

Behavioral level profiling is similar to program profiling. For profile data to make sense in case of high level synthesis, one needs to understand the correspondence between the constructs (variables, operations, loops etc.) in the behavior representation to elements in the resulting hardware. Understanding this correspondence helps in determining the data to be gathered during profiling. The profiling strategy is mainly dependent on how different synthesis tasks go about synthesizing the target design.

Consider the behavior description written in VHDL as shown in Figure 1. One possible RTL data path synthesized from the specification is as shown in Figure 2. The correspondences between elements of the specification and the elements of

```
(1)     ENTITY toy IS
(2)        PORT(a, b : IN INTEGER;
(3)             c : OUT INTEGER);
(4)     END toy;
(5)
(6)     ARCHITECTURE foo OF toy IS
(7)     BEGIN
(8)       p: PROCESS(a, b)
(10)      VARIABLE u, v : INTEGER;
(11)      BEGIN
(12)         u := a+b;
(13)         v := a-b;
(14)         IF (a > b)
(15)          THEN
(16)            c <= u;
(17)          ELSE
(18)            c <= v;
(19)         END IF;
(20)      END PROCESS;
(21)    END foo;
```

FIGURE 1 A Behavioral Specification in VHDL.

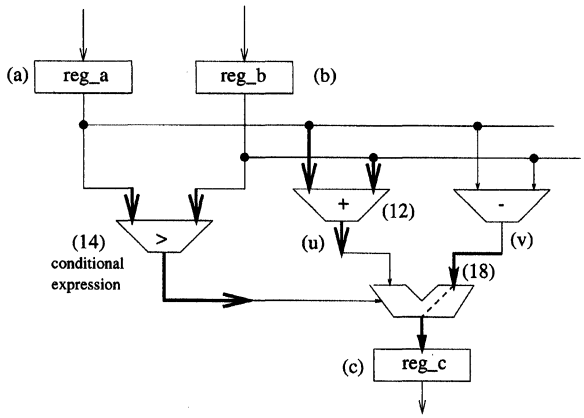

FIGURE 2 A RTL Data path Synthesized from Specification in Figure 1.

the RTL design is also shown by the line number annotations. Each register is associated with a carrier in the description, for example, register *a* corresponds to the port *a* in the specification.

The profile data obtained by behavioral profiling should indicate the usage of different hardware elements. For example, the profile data of an assignment statement in the behavioral description gives an estimate of the excitation frequency of the corresponding path in the hardware. In our example, if line number (18) has a profile data of 10, it means that the corresponding path from the output of subtracter through the multiplexor to the input of the register *c*, is excited ten times.

RTL designs generated by high level synthesis systems contain temporary registers and interconnect units which have no direct correspondence with constructs in behavior level specification. Profile data for such RTL components which do not explicitly appear in the specification has to be calculated by some indirect means.

In order to profile a behavioral specification the profiler inserts monitoring code in the specification. This code typically declares, initializes and increments various counters to measure various types of event activity. The modified program is then simulated to determine the profile data.

Behavior profiler takes the CDFG representation of the specification and generates equivalent VHDL program with probes (counters and similar monitoring variables) to gather various event activities. We need to profile the CDFG rather than the original specification since the CDFG representation exposes all the operations and carriers (edges in CDGF) that will be bound to hardware resources.

The generated VHDL program is simulated using input vectors called the *profiling stimuli* supplied by the user. Profiling stimuli should represent typical usage of the design being synthesized. Since profiling stimuli will decide the event activity in the design, the user should take extreme care in preparing this data. Some suggestions as how to prepare profiling stimuli for different classes of designs are given in [11].

For the given profiling stimuli, the *profile data* of the specification constitutes the following information associated with the CDFG nodes and edges. The *event activity* of a CDFG node *op* is the number of times that node is executed and is denoted by E_{op}. The *transaction activity* of an edge *e* is the number of times the edge is traversed during the execution and is denoted by T_e. The *event activity* of an edge *e* is the numbers of times input changed on the edge and is denoted by E_e.

Note that $E_e \leq T_e$. Probes are inserted by the profiler to measure the *profile data.*

5. POWER CHARACTERIZATION OF RTL MODULES AND PLAs

5.1. Module Library Characterization

The RTL module library contains parameterized modules such as n-bit registers, n-bit adders and n-bit m-to-1 multiplexors. Modules are parameterized with respect to bit-width of each input and, where applicable, the number of inputs. For each module in the library, its interface description, parameters such as area, delay and average intrinsic switched capacitance (ISC) characteristics are stored in the library. The area, delay and ISC characteristics are expressed as a function of parameter variables such as bit-width, word length etc. and are in the form of either equations or tables. If the data cannot be fit into an equation, then it is stored as a table. For tables, linear interpolation or extrapolation is assumed whenever the parameter value is not available for a given value of parameter variable.

For a given library module, area, delay and ISC values are determined by generating layouts for different parameter values. Linear regression by the method of least squares is used to find an equation which determines the area, delay or ISC characteristic given the bit-width parameter value. If the standard error is too high, then the data is entered as a table assuming the use of linear interpolation in between the data points. Determination of area and delay parameters for layout instances is straightforward. Area can be directly measured from the layout and delay can be determined through simulation or a timing analysis programs such as Crystal [34]. Determination of ISC which depends on input patterns is more involved and is described below.

