Techniques for the Power Estimation of Sequential Logic Circuits Under User-Specified Input Sequences and Programs

José Monteiro Srinivas Devadas
Department of EECS
MIT, Cambridge, MA 02139

We describe an approach to estimate the average power dissipation in sequential logic circuits under user-specified input sequences or programs. This approach will aid the design of programmable controllers or processors, by enabling the estimation of the power dissipated when the controller or processor is running specific application programs.

Current approaches to sequential circuit power estimation are limited by the fact that the input sequences to the sequential circuit are assumed to be uncorrelated. In reality, the inputs come from other sequential circuits, or are application programs.

In this paper we show how user-specified sequences and programs can be modeled using a finite state machine, termed an input-modeling finite state machines or IMFSM. Power estimation can be carried out using existing sequential circuit power estimation methods on a cascade circuit consisting of the IMFSM and the original sequential circuit.

I. Introduction

Average power dissipation estimation is an important problem that has become more relevant with the growing need for low-power electronic circuits. A comprehensive review of existing power estimation techniques is presented in [6]. Most of the power estimation techniques available today are restricted to combinational circuits, i.e., circuits without memory. Exact approaches and efficient approximate approaches to sequential power estimation have been presented recently [9].

One of the limitations of the approach of [9] is that the input sequences to the sequential circuit are assumed to be uncorrelated. In reality, the inputs come from other sequential circuits, or are application programs. A high degree of correlation could exist in the applied input sequence. This correlation could be temporal, i.e., consecutive vectors could bear some relationship, or could be spatial, i.e., bits within a vector could bear some relationship.

In this paper, we describe an approach to estimate the average power dissipation in sequential logic circuits under user-specified input sequences or programs. Both temporal and spatially correlated sequences can be modeled using a finite state machine, termed an input-modeling finite state machine or IMFSM. Power estimation can be carried out using existing sequential circuit power estimation methods (e.g., [9]) on a cascade circuit consisting of the IMFSM and the original sequential circuit.

Our techniques are applicable to estimating the switching activity, and therefore power dissipation, of processors running application programs. We do not, however, model the power dissipated in external memory (e.g., DRAM, SRAM), or caches. Our approach is useful in the architectural and logical design of programmable controllers and processors, because it enables the accurate evaluation of power dissipated in a controller or processor, when specific application programs are run.

Recent work in power analysis of embedded software [8] uses a different approach to estimate the power dissipated by a processor when a given program is run on the processor. An instruction-level energy model has been developed, and validated on the 486DX2. The advantages of this approach are that it is efficient and quite accurate and can take into account the power dissipated in the entire system, i.e., processor + memory + interconnect. A disadvantage is that each different architecture or different instruction set requires a significant amount of empirical analysis on implemented hardware to determine the base cost of individual instructions.

The model we use to relate switching activity to power dissipation is briefly described in Section II. In Section III we briefly describe the approach to sequential power estimation originally described in [9]. In Sections IV-A, IV-B and V, we describe how completely-specified input sequences, incompletely-specified input sequences and assembly programs, respectively, can be modeled using IMFSMs. Preliminary experimental results are presented in Section VI.

II. A Power Dissipation Model

Under a simplified model, the energy dissipation of a CMOS circuit is directly related to the switching activity.

In particular, the three simplifying assumptions are:

- The only capacitance in a CMOS logic-gate is at the output node of the gate.
- Either current is flowing through some path from V_{DD} to the output capacitor, or current is flowing from the output capacitor to ground.
- Any change in a logic-gate output voltage is a change from V_{DD} to ground or vice-versa.

All of these are reasonably accurate assumptions for well-designed CMOS gates [4], and when combined imply that the energy dissipated by a CMOS logic gate each time its output changes is roughly equal to the change in energy stored in the gate's output capacitance. If the gate is part of a synchronous digital system controlled by a global clock, it follows that the average power dissipated by the gate is given by:

$$P_{avg} = 0.5 \times C_{load} \times (V_{dd}^2/T_{cyc}) \times E(transitions) \quad (1)$$

where P_{avg} denotes the average power, C_{load} is the load capacitance, V_{dd} is the supply voltage, T_{cyc} is the global clock period, and $E(transitions)$ is the *expected value* of the number of gate output transitions per global clock cycle [5], or equivalently the average number of gate output transitions per clock cycle. All of the parameters in (1) can be determined from technology or circuit layout information

