RT Level Power Analysis †

Jianwen Zhu, Poonam Agrawal, Daniel D. Gajski
Department of Information and Computer Science
University of California, Irvine, CA 92717-3425

Abstract

Elevating power estimation to architectural and behavioral level is essential for design exploration beyond logic level. In contrast with purely statistical approach, an analytical model is presented to estimate the power consumption in datapath and controller for a given RT level design. Experimental result shows that order of magnitude speed-up over low level tools as well as satisfactory accuracy can be achieved. This work can also serve as the basis for behavioral level estimation tool.

1 Introduction

With the increasing demand of low power applications, there is a growing interest in power estimation techniques. It is essential for the power optimization tools in that

- it provides the evaluation of the cost function,
- it helps to identify the "hot-spots" – the candidates for further optimization.

Power estimation tools can operate on different levels of abstraction. A lot of interesting work has been done on circuit and gate level [Na94]. While these tools can often achieve very high accuracy, they are prohibitively expensive in architecture exploration, which is believed to be able to bring most of the power reduction. It is thus desirable to have estimator operating on RT level in order to provide fast evaluation of the power metric without sacrificing too much accuracy.

Some related work at this level include [La94] [Me94]. In contrast with those purely statistical approach, we present in this paper a power analysis technique which is analytical in nature.

The rest of the paper is organized in a bottom-up fashion. In Section 2, the power model of datapath components as well as interconnections is discussed. Then we present the power analysis techniques at the RT level in Section 3. We conclude the paper with some experimental results.

2 Component Level Power Analysis

In this section, we try to identify the sources of power consumption for the components in the datapath library as well as the interconnections such as buses and clock trees.

Power Model of Static CMOS Gates

Three main sources of power dissipation in static CMOS circuit are *dynamic switching*, *leakage current* and *short circuit current* respectively. The dominant factor is the first one due to the charging or discharging of circuit capacitances.

Power Model of Datapath Components

Having identified the sources of power dissipation for the gates, we need to investigate the power consumption model at the component level. In other words, we need to know the capacitance switched during each access of the *functional units*, *registers*, and *bus drivers*.

Ideally, the energy consumed for each access of a component should be a function of its (1) *bitwidth*, (2) its *previous data*, which determine the previous states of all the internal circuit nodes, (3) the *current data*, which determine the current states of all the circuit nodes and in turn their switching activities. This is not practical since the data is not available until run time. However, statistics measures such as mean, variance, and correlation on the input data are relatively easy to obtain through functional simulation. It is reasonable to expect that the energy of the component is a function of the statistics of the data and the bitwidth. Based on this idea, Component characterization techniques such as Dual Bit Model (DBT) are proposed to model the power consumption of datapath components [La94].

An alternative is to assume uniform white noise inputs for each component. Based on this assumption, the power consumption of a component depends solely on its size. Statistical methods can be applied to obtain an average value for each component in the library. We adopt this approach because of its simplicity.

In the discussions that follow, we assume each component *c* in the datapath library is associated with a capacitance C, the value of which is defined as the average capacitance switched for each access of the component.

Power Model of Interconnections

Strictly speaking, the power model for the bus and clock tree belongs to the subject of next section because they all depends on the RT level netlist. However, we advance it here for ease of discussion.

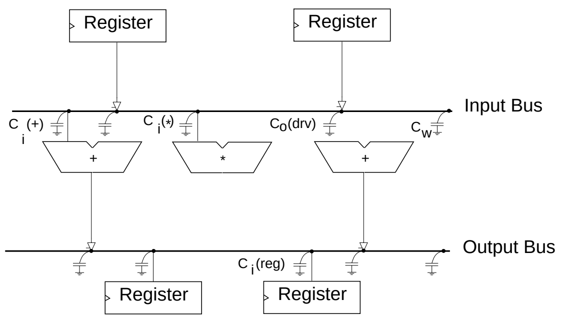

Figure 1: Capacitances of Multiplexed Bus

†This work is partially supported by Toshiba Inc.

There are two factors that contribute to the capacitance of the bus:

- **wire capacitance**, as indicated by C_w in Figure 1.