We define the *average intrinsic switched capacitance* (ISC) of a module instance as the average capacitance that is expected to switch when an input event (change of logic values on the input lines) takes place. ISC of a module instance is determined by extracting a switch level model from its layout, simulating the switch level module using a very long stream of randomly generated input patterns and monitoring the capacitance switched per pattern, until convergence occurs as discussed below. The capacitance measurements are carried out by IRSIM-CAP [37], which is a modified version of IRSIM [38] switch level simulator for better capacitance measurements.

Let C_k be the total capacitance charged after applying k random input patterns without reinitialization between successive patterns. $Z_k = C_k/k$ denotes the average capacitance per input pattern after applying k patterns. $\delta_k = |Z_k - Z_{k-1}|/Z_{k-1}$ denotes variation in the average capacitance between the $k-1$ th and k th patterns. We continue to apply random input patterns until δ_k remains less than 0.001 over 1000 consecutive input pattern applications. At this point we say that the average switching capacitance estimation converged and accept the value of Z_k after the last input pattern is applied. This value is the ISC of that instance of the module. Similar procedure is used to determine the ISC of various instances of the module and results are expressed as an equation or table.

Figure 3 shows the ISC characteristics of a library module. Figure 4 shows ISC plots with

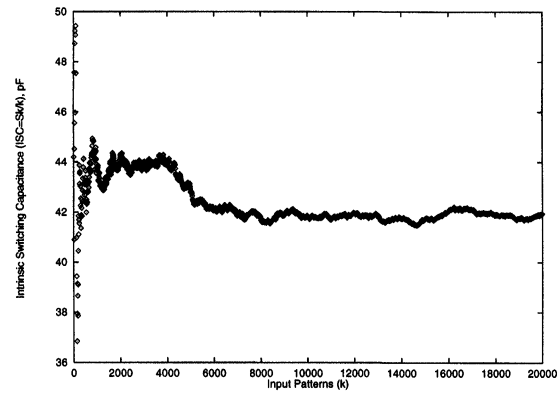

FIGURE 3 ISC Characteristic of a 16-bit Register.

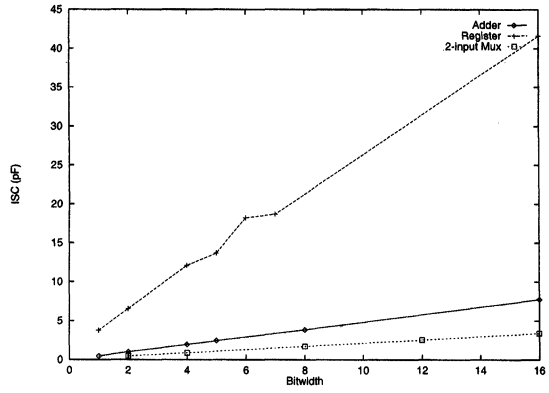

FIGURE 4 ISC vs. Bit-Width for three Parameterized Modules.

respect to the bit-width for three modules, namely, adder, register and two-input multiplexor. Table I shows the ISC characteristics of some PDSS library modules. For RAM component, there are two parameters namely, select size and the word size. The ISC value shown for RAM is the average capacitance switched for either a Read or a Write operation.

5.2. PLA Characterization

The controller is a finite-state machine which we assume is implemented as a PLA structure. The PLA structure consists of an input plane, an output plane, and I/O buffers. We assume that the PLA is implemented using dynamic CMOS with pre-charged product and output lines [19]. The product and output lines are selectively discharged based on the input conditions and are controlled by two non-overlapping clocks.

A PLA is characterized by three parameters: (1) *input size,* $\mathcal{I}$; (2) *output size,* $\mathcal{O}$; and (3) the *number of states,* $\mathcal{S}$. The ISC for any controller function of these parameters. By varying $\mathcal{I}, \mathcal{O}, \mathcal{S}$, random PLAs are generated and characterized as follows: The switch level model of the controller, extracted from the layout, is simulated using random input vectors. Simulation is carried out until the capacitance switched per clock step (as opposed to per input pattern in the case of the modules in the library) converges in a fashion similar to the one described in module library characterization.