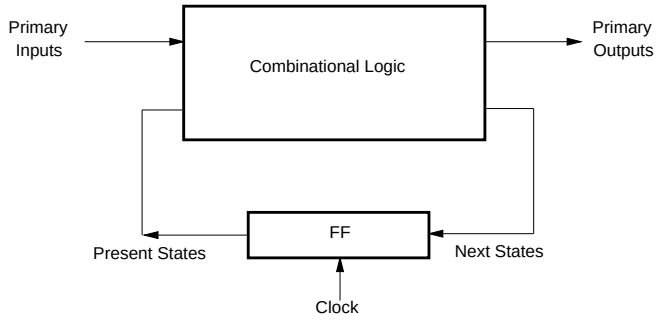

Fig. 1. A Synchronous Sequential Circuit

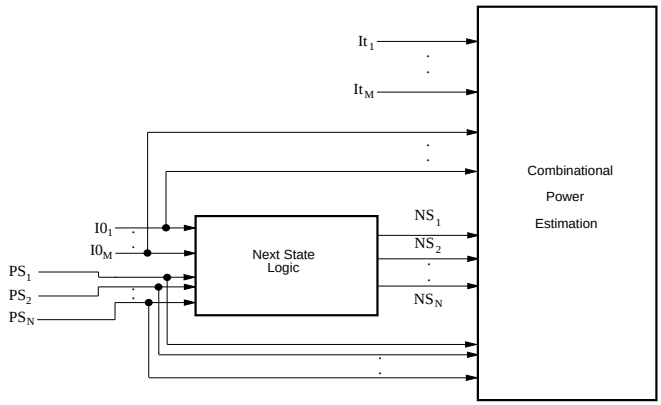

Fig. 2. Modeling Correlation in a Sequential Circuit

except $E(transitions)$, which depends on both the logic function being performed and the statistical properties of the primary inputs.

Equation (1) is used by the power estimation techniques such as [3], [5] to relate switching activity to power dissipation.

III. A Strategy for Sequential Circuit Power Estimation

The sequential logic estimation techniques summarized here were originally presented in [9].

Power and switching activity estimation for sequential circuits is significantly more difficult than combinational circuits because the probability of the circuit being in any of its possible states has to be computed. As an example, consider the sequential circuit of Figure 1. When a vector pair $\langle v_1,\ v_2\rangle$ is applied to the combinational logic, it is composed of a primary input part and a present state part, namely $\langle i_1@s_1,\ i_2@s_2\rangle$. Given $i_1@s_1$, the next state s_2 is uniquely determined by the functionality of the combinational logic. This *correlation* between the vector pairs has to be taken into account in accurate, sequential switching activity estimation.

A. Modeling Correlation

To model the correlation between two vectors in a sequential circuit, combinational estimation methods (e.g., [3], [5]) have to be augmented. This augmentation is summarized in Figure 2.

The combinational logic power estimator receives two sets of inputs, namely $\langle I0,\ It\rangle$ for the primary inputs and $\langle PS,\ NS\rangle$ for the present state lines. Given $I0$ and PS, NS is determined by the functionality of the combinational logic. This is modeled by the next state logic.

The configuration of Figure 2 implies that the switching activity can be determined given the vector pair $\langle I0,\ It\rangle$ for the primary inputs and PS for the state lines. Therefore, to compute the average switching activity, we require the transition probabilities for the primary inputs and the static probabilities for the present state lines.

B. State and Line Probability Computation

The static probabilities for the present state lines marked PS in Figure 2 are also correlated. Knowledge of *present state probabilities* as opposed to present state line (PS) probabilities is required. The state probabilities depend on the connectivity of the State Transition Graph (STG) of the circuit and can be computed using the Chapman-Kolmogorov equations for discrete-time Markov Chains [7]. However, this can be very expensive.

An efficient and accurate approximation strategy is to ignore this correlation and directly determine present state line probabilities [9]. These probabilities are directly computed by solving a nonlinear system of equations obtained from the next state logic equations.

Experiments on a wide variety of benchmarks indicate that the approximation methods are accurate to within 5%.

C. Primary Input Probability Assumption

In the experiments of [9], it was assumed that the transition probabilities of the primary inputs were *given* and that the primary inputs were uncorrelated. This is not a tenable assumption when the sequential circuit is embedded within a larger circuit and/or receives inputs from an instruction memory. In the next two sections we will describe how the transition probabilities of the primary inputs and correlation between primary inputs can be modeled using input-modeling finite state machines (IMFSMs).

IV. Input Sequences

We consider the problem of estimating power dissipation of a sequential circuit when completely-specified or incompletely-specified sequences are applied to the circuit.

A. Completely-Specified Input Sequences

Assume that we are given a sequential circuit M. We consider the problem of estimating the average power dissipation in M upon the application of a periodic completely-specified input sequence C. An easy way of doing this is to perform timing simulation on the circuit for the particular vectors, and measure the activities at each gate. However, this will become very time-consuming for incompletely-specified vector sequences.