  The wire capacitance is determined by the length of the wire and in turn the result of routing. Estimation of wire length can be one of the following:

  1. performing detailed placement and routing;
  2. performing rough floorplaning, and then use the square root of the resultant chip area as an approximation of the wire length;
  3. summing up the area of all the components as an approximation of the chip area (assume the floorplaner is perfect), and then use the square root of the chip area.

  While 1 is too expensive to be practical and 2 needs an additional floorplaner, 3 is adopted for its simplicity.

- **component load**: the component load refers to the capacitances contributed by the units attached to the bus.

  There are two types of buses:

  1. *Multiplexed Bus:* As shown in Figure 1, bus drivers are used for multiplexed bus. For every data transfer bound to the bus, the capacitances introduced are: (1) the output capacitance of the bus driver ($C_o(Drv)$), (2) the input capacitance of the functional units for input buses (like $C_i(+), C_i(*)$), or the input capacitance of the register for output buses ($C_i(Reg)$).
  2. *Direct Connection*:

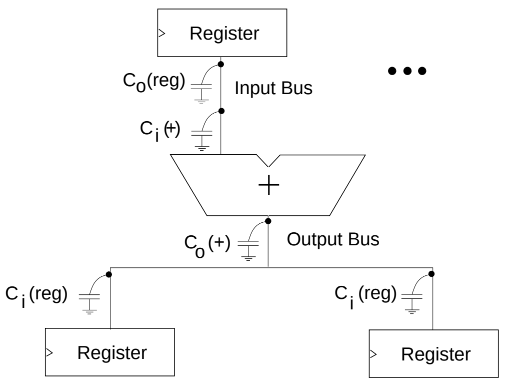

Figure 2: Capacitances of Direct Connection Bus

As shown in Figure 2, for direct connection bus, there is no need for bus drivers. For every data transfer bound to the bus, the capacitance introduced are: (1) the output capacitance of the source functional unit or register (like $C_o(+)$ or $C_o(Reg)$), (2) the input capacitance of the sink functional unit or register ($C_i(+)$ and $C_i(Reg)$).

Similarly, the capacitance of the clock tree is the wire capacitance plus the capacitance of the clock input $C_{clk}(Reg)$ of each register. Same technique can be applied:

$$C(Clock) = (C_w + C_{clk}(Reg)) \cdot |Reg|$$

where Reg is the set of registers in the design.

3 RT Level Power Analysis

3.1 Overview

Problem Statement

This section addresses the problem of estimating power at the RT level, which implies that the following is known:

1. *RT Level Description*

   A register transfer level design can be conveniently specified by a state action table (SAT), each row of which indicates that at a particular state, under a particular condition, the system will evolve to another given state, and the datpath will perform some given computation. A formal definition of the state action table will be given in Section 3.2.

2. *Branching Probability*:

   Given a state action table, the execution sequence of the system is still not known due to the unavailability of the conditions. We assume some profiling techniques are applied prior to the power analysis so that for each pair of rows (i, j) in the state action table, a branching probability $Prob(i, j)$ is obtained. A more detailed treatment will be presented in Section 3.4.

3. *Component Capacitance*:

   Based on discussions in Section 2, for every component in the datapath library, we assume that the average capacitance switched for each access is known. In other words, the average capacitance of each bus driver can be written as $C(Drv)$, each register can be written as $C(Reg)$, and each functional unit FU_i can be written as $C(FU_i)$. We also assume the input and output capacitances of each component are known.

   For interconnections such as bus and clock tree, although accurate information is not known until the layout stage, we assume some area estimation techniques discussed in Section 2 are applied such that for each bus Bus_i, we know the average capacitance switched for each access, denoted as $C(Bus_i)$. Similarly, the capacitance of the clock tree can be denoted as $C(Clock)$.

With the above information given, we need to estimate the power consumption of the hardware, which is defined as

$$Power = \frac{Energy}{Cycles \times Clock\ Period} \tag{1}$$

where $Cycles$ is the total number of clock cycles.

Architectural Model

In general, digital hardware can be modeled as an FSMD (Finite State Machine with a Datapath), where the datapath is responsible for the computation, and the controller determines when and what computation will be performed [Ga92].