TABLE I ISC Data for Some Parameterized Library Modules (Bit Width ≥ 1)

Sl.	Module	ISC Table (Bit Width-ISC(pF))
1.	Adder	1–0.45, 2–0.98, 4–1.93, 5–2.43, 8–3.84, 16–7.74
2.	Subtracter	2–0.97, 4–2.50, 6–3.26, 8–5.64, 10–7.05, 16–12.16
3.	Comparator (>)	1–0.44, 2–0.88, 4–1.82, 5–2.00, 6–2.78, 8–3.99, 16–12.57
5.	Multiplier	2–2.27, 3–3.53, 4–7.99, 5–15.30, 8–60.48, 16–455.39
6.	Multiplexor	2-inputs: 2–0.45, 4–0.86, 8–1.70, 12–2.53, 16–3.39 4-inputs: 2–1.41, 4–2.68, 8–5.20, 12–7.95, 16–10.79 6-inputs: 2–2.46, 4–4.69, 8–9.46, 12–14.53, 16–19.53 8-inputs: 2–3.29, 4–6.23, 8–13.10, 12–19.89, 16–26.73
7.	Register	1–3.77, 2–6.53, 4–12.09, 5–13.68, 6–18.19, 7–18.67, 16–41.62
8.	Signal Register (Register + Glue Logic)	2–10.90, 3–12.41, 4–15.45, 5–15.99, 8–23.78, 16–39.35
9.	AND	2–0.17, 3–0.29, 4–0.36, 5–0.45, 6–0.55, 8–0.76, 10–0.97, 16–1.55
10.	OR	2–0.18, 3–0.27, 4–0.38, 5–0.48, 6–0.51, 8–0.71, 10–0.98, 16–1.48
11.	NOT	1–0.04, 2–0.08, 3–0.12, 4–0.16, 5–0.20, 8–0.33, 16–0.66
12.	NAND	1–0.06, 2–0.13, 3–0.19, 4–0.26, 5–0.32, 6–0.38, 7–0.44, 8–0.53, 16–1.06
13.	NOR	3–0.17, 4–0.22, 5–0.28, 6–0.35, 8–0.47
14.	XOR	2–0.31, 3–0.50, 4–0.68, 5–0.86, 6–0.98, 8–1.35, 10–1.69
15.	XNOR	2–0.31, 3–0.50, 4–0.64, 5–0.80, 6–0.97, 8–1.26, 10–1.61, 16–2.56
16.	RAM	sel_size = 2: 1–159.84, 2–187.12, 4–244.310, 8–392.75, 16–592.48 sel_size = 3: 1–217.57, 2–250.65, 4–318.67, 8–463.22, 16–736.95

A PLA characterization table is obtained, which is used later for the estimation of switched capacitance in the controller. Table II shows a portion of the PLA characterization table. Figure 5 shows a three dimensional plot of ISC values for controllers with varying $\mathcal{O}$, $\mathcal{S}$ and with input size, $\mathcal{I}=5$.

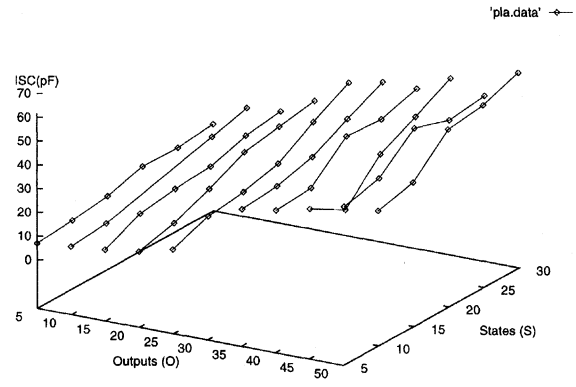

FIGURE 5 PLA Characterization with size of Inputs $I=5$.

6. ARCHITECTURAL POWER ESTIMATION

PDSS (Fig. 6) accepts specifications in a behavioral subset of VHDL and user-specified constraints in terms of clock-period and area. It generates a RT level design satisfying the given constraints. PDSS consists of four main modules: scheduling and performance estimation, register optimization, interconnect optimization and controller generation. More detailed discussion on PDSS appeared in [11, 33].

The RT level design produced by PDSS consists of four major subunits from the power estimation view point: Datapath, Controller, Interconnect

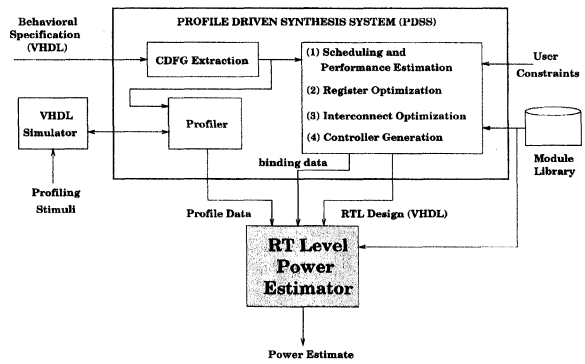

FIGURE 6 PDSS Environment.

TABLE II A portion of the PLA Characterization table

Sl.No	$\mathcal{I}$	$\mathcal{O}$	$\mathcal{S}$	ISC(pF)
1.	5	15	5	9.50
2.	5	15	10	16.45
3.	5	15	15	18.85
4.	5	15	20	20.00
5.	5	15	25	24.78
6.	5	15	30	26.84
7.	5	20	5	11.39
8.	5	20	10	15.14
9.	5	20	15	21.42
10.	5	20	20	28.66
11.	5	20	25	31.34
12.	5	20	30	33.82
13.	10	25	10	23.37
14.	10	25	15	25.36
15.	10	25	20	35.97
16.	10	25	25	43.93
17.	10	25	30	43.99
18.	10	30	10	27.40
19.	10	30	15	29.35
20.	10	30	20	41.40
21.	10	30	25	47.96
22.	10	30	30	48.61

and System Clock. Power consumed in the design is given by,

$$P_{\text{design}} = P_{\text{dp}} + P_{\text{con}} + P_{\text{inter}} + P_{\text{clock}}$$

where P_{dp}, P_{con}, P_{inter}, and P_{clock} are the power consumed in the datapath, controller, and interconnect and system clock respectively.