Given the input sequence $C = \{c_1,\ c_2,\ \ldots,\ c_N\}$, we specify the State Transition Graph (STG) of an autonomous input-modeling finite state machine (IMFSM), call it A, as follows. A has N states, s_1 through s_N. For $1 \leq i < N$ we have a transition from s_i to s_{i+1}. We also have a transition from s_N to s_1. A is a Moore machine, and the output associated with each state s_i is the corresponding completely-specified vector c_i. An example of a four-vector sequence with each vector completely-specified over three bits is given in Figure 3(a), and the STG of the derived IMFSM is shown in Figure 3(b).

A logic-level implementation of A can be obtained by arbitrarily assigning distinct codes to the states $s_i,\ 1 \leq i \leq N$, using $\lceil \log_2 N \rceil$ bits. The encoding does not affect the power estimation step.[1]

[1] We will ignore any switching activity or power dissipation in A during the estimation step. Hence the encoding does not affect the estimation of the power dissipated in M.

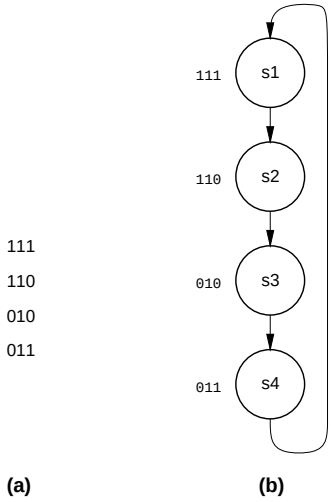

Fig. 3. Example of Autonomous IMFSM for a Four-Vector Sequence

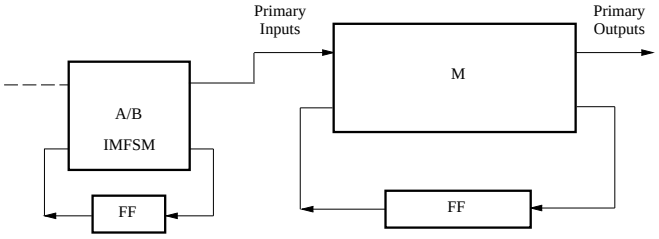

Fig. 4. Cascade of IMFSM and Given Sequential Circuit

In order to estimate the average power dissipated in M upon the application of a given completely-specified input sequence C, the power estimation strategies reviewed in Section III are applied to the cascade $A \rightarrow M$ depicted in Figure 4. Since the cascade $A \rightarrow M$ does not have any external inputs, no assumptions regarding their probabilities need to be made (cf. Section III-C).

B. Incompletely-Specified Input Sequences

We consider the problem of estimating the average power dissipation in M upon the application of a periodic incompletely-specified input sequence I. By incompletely-specified we mean that the unspecified inputs can take on either the 0 or 1 value with known probability.

As an example, consider the incompletely-specified sequence

```
11-
-1-
-01
-11
```

Completely-specified sequences that can possibly be applied to M are

```
110  111  110  110
010  111  010  111
001  101  001  001
011  111  011  011
```

among many others.

We are given the input sequence $D = \{d_1,\ d_2,\ \ldots,\ d_N\}$, over inputs $p_1,\ p_2,\ \ldots,\ p_M$. We will assume that the - entries for any p_j are uncorrelated. The - entries for each p_j have a user-specified probability of being a 1 denoted by $prob(p_j = 1)$.

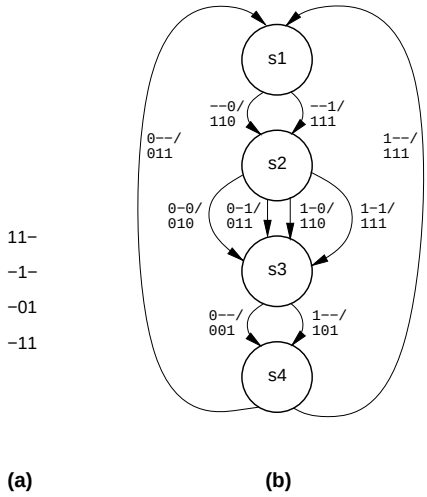

Fig. 5. Example of Mealy IMFSM for a Four-Vector Sequence

We specify the State Transition Graph (STG) of an input-modeling finite state machine (IMFSM), call it B, as follows. B has N states, s_1 through s_N, M primary inputs $p_1,\ p_2,\ \ldots,\ p_M$, and M primary outputs $o_1,\ o_2,\ \ldots,\ o_M$. For $1 \leq i < N$ we have a transition from s_i to s_{i+1} regardless of the values of the p_j's. We also have a transition from s_N to s_1 regardless of the values of the p_j's. However, B is a Mealy machine, and the output associated with each transition $s_i \rightarrow s_{i+1}$ is a logical function dependent on the corresponding d_i. An example of the incompletely-specified four-vector sequence used above is reproduced in Figure 5(a), and the STG of the derived IMFSM is shown in Figure 5(b). Since $d_1 =$ `11-`, we have $o_1 =$ `1`, $o_2 =$ `1` and $o_3 = p_3$ for the transition from s_1. Similarly for the other transitions.