Datapath

A typical datapath is shown in Figure 3. The datapath consists of *functional units*, *registers*, and *buses* (interconnections). The bus may or may not be attached with a *bus driver* depending upon whether it has different sources. We omit the case of multiplexers since they can be treated as bus drivers.

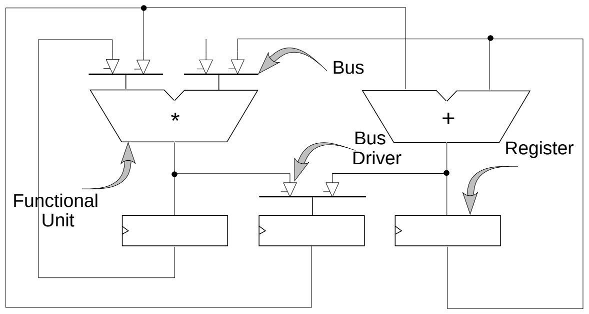

Figure 3: Datapath Model

Because the applications concerned in this work are often power critical, we assume another design style called *dynamic power management*, which is frequently adopted by designers (Figure 4): we assume each functional unit has an enable input in order to shut down the unit during its inactivity. The enable circuitry can be implemented simply as a switch which separates the bus and the functional unit. The enable controls the on/off of the switch. Note that in order for this technique to take effect, design care has to be exercised to ensure that the enable signal is asserted before the change of register output.

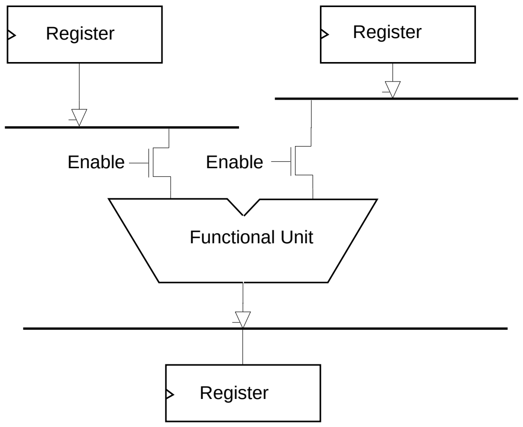

Figure 4: Functional Unit with Enable Input

Controller

Controller implements a finite state machine. In general, it contains a state register, which stores the current state, as well as some control logic to compute the next state and output signals.

Control logic can be implemented either as (a)ROM, or (b)PLA, or (c)2-level random logic, or (d)multi-level random logic.

While multi-level logic implementation is very difficult to predict, the analysis of the rest is similar and relatively simple. We take (c) as a representative of (a), (b), (c) and an approximation of (d) in this paper.

A typical 2-level logic controller implementation is shown in Figure 5.

As shown in Figure 5, A typical controller is composed of four parts, namely, the *state register*, *the decoder*, *next state logic* and *output logic*.

The decoder is implemented as a set of AND-gates. It takes as inputs each bit of the state register and the status signals as well as their complements.

The next state logic and output logic is implemented as a set of OR-gates. It take as input the output of the state decoder. There are three types of control lines in the output logic: (1) control lines for loading registers, (2) control lines for enabling (shutting down) the functional units, (3) control lines for the bus drivers.

Based on the structure of the hardware, computation of the energy consumption can be decomposed into

$$Energy = E(Datapath) + E(Controller) + E(Clock)$$

where

$$E(Datapath) = E(FU) + E(Reg) + E(Bus)$$

and

$$E(Controller) = E(SR) + E(NS) + E(Decoder) + E(OutputLogic)$$

3.2 Formal Definition of State Action Table

In this section we introduce some notations as well as a formal definition of the state action table.

An **activity vector** $\vec{V} = (v_1, v_2, ...)$ is defined as a boolean vector with $v_i \in \{0, 1\}$.

At a particular state, the state of the hardware can be characterized by a set of activity vectors, namely, the *current state vector* $\vec{S}$, the *status vector* $\vec{C}$, the *next state vector* $\vec{NS}$, the *function unit vector* $\vec{FU}$, the *the register vector* $\vec{Reg}$, the *the bus vector* $\vec{Bus}$, the *the bus driver vector* $\vec{Drv}$. While $\vec{S}, \vec{C}, \vec{NS}$ indicates the value of the state register, status signals and next state signals, the value of $\vec{FU}$, $\vec{Reg}$, $\vec{Bus}$, $\vec{Drv}$ indicates the activeness of corresponding datapath components.