Our RT level power estimator needs the following inputs as shown in Figure 6:

1. *Profile Data*: This is obtained from the behavioral profiling of the high level specification given as input to the PDSS. For each operator and each edge in the CDFG a count of total event activity occured on the operator/edge is recorded.

2. *Binding Information*: One of the synthesis tasks is to bind each operator and edge in the CDFG to an instance of one of the modules in the module library. It also binds the temporary variables introduced to hardware registers in the module library.
3. *Module Library*: The module library is pre-characterized for ISC as explained in the section 5.
4. *RT Level Output*: This is the structural implementation containing instantiations of modules from the module library. The controller is a finite state machine description. The datapath and controller interact with each other to form the entire design.

To estimate average power of a given RT-Level design, the power estimator goes through the following phases: (1) Pre-processing stage; (2) Profile data computation of hardware resources introduced during synthesis; and (3) Power estimation of the design.

6.1. Pre-processing Stage

The power estimator initializes with the ISC values of all the modules obtained from the library characterization. The binding information provided by the synthesis tool is used to build a list of instances (*inst_list*) of modules. Each instance is initialized with sum of the profile data of all the operators (or edges) in the CDFG which are bound to that particular instance. Note that some of the instances' profile data is not known as they are introduced during synthesis. The profile data of such instances is computed in the next phase.

6.2. Profile Data Computation

Algorithm **Compute_profile()** in Figure 9 is used to compute the profile data of the temporary registers and the interconnect units introduced during the synthesis.

Procedure **Build_dependency_list()** builds a dependency list of the instances. It goes through each instance $inst_i$ in the instance list *inst_list* and if the profile data of the instance is unknown, then it adds the instances at the inputs, to the $inst_i$'s dependency list.

Consider Figure 7 in which there is a feedback from the output of the multiplexor inst(j) to the input of the register inst(i). Such a configuration gives rise to dependency cycles. Procedure **Remove_cycles ()** removes the above described dependency cycles in the following way. Let two instances i and j be in a dependency cycle. Besides the input which gives rise to a dependency cycle, if the profile data on remaining inputs of an instance is known, then let us say that the profile data of that instance is known. Otherwise, the profile data of the instance is said to be unknown. The following three possibilities can occur:

Case 1: The profile data of both instances is known. The profile data of each of the instances is equal to the sum of the profile data of both instances.

Case 2: The profile data on only one of the instances (*say* i) is known. We remove the edge from j to i. Assuming that instance j is not in a dependency cycle with any other instance, the profile data of j can be computed, which is the sum of the profile data of all the instances (including i) at its inputs. Since there was an edge from j to i, the instance i has event activity from the output of instance j. Thus the new profile data of i is the old profile data plus the computed profile data of instance j.

Case 3: The profile data of both instances is not known. Both the edges in the cycle are removed,

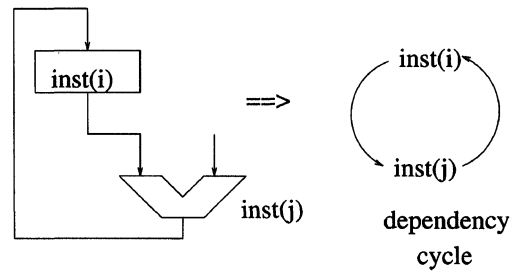

FIGURE 7 An example of dependency cycle.

profile data of i and j are computed based on the profile data of the instances at other inputs. The new profile data of each instance is the sum of the profile data of both instances.

To illustrate Case 1, consider Figure 8. Two instances inst(i) and inst(j) are both in a dependency cycle. The profile data on inputs A and B of inst(i) are 10 and 20 respectively. Similarly, the profile data on inputs of inst(j) namely, C and D are 15 and 12 respectively. We make a conservative assumption that the inputs of a multiplexor are not switching simultaneously. Thus, the profile data on the output of a multiplexor is the sum of the profile data on all the inputs. Thus, the equations to compute profile data on outputs of both instances are:

$$P(X) = P(Y) + P(A) + P(B)$$

$$P(Y) = P(X) + P(C) + P(D)$$

Where $P(X)$ is the profile data on the output of inst(i) and $P(Y)$ is the profile data on the output of inst(j). $P(X)$ appears on the right hand side of $P(Y)$ equation and vice versa. The above set of equations cannot be solved, unless we remove the dependency cycle. Since $P(A)$, $P(B)$, $P(C)$ and $P(D)$ are known, the example belongs to Case 1 as discussed above. With the dependency cycle removed, the profile data of X and Y are $P'(X) = P(A) + P(B) = 30$ and $P'(Y) = P(C) + P(D) = 27$. With the dependency cycle included, the new profile data for both X and Y are, $P(X) = P(Y) = P'(X) + P'(Y) = 57$. If $P(A)$ or $P(B)$ is unknown to start with, then the example belongs to Case 2. If $P(A)$ or $P(B)$ and $P(C)$ or $P(D)$ is not known then the example belongs to Case 3.