The **cardinality** of the vector $\vec{V}$ is defined as the total number of 1's of the vector:

$$|\vec{V}| = \sum_{i=1}^{n} v_i$$

For $\vec{V} = (v_1, v_2, ..., v_n)$ and $\vec{W} = (w_1, w_2, ..., w_n)$, their **exclusive or** is defined as

$$\vec{V} \oplus \vec{W} = (v_1 \oplus w_1, v_2 \oplus w_2, ...)$$

their **concatenation** is defined as

$$\vec{V} \# \vec{W} = (v_1, v_2, ..., v_n, w_1, w_2, ..., w_n)$$

The **state tuple** $\vec{t}$ can then be defined as the concatenation of the above activity vectors.

$$\vec{t} = \vec{S} \# \vec{C} \# \vec{NS} \# \vec{FU} \# \vec{Reg} \# \vec{Bus} \# \vec{Drv}$$

The behavior of a RT level design can be specified by the **state action table** SAT, defined as a set of distinct state tuples:

$$SAT = \{\vec{t_i}\}$$

A **state trace** ST of SAT is defined as a sequence of state tuples in SAT:

$$ST = [\vec{t_1}, \vec{t_2}, ..., \vec{t_n}]$$

such that the next state vector of $\vec{t_i}$ equals to the current state vector of $\vec{t}_{i+1}$.

Note that the state action table defines the behavior of the hardware, whereas the state trace defines an actual execution scenario of the hardware. In the next two sections, we first discuss the computation of power consumption for an execution sequence in Section 3.3, based on which we derive estimation techniques for power consumption directly from the state action table in Section 3.4.

3.3 Power Estimation from State Trace

This section presents the analysis of power if a state trace $ST = [\vec{t}_1, \vec{t}_2, ..., \vec{t}_n]$ of the state action table SAT is given.

3.3.1 Cycles

The number of cycles of the state trace ST is simply the number of state tuples in ST:

$$Cycles = |ST| = n \tag{2}$$

It follows that

$$\begin{aligned} E(Clock) &= 2 \times C(Clock) \times V_{DD}^2 \times Cycles \\ &= 2 \times C(Clock) \times V_{DD}^2 \times |ST| \end{aligned} \tag{3}$$

where the factor 2 accounts for the switches of both the falling and rising edges of the clock.

3.3.2 Datapath

The activity of the datapath at state $\vec{t}_i$ can be characterized by the corresponding activity vectors: $\vec{FU}_i$, $\vec{Reg}_i$, $\vec{Bus}_i$, and $\vec{Drv}_i$. We denote their concatenation as $\vec{DP}_i$:

$$\vec{DP}_i = \vec{FU}_i \# \vec{Reg}_i \# \vec{Bus}_i \# \vec{Drv}_i$$

The capacitances of all the functional units in the datapath forms a capacitance vector $\vec{C}_{FU} = (C(FU_1), C(FU_2), ...)$. Similarly, we can define the capacitance vectors for registers, buses, and bus drivers as $\vec{C}_{Reg}, \vec{C}_{Bus}$ and $\vec{C}_{Drv}$ respectively. We denote their concatenation as $\vec{C}_{DP}$:

$$\vec{C}_{DP} = \vec{C}_{FU} \# \vec{C}_{Reg} \# \vec{C}_{Bus} \# \vec{C}_{Drv}$$

So the energy consumed at state $\vec{t}_i$ is

$$E(\vec{t}_i) = (\vec{DP}_i \cdot \vec{C}_{DP}) \times V_{DD}^2$$

where $|.|$ stands for the dot product of two vectors.

It follows that the total energy consumed on the execution sequence can be computed as

$$E(Datapath) = \sum_{\vec{t}_i \in ST} (\vec{DP}_i \cdot \vec{C}_{DP}) \times V_{DD}^2 \tag{4}$$

3.3.3 Controller

General Model

Figure 5 shows the controller implementation. The controller falls naturally into four parts, namely, the *state register*, the *decoder*, the *next state logic*, and the *output logic*. The decoder is essentially a set of AND-gates, inputs of which are connected to the output of the state register and the status signals. Note that each input is indicated as a *dot* in Figure 5 and introduces a capacitance load (C_{And}) for the state register output. The next state logic and the output logic are essentially a set of OR-gates, inputs of which are outputs of the decoder. Again, each input is indicated as a *dot* in Figure 5 and will introduce a capacitance load (C_{Or}) for the AND-gates of the decoder.

The dots in next state logic and output logic forms two matrices: *next state matrix* and *output logic matrix*. The rows of the matrices correspond to the decoder outputs, which in turn correspond to a state tuple in the state action table. The

$\vec{t}$	$\vec{S}$	$\vec{C}$	$\vec{NS}$	$\vec{D}$	$\vec{O}$
$\vec{t}_1$	00	0	00	10000000	110000000
$\vec{t}_2$	00	1	01	01000000	100101001
$\vec{t}_3$	01	0	10	00100000	001100101
$\vec{t}_4$	01	1	10	00010000	010001001
$\vec{t}_5$	10	0	11	00001000	001000010
$\vec{t}_6$	10	1	11	00000100	000000010
$\vec{t}_7$	11	0	00	00000010	000010000
$\vec{t}_8$	11	1	01	00000001	000000001

Figure 6: The Activity Vectors

columns of the matrices correspond to the next state signals and output signals respectively. The role of the dots in power analysis of the controller is two-fold: (1) Since each dot introduces a capacitance of size C_{Or}, the number of dots along each row gives the total load of corresponding state decoder AND-gate. (2) The dots along each column indicates a true value of the corresponding signal. Note that distribution of the dots at each row correspond exactly to the value of the state tuple in the state action table.

The activity of the controller at state $\vec{t}_i$ can be characterized by a set of activity vectors, namely the *current state vector* $\vec{S}_i$, the *next state vector* $\vec{NS}_i$; the *decoder vector* $\vec{D}_i$; and the *output vector* $\vec{O}_i$. Each activity vector correspond to the output of state register, status signals, next state logic, decoder and output logic respectively. Figure 6 shows the values of these vectors at each state for the example shown in Figure 5. It is obvious that

$$\vec{O} = \vec{FU} \# \vec{Reg} \# \vec{Drv}$$

Each bit of the activity vector $\vec{V}$ (could be one of $\vec{S}, \vec{NS}, \vec{D}, \vec{O}$) is associated with a capacitance. The capacitances for all the bits also form a *capacitance vector*, denoted as $\vec{C}_L = (C_{L0}, C_{L1}, ...)$. The energy consumed at state i can then be measured as $(\vec{V}_i \oplus \vec{V}_{i+1}) \cdot \vec{C}_L \times V_{DD}^2$ The total energy consumed for the entire state trace on this vector can be computed as

$$\sum_{\vec{t}_i \in ST} (\vec{V}_i \oplus \vec{V}_{i+1}) \cdot \vec{C}_L \times V_{DD}^2$$

Based on this model, we will identify the capacitance vector as well as activity vector for each part of the controller.

State Register and Next State Logic

Since for $\vec{t}_i, \vec{t}_{i+1} \in ST$, we always have $\vec{NS}_i = \vec{S}_{i+1}$. The switching activities of the state vector and next state vector are the same, so we treat them together.

The capacitance of each bit of the state register consists of its (1) internal capacitances and (2) the output loads due to its fanout to the state decoder. The capacitance of each bit of the next state logic is the input capacitance of the state register. The capacitances mentioned above are the same for each bit, so we denote their sum as C_{Reg}, and the corresponding capacitance vector becomes $C_{Reg} \times \vec{I}$, where $\vec{I} = (1, 1, ..., 1)$ is the unit vector.

the total energy consumption of the state register and next state logic can then be computed as

$$E(SR) + E(NS) = C_{Reg} \times V_{DD}^2 \times \sum_{\forall \vec{t}_i \in ST} ((\vec{S}_i \oplus \vec{NS}_i) \cdot \vec{I})$$