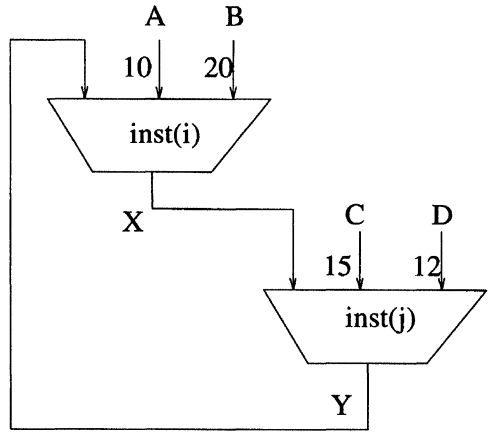

FIGURE 8 An example to illustrate profile data computation in presence of a dependency cycle.

After removing the cycles, for each instance i whose profile data is unknown, it is calculated as the sum of profile data of all the instances at i's inputs. After computing profile data for all the instances, the data path power can be computed as follows.

6.3. Power Estimation in Data Path

The data path consists of the execution units such as adders and multipliers and storage units such as latches and shift registers. The power consumed by the datapath P_{dp}, and is computed by lines 2–4 of the procedure **Estimate_Power** ( ). P_{dp} is given by:

$$P_{dp} = \sum_{op} E_{op} * \mathrm{ISC}_{op}$$

Where E_{op} is the event activity (or profile data) of the operator (or register) and ISC_{op} is the average switched capacitance value of a hardware module instance to which the operator node op is bound.

6.4. Power Consumed by System Clock

As the system clock controls all the clocked components in the data path, it is loaded by a

```
Algorithm Compute_profile()
begin
1. T = Build_dependency_list();
2. Remove_cycles(T);
3. for each I in inst_list do
4.         for each J in I.dependency_list do
5.                 I.profile_data += J.profile_data
6.         end for
7. end for
end
```

FIGURE 9 Algorithm for the computation of profile data.

large amount of capacitance. The power consumed by the system clock is estimated in Algorithm **Estimate_Power** ( ) shown in Figure 10. The lines 8 – 10 estimate the load capacitance C_{clock} on the system clock. In the clocked components such as registers and latches, the load capacitance on the clock line varies approximately 50fF per bit-width. Thus, total capacitive load on the system clock is the sum of the clock capacitances of each instance. The total capacitance switched in a design is given by the product of number of input vectors (N_v), total number of clock cycles required to process an input vector (T_{total}) and clock capacitance (C_{clock}).

6.5. Power Estimation in Controller

The controller is a finite state machine implemented as a PLA. Any PLA is characterized by three parameters: the number of inputs $\mathcal{I}$, the number of outputs $\mathcal{O}$ and the number of states, $\mathcal{S}$. In the module library, there already exists a PLA characterization table, which was explained in detail in section 5. From the table, we can obtain the average intrinsic switching capacitance ISC of a PLA of a given size. Interpolation/Extrapolation is assumed where ever the values are not available for a given set of parameter values. The ISC value so obtained is the average capacitance that switches per clock step in the PLA of size $(\mathcal{I}, \mathcal{O}, \mathcal{S})$.

```
Algorithm Estimate_power()
begin
1. for each I in inst_list do
2.       if (I.module_type == OPERATOR OR REGISTER) then
3.             P_dp += I.profile_data * ISC(I.op_type, I.size)
4.        else
5.         if (I.module_type == INTERCONNECT) then
6.             P_inter += I.profile_data * ISC(MULTIPLEXOR, I.size)
7.          endif
8.       endif
9.       if (I.module_type == CLOCKED_COMP) then
10.            C_clock += 50fF * (I.size)
11.       endif
12. end for
13.
14. Let I, O and S be the controller size.
15. T_total=Estimate_clock_steps()
16. C_con = ISC(I, O, S)
17. P_con = N_v * T_total * C_con
18. P_clock = N_v * T_total * C_clock
20. P_total = P_dp + P_con + P_inter + P_clock
end
```

FIGURE 10 Algorithm for the estimation of power.

Let N_v be the total number of profiling stimuli applied. Let the CDFG be scheduled in N_c number of control steps. In the module library, for each module, the number of clock steps needed to process an input vector is stored as function of its parameters such as bit-width, wordsize etc. The total number of clock steps required to process an input vector is sum of the maximum number of clock steps needed in each of the control step. Algorithm **Estimate_clock_steps** ( ) estimates the number of clock steps needed by the design to process an input vector (say T_{total}). The power consumed in the controller is given by the product of N_v, T_{total} and the ISC$(\mathcal{I}, \mathcal{O}, \mathcal{S})$ as given in Algorithm shown in Figure 10.