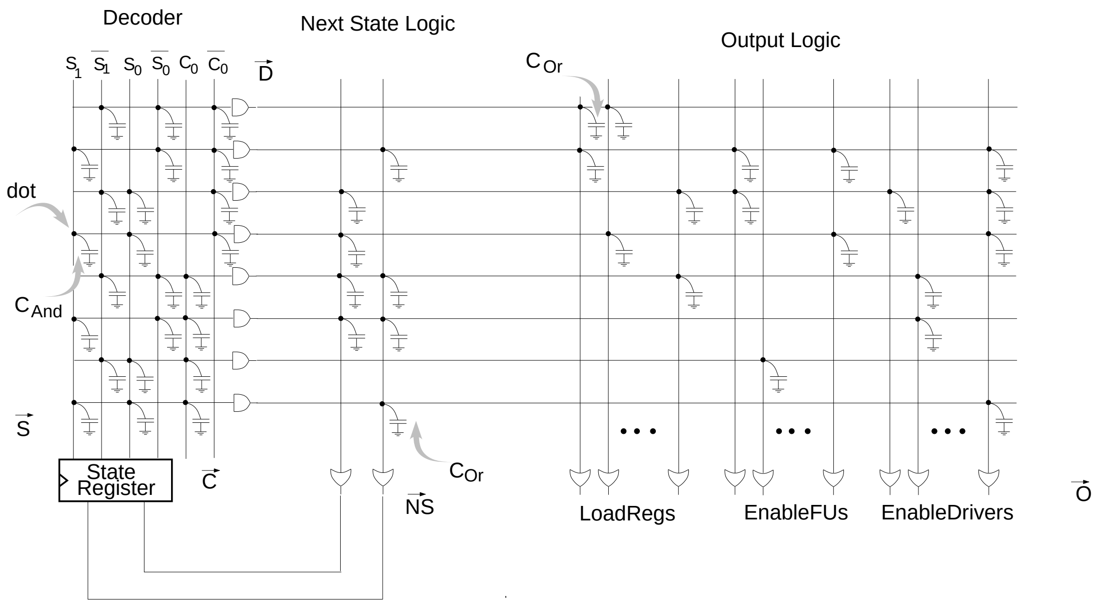

Figure 5: Controller

$$=C_{Reg} \times V_{DD}^2 \times \sum_{\forall \vec{t_i} \in ST} |\vec{S_i} \oplus \vec{NS_i}| \tag{5}$$

Decoder

The switching activity of the decoder is elegantly simple to analyze. At every state $\vec{t_i} \in ST$, only the output of corresponding AND-gate is 1. In other words, at every state, exactly two AND-gates will switch: The gate corresponds to previous state will switch from 1 to 0; the gate corresponds to current state will switch from 0 to 1, and the rest of the gates will remain unchanged.

The capacitance of each AND-gate in the state decoder is determined by its fanout, that is, how many dots along the row in Figure 5. If we assume *each input of the OR-gates introduces the same capacitances as* C_{Or}, the ith bit of the capacitance vector $\vec{C_L}$ can be computed as $\#dots(row_i) \times C_{Or}$, where $\#dots(row_i)$ can be computed as $|\vec{NS_i} \# \vec{O_i}|$.

Due to the "one-hot" property of the activity vector D, the energy consumed on the decoder can then be computed by counting the number of dots along the rows.

$$E(Decoder) = 2 \times C_{Or} \times V_{DD}^2 \times \sum_{\forall \vec{t_i} \in ST} |\vec{NS_i} \# \vec{O_i}| \tag{6}$$

Output Logic

The activity vector of the output logic is $\vec{O} = \vec{FU}_i \# \vec{Reg}_i \# \vec{Drv}_i$. If we denote the capacitance vector as $\vec{C_O}$, then the energy consumed on the output logic is:

$$Energy(OutputLogic) = V_{DD}^2 \times \sum_{\vec{t_i} \in ST} (\vec{O_i} \oplus \vec{O}_{i+1}) \cdot \vec{C_O} \tag{7}$$

3.4 RT Level Power Estimation

Branching Probability and Execution Frequency

In the previous section, we develop a set of formula for power estimation of a state trace. However, the state trace information is not available in general. We resort to profiling techniques to obtain *branching probability function* $Prob(i,j)$ defined for every pair of tuples $(\vec{t_i}, \vec{t_j})$ in the state action table SAT.