6.6. Power Estimation in Interconnect

Already the profile data for the interconnect units has been calculated as discussed in profile data computation phase. The interconnect units are not present in CDFG and arise due to operator sharing, register sharing and interconnect sharing. In this work, we consider only Multiplexor-based designs.

The profile data of a multiplexor is computed as the sum of the event activities on all the inputs. This is a conservative estimate of the total number of events that the multiplexor is subjected to.

The power consumed in the interconnect is calculated in the same way as is done for the datapath.

```
Algorithm Estimate_clock_steps()
begin
0
1.       for i in 0 to N_c do
2.         max_clock_cnt ← 0
3.         for each op scheduled in control step i do
4.            if( module_op.clock_steps > max_clock_cnt)
                        where op is bound to module_op then
5.                max_clock_cnt ← module_op.clock_steps
6.            endif
7.         end for
8.         T_total += max_clock_cnt
9.       end for
end
```

FIGURE 11 Algorithm to estimate the number of clock steps.

It is given by:

$$P_{\text{inter}} = \sum_i E_i * \text{ISC}(MUX,\ i.\text{size})$$

where i is a instance of a multiplexor of size i.size. and ISC (MUX, i.size) is the average intrinsic switching capacitance of a multiplexor of size i.size.

7. RESULTS

In this section we present experimental results for six designs:

1. Compression chip
2. Decompression chip
3. FIFO, a first-in first-out queue
4. Find, sort and search chip
5. Shuffle Exchange Network [35]
6. Traffic light controller

Table III shows the behavioral specification data. PDSS system is implemented in C++ on Sun Sparcstation platforms.

Each register level design produced by PDSS is processed by the Lager IV silicon compiler [36] to generate mask layouts. The designs generated use a two phase non-overlapping clocking scheme. Although the designs are generated in a scalable CMOS technology, all results for this paper are obtained using 2 micron feature size. Switch level models are extracted from the layouts and simulated using the IRSIM-CAP [37] switch level simulator. Table IV shows the synthesized design data at the layout level.

Table V shows the estimated and actual powers in the data path and interconnect of the six designs. The estimated power is computed at the RT-level and actual power is determined by the switch level simulation of the synthesized designs. As shown in the table, the percentage error in estimation for data path is in the range of 2.51% – 12.58% with the average deviation from the actual value being 6.25%.

Table VI shows the comparison of powers for controller. The estimation error is in the range 3.53% – 15.22% with the average deviation being 10.51%. Table VII shows the comparison of the power dissipated due to the system clock. The estimation error is in the range 18.59% – 30.69% with the average deviation of 22.32%. Table VIII shows the power values for the entire design, which is the sum of the power in datapath (P_{dp}), interconnect (P_{inter}), system clock (P_{clock}), and

TABLE III Behavioral Specification Data for Six Designs

Sl.	Design	LOC	DFG Nodes	DFG Edges	Profiling Stimuli	Profiling Time (s)
1.	Compress	42	22	107	25	9.48
2.	Decompress	40	22	106	25	6.00
3.	FIFO	70	38	176	25	16.33
4.	Find	63	33	121	16	9.81
5.	Shuffle Xchg NW	450	31	2040	14	30.90
6.	TLC	72	27	123	10	1.37

TABLE IV Synthesized Design Data at the Layout Level

Sl.	Design	Clock Period	Nodes	Transistors	Area (sq. mm.)	Cycles	Simulation Time (min)
1.	Compress	200ns	2,946	6,315	10.9	1450	5.34
2.	Decompress	200ns	2,803	6,059	10.3	825	3.23
3.	FIFO	900ns	4,438	10,688	24.6	2,364	20.44
4.	Find	550ns	5,602	11,458	20.3	5,360	35.00
5.	Shuffle Xchg	160ns	49,655	95,004	418.7	1,975	240.00
6.	TLC	200ns	1,938	4,769	6.9	420	1.28

TABLE V Comparison of the power in the Data path and Interconnect

Sl.No	Design	Total ($P_{dp}+P_{inter}$)		
		Estimated pF	Actual pF	%Devn.
1.	Compress	24219	21171	12.58
2.	Decompress	15034	14473	3.73
3.	FIFO	56863	51614	9.23
4.	Find	266602	281415	5.55
5.	Shuffle	525207	545976	3.95
6.	TLC	4415	4526	2.51
	Average Error			6.25

TABLE VI Comparison of the power in the Controller

Sl.No	Design	Total (P_{con})		
		Estimated pF	Actual pF	%Devn.
1.	Compress	45974	39209	14.71
2.	Decompress	26309	22303	15.22
3.	FIFO	338266	304340	10.03
4.	Find	351066	330718	5.79
5.	Shuffle	297400	256303	13.81
6.	TLC	13906	13414	3.53
	Average Error			10.51

TABLE VII Comparison of Clock power

Sl.No	Design	Total (P_{clock})		
		Estimated pF	Actual pF	%Devn.
1.	Compress	10793	13036	20.78
2.	Decompress	6249	8167	30.69
3.	FIFO	27580	33958	23.12
4.	Find	59787	47241	20.98
5.	Shuffle	554768	444923	19.80
6.	TLC	2786	2268	18.59
	Average Error			22.32

TABLE VIII Comparison of the total power for the Entire Design

Sl.No	Design	Total (P_{design})		
		Estimated pF	Actual pF	%Devn.
1.	Compress	80986	73416	9.35
2.	Decompress	47592	44943	5.57
3.	FIFO	422709	389912	7.75
4.	Find	677455	659374	2.66
5.	Shuffle	1377375	1247202	9.45
6.	TLC	21107	20208	4.26
	Average Error			6.51

controller (P_{con}). The percentage error is in the range 4.26% – 9.35% and the average deviation is 6.51%. This shows reasonable correlation between the estimated and actual values not only in the entire design but also in the datapath and controller seperately.

8. DISCUSSION AND CONCLUSIONS

The following are some of the factors which have not been taken into account during the power estimation:

1. Effect of Placement and Routing
2. The random characterization of the RTL module library and PLAs gives rise to an inherent estimation error. This can be remedied by taking the activity on the inputs into the estimation procedure.
3. Glitch power consumption.
4. PLA characterization based on only inputs, outputs and states is not sufficient. The state table information has to be taken into account.
5. In the estimation of power in multiplexors, we assumed that the activity on the inputs is added up to get the activity of the multiplexor. We are making a very conservative assumption that all the inputs are not switching simultaneously. This is another source of error.

In this work, we presented an accurate power estimation technique based on the profile data obtained at the behavior level. The estimation technique is implemented in the framework of a high level synthesis system. Compared to the estimation techniques at the lower levels of abstraction, the technique is faster in the execution time. For the six examples considered, the average estimation error at the design level is within 10%, which demonstrates that the estimation technique is reliable.

Acknowledgements

This work is done at the University of Cincinnati and is supported in part by the Solid State Electronics Directorate of the Wright Laboratory of the US Air Force under contract number F33615-91-C-1811 and by the Advanced Research Projects Agency under order no. 7056 monitored by the Federal Bureau of Investigation under contract no. J-FBI-89-094.

References

[1] Camposano, R. and Wayne Wolf. (1991). "High Level VLSI Synthesis", *Kluwer Academic Publishers.*

[2] Lemnois, Z. J. and Gabriel, K. J. (1994). "Low-Power Electronics", *IEEE Design and Test of Computers*, pp. 8 – 13, Winter.

[3] Najm, F., "Towards a high-level power estimation capability", In *Proceedings of the 1995 International Symposium on Low Power Design*, April 1995.

[4] Powell, Scott R. and Chau, Paul M. (1990). "Estimating Power Dissipation of VLSI Signal Processing Chips: The PFA Technique," *VLSI Signal Processing IV*, pp. 250 – 259.

[5] Powell, Scott R. and Chau, Paul M., "A Model for Estimating Power Dissipation in a Class of DSP VLSI Chips", *IEEE Transactions on Circuits and Systems*, **38**(6), June 1991.

[6] Chandrakasan, Anantha P. *et al.*, "HYPER-LP: A System for Power Minimization Using Architectural Transformations", *Proceedings of ICCAD*, pp. 300 – 303, November 1992.

[7] Chandrakasan, Anantha P. *et al.*, "Optimizing Power Using Transformations", *IEEE Transactions on Computer Aided Design*, pp. 12 – 31, January 1995.

[8] Landman, P., "Low-Power Architectural Design Methodologies", Ph.d. Thesis, Memorandum No. UCB/ERL M94/62, 30th August 1994.

[9] Landman, P. and Rabaey, J., "Black-Box Capacitance Models for Architectural Power Analysis", *Proceedings of the 1994 International Workshop on Low Power Design*, Napa Valley, CA, pp 165 – 170, April 1994.

[10] Rabaey, J. M., Chu, C., Hoang, P. and Potkonjak, M., "Fast Prototyping of Datapath-Intensive Architectures," *IEEE Design and Test of Computers*, pp. 40 – 51, June 1991.

[11] Nand Kumar, Srinivas Katkoori, Leo Rader and Ranga Vemuri (1995). "Profile-Driven Behavioral Synthesis for Low Power VLSI Systems", *IEEE Design and Test of Computers*, pp. 70 – 84, Fall.

[12] Renu Mehra and Rabaey, Jan, M., "Behavioral Level Power Estimation and Exploration", *Proceedings of the International Workshop on Low Power Design*, pp. 165 – 170, April 1994.

[13] Paul Landman and Jan Rabaey, "Power Estimation for High Level Synthesis", *Proceedings of EDAC-EUROASIC*, pp 361 – 366, February 1993.