The *execution frequency* of a state tuple in SAT is defined as the expected number of times the state tuple will be executed. The execution frequency can be obtained either from the profiling tool or directly from the branching probability function. Given the branching probability function, the determination of execution frequency of each state tuple can be formulated as solving a set of linear equations with the form

$$Freq(\vec{t_j}) = \sum_{\forall \vec{t_i} \in SAT} Freq(\vec{t_i}) \times Prob(i,j)$$

for $\forall \vec{t_j} \in SAT$. Solution can then be obtained through standard procedures such as Gaussian elimination or LU factorization.

Formula

The formula developed in the previous section can then be rewritten by inspecting the state tuples in SAT one by one. In other words, the power metrics can be measured as the sum of the corresponding metrics of all the state tuples weighted by their execution frequencies.

$$Cycles = \sum_{\vec{t_i} \in SAT} Freq(\vec{t_i}) \tag{8}$$

$$E(Clock) = 2 \times C(Clock) \times V_{DD}^2 \times Cycles \tag{9}$$

$$E(Datapath) = V_{DD}^2 \times \sum_{\vec{t_i} \in SAT} Freq(\vec{t_i}) \times (\vec{DP}_i \cdot \vec{C}_{DP}) \tag{10}$$

$$E(SR) + E(NS) = C_{Reg} \times V_{DD}^2 \times \sum_{\vec{t_i} \in SAT} Freq(\vec{t_i}) \times \sum_{\vec{t_j} \in SAT} Prob(i,j) \times |\vec{S_i} \oplus \vec{S_j}| \tag{11}$$

$$E(Decoder) = 2 \times C_{Or} \times V_{DD}^2 \times \sum_{\vec{t}_i \in SAT} Freq(\vec{t}_i) \times |\vec{NS}_i \# \vec{O}_i| \qquad (12)$$

$$E(OutputLogic) = V_{DD}^2 \times \sum_{\vec{t}_i \in SAT} Freq(\vec{t}_i) \times \sum_{\vec{t}_j \in SAT} Prob(i,j) \times ((\vec{O}_i \oplus \vec{O}_j) \cdot \vec{C}_O) \qquad (13)$$

4 Experimental Results

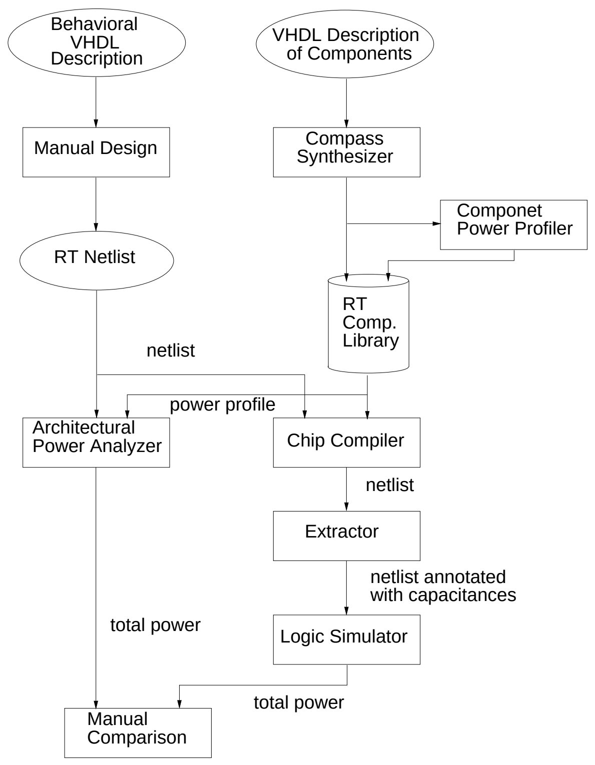

Figure 7: Block Diagram of the Experiment

In order to evaluate the estimation tool, we applied it to a set of well known benchmarks [HW92]. Figure 7 shows the block diagram of the experiment.