[14] Anand Raghunathan and Jha, Niraj K. (1994), "Behavioral Synthesis for Low Power", *Proceedings of ICCD.*

[15] Anand Raghunathan and Jha, Niraj K. (1995). " An ILP formulation for low power based on minimizing switched capacitance during data path allocation", *in the Proceedings of IEEE Symposium on Circuits and Systems.*

[16] Chandrakasan, Anantha P., Sheng, S. and Brodersen, R., "Low Power CMOS digital design", *IEEE Transactions of Solid State Circuits*, April 1992.
[17] Najm, F. N., "A survey of power Estimation Technique in VLSI circuits (Invited Paper)" *IEEE Transactions VLSI Systems*, **2**(4), 446–455, January 1995.
[18] Srinivas Katkoori, Nand Kumar and Ranga Vemuri, "High Level Profiling Based low Power Synthesis Technique", *In the Proceeedings of ICCD*, October 1995.
[19] Weste, N. and Eshraghian, K. (1985). "*Principles of CMOS VLSI Design: A Systems Perspective*", Addison-Wesley.
[20] Veendrick, H. J. M., "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *IEEE Journal on Solid State Circuits*, **SC-19**, pp. 468–473, August 1984.
[21] Ravi Kalyanaraman, "Behavioral Test Generation for VHDL Programs", MS Thesis, Department of Electrical and Computer Engineering, University of Cincinnati, September 1993.
[22] Darrel Ince (1991). "Software Testing", in John McDermid (ed.) *Software Engineer's Reference Book*, Butterworth-Heinemann Ltd.
[23] John Hennessy and David Patterson (1990). "*Computer Architecture: A Quantitative Approach*", Morgan Kaufmann Publishers.
[24] Stallings, W. (ed.) (1990). "*Reduced Instruction Set Computers (RISC)*", IEEE Press.
[25] Cmelik, R. F., Kong, S. I., Ditzel, D. R. and Kelly, E. J., "An Analysis of MIPS and SPARC instruction set utilization on the SPEC benchmarks", *In ASPLOS-IV Proceedings, SIGARCH Computer Architecture News 19*, pp. 290–302, 2 April 1991.
[26] Graham, S. L., Kessler, P. B. and McKusick, M. K., "An execution profiler for modular programs", Software Practice Exper. 13, pp. 671–685.
[27] Morris, W. G., "CCG: A prototype coagulating code generator", *Proceedings of the SIGPLAN 91 Conference on Programming Language Design and Implementation*, SIGPLAN Nat.(ACM) pp. 45–58, 26, June 1991.
[28] Pettis, K. and Hanson, R. C., "Profile guided code positioning", *Proceedings of the SIGPLAN 91 Conference on Programming Language Design and Implementation*, SIGPLAN Nat. (ACM) pp. 16–27, June 1990.
[29] Sarkar, V., "Determining average program execution times and their variance", *In Proceedings of the ACM SIGPLAN 89 Conference on Programming Language Design and Implementation*, SIGPLAN Nat. (ACM), 289–312, 24 June 1989.
[30] Ball, T. and Larus, J. R., "Optimally Profiling and Tracing Programs", *ACM Transactions on Programming Languages and Systems*, **16**(4), 1319–1350, July 1994.
[31] Goldberg, A., "Reducing overhead in counter- based execution profiling", Tech Rep. CSL-TR-91–495, Computer Systems Lab., Stanford Univ., Standford, Calif, Oct. 1991.
[32] Samples, A. D., "Profile Driven Compilation", Ph.D thesis (Rep. UCB/CSD 91/627), Computer Science Dept., Univ. of California, Berkeley, Apr. 1991.
[33] Jayanta Roy and Ranga Vemuri (1992). "DSS : A Distributed Synthesis System", *IEEE Design and Test of Computers.*
[34] John Ousterhout (1987). "Using Crystal for Timing Analysis", Electrical Engineering and Computer Sciences, University of California at Berkeley.
[35] "Novel IC Shuffles Parallel Processing Data", *Electronic Products*, pp. 42–50, August 1, 1986.
[36] Rajeev Jain *et al.*, "An Integrated CAD System for Algorithm-Specific IC Design", *IEEE Transactions on Computer Aided design*, **10**(4), April 1991.
[37] Landman, P., "IRSIM-CAP - Modified version of IRSIM for better Capacitance Measurements, Univ. of Calif. Berkerley.
[38] Salz, A. and Horowitz, M., "IRSIM: an incremental MOS switch-level simulator," in *Proc Design Automation Conf.*, pp. 173–178, June 1989.

Authors' Biographies

Srinivas Katkoori is an assistant professor in computer science and engineering at the University of South Florida, Tampa. In 1997, he received his doctoral degree in computer engineering from the University of Cincinnati. In 1992, he received his bachelor's degree in electronics and communication engineering from the Osmania University, India. His research interests are in high-level synthesis and low-power synthesis of VLSI systems. Katkoori is a member of IEEE and ACM SIGDA.

Ranga Vemuri, an associate professor of electrical and computer engineering at the University of Cincinnati, also directs its Laboratory for Digital Design Environments. His interests include the computer-aided design of digital systems, formal verification, system synthesis, performance modeling, hardware description languages, and parallel algorithms. Vemuri received the M.Tech., degree from the Indian Institute of Technology, Kharagpur, and the Ph.D., from Case Western Reserve University, both in computer engineering. He received the Siddhartha gold medal, a distinguished research award, and an outstanding teacher award. He is a member of the IEEE Computer Society, IEEE Circuits and Systems Society, ACM SIGDA, American Society of Electronic Engineers, and Eta Kappa Nu.