The component library was built by feeding functional VHDL description of each component to COMPASS ASIC Synthesizer. The synthesized components were then fed into the component power profiler [Ag95] to obtain an average power for each component. The average component power was stored in the library. The RT level design of each benchmark was manually synthesized from behavioral VHDL description. Assuming architectural model in Section 3.1, the power estimation of each benchmark was obtained by applying equations 8-13 in Section 3.4. We are able to obtain the average power of each benchmark in a couple of seconds on a Sparc 5 station.

The RT level VHDL description of each benchmark instantiating the components in the same library was also fed into the COMPASS chip compiler to obtain the layout. Netlists annotated with node capacitances were then extracted from the layout. Logic simulation assuming random input was invoked to obtain the total switched capacitances.

We compare the estimated results of datapath and controller with the measured results obtained from the layout in Table 1 and Table 2 respectively. The columns of the tables show the estimated switched capacitance for different classes of components (such as the functional units (FU), registers (Reg), buses (Bus), bus drivers (Drv), clock (Clk), state register (SR), next state logic (NS), decoder (Dec), output logic (Output)), the total estimated switched capacitance, the measured switched capacitance, and the error computed as $\frac{|measured - estimated|}{measured}$. The rows of the tables correspond to different benchmarks.

5 Conclusions

The described power estimation technique which is statistical in nature at the component level, and analytical at the RT level, offers fast feedback for high level exploration tools. Experiments on standard benchmarks show that the average error of the datapath is 5% and the controller is 7%. Our future work will extend this technique to the behavioral level.

	FU	Reg	Bus	Drv	Clk	Total	Measured	Err
HAL	27.14	11.40	37.64	6.24	19.2	101.62	100.5	1%
DCT	49312	3108	1372	846	16452	71091	67834	4%
SRA	116.1	16.7	36.2	9.9	23.3	202.2	208.4	-2%
ELL	517.7	51.6	121.14	28.1	328.4	1046.9	933.4	12%

Table 1: Switched Capacitance of the Datapath

	SR,NS	Dec	Output	Clk	Total	Measured	Err
HAL	3.86	2.71	21.28	17.92	45.77	49.00	-6%
DCT	8707.2	1212.5	10148.9	17699.0	37767.5	39273.2	-3%
SRA	10.0	4.3	25.7	29.4	69.8	76.0	-8%
ELL	42.3	13.068	95.85	185.6	336.8	384.2	-12%

Table 2: Switched Capacitance of the Controller

6 References

[Ag95] P. Agrawal, D. Gajski, F. Kurdahi, "Component Power Profiler (CPP)", TR-ICS-95-x, UC, Irvine

[Ga92] D. Gajski, N. Dutt, A. Wu, S. Lin, *High Level Synthesis: Introduction to Chip and System Design*, Kluwer, 1992

[Ga94] D. Gajski, F. Vahid, S. Narayan, J. Gong, *Specification and Design of Embedded Systems*, Prentice Hall, 1994

[Go93] J. Gong, D. Gajski, S. Narayan, "Software Estimation from Executable Specifications", TR-ICS-93-5, UC, Irvine

[HW92] *Benchmarks for the Sixth International Workshop on High-Level Synthesis*, 1992.

[La94] P. Landman, J. Rabaey, "Black-Box Capacitance Models for Architectural Power Analysis", *International Workshop on Low Power Design*, Napa Valley, CA, April 1994

[Me94] R. Mehra, J. Rabaey, "Behavioral Level Power Estimation and Exploration", *International Workshop on Low Power Design*, Napa Valley, CA, April 1994

[Na94] F. Najm, "A Survey of Power Estimation Techniques in VLSI Circuits", *IEEE Transaction on VLSI Systems*, pp.446-455, Dec., 1994,

[Na95] F. Najm, "Feedback, Correlation, and Delay Concerns in the Power Estimation of VLSI Circuits", *Proceedings of the Design Automation Conference*, pp. 612-617, 1995

[We93] N. H.E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design: A System Perspective*, Second Edition, Addison-Wesley, 1993