As before, a logic-level implementation of B can be obtained by arbitrarily assigning distinct codes to the states $s_i,\ 1 \leq i \leq N$, using $\lceil \log_2 N \rceil$ bits. The encoding does not affect the power estimation step.

In order to estimate the average power dissipated in M upon the application of a given incompletely-specified input sequence C, the strategies reviewed in Section III are applied to the cascade $B \rightarrow M$. The given static or transition probabilities $prob(p_j = 1)$ of the primary inputs $p_1,\ p_2,\ \ldots,\ p_M$ to B are used to estimate the power. Note that the probabilities for all inputs to M are automatically derived.

V. Input Assembly Programs

In many applications, a processor receives a set of instructions as an input. An important problem is to estimate the power dissipated in the processor when it runs a given application program or a set of application programs. In this section, we describe ways of modeling an input assembly program as a IMFSM so conventional sequential estimation methods can be used.

For this purpose we will focus on a simple instruction set for a RISC processor α_0, which is a subset of the instruction set for the DEC-Alpha^{TM} microprocessor. Table I gives a description of the α_0 instruction set.

Given an arbitrary α_0 program, we will derive a logic-level IMFSM B which is cascaded with the processor as illustrated in Figure 4 to estimate average power consumption when the program runs

Format	$\langle 31:26\rangle$	$\langle 25:21\rangle$	$\langle 20:16\rangle$	$\langle 15:13\rangle$	$\langle 12\rangle$	$\langle 11:5\rangle$	$\langle 4:0\rangle$
Operate	Opcode	R_a	R_b	000	0	Function	R_c
Operate with Literal	Opcode	R_a	$Literal$		1	Function	R_c
Memory	Opcode	R_a	R_b	$disp.m$			
Branch	Opcode	R_a	$disp.b$				

Instruction	Opcode	Function	Operation
add	0x10	0x20	$R_c \leftarrow \langle R_a\rangle + \langle R_b\rangle \vert Lit$
and	0x11	0x00	$R_c \leftarrow \langle R_a\rangle \wedge \langle R_b\rangle \vert Lit$
or	0x11	0x20	$R_c \leftarrow \langle R_a\rangle \vee \langle R_b\rangle \vert Lit$
sll	0x12	0x39	$R_c \leftarrow \langle R_a\rangle\ SLL\ \langle R_b\rangle \vert Lit_{5:0}$
srl	0x12	0x34	$R_c \leftarrow \langle R_a\rangle\ SRL\ \langle R_b\rangle \vert Lit_{5:0}$
sub	0x10	0x29	$R_c \leftarrow \langle R_a\rangle - \langle R_b\rangle \vert Lit$
xor	0x11	0x40	$R_c \leftarrow \langle R_a\rangle \oplus \langle R_b\rangle \vert Lit$
cmpeq	0x10	0x2D	if $\langle R_a\rangle = \langle R_b\rangle$, $R_c \leftarrow 1$, else $R_c \leftarrow 0$
cmple	0x10	0x6D	if $\langle R_a\rangle \leq \langle R_b\rangle$, $R_c \leftarrow 1$, else $R_c \leftarrow 0$
cmplt	0x10	0x4D	if $\langle R_a\rangle < \langle R_b\rangle$, $R_c \leftarrow 1$, else $R_c \leftarrow 0$
ld	0x29		$EA \leftarrow \langle R_b\rangle + SEXT(disp.m),\ R_a \leftarrow MEMORY[EA]$
st	0x2D		$EA \leftarrow \langle R_b\rangle + SEXT(disp.m),\ MEMORY[EA] \leftarrow \langle R_a\rangle$
br	0x30		$R_a \leftarrow PC,\ PC \leftarrow \langle PC\rangle + 4 \cdot SEXT(disp.b)$
bf	0x39		Update $PC,\ EA \leftarrow \langle PC\rangle + 4 \cdot SEXT(disp.b)$, if $\langle R_a\rangle = 0,\ PC \leftarrow EA$
bt	0x3D		Update $PC,\ EA \leftarrow \langle PC\rangle + 4 \cdot SEXT(disp.b)$, if $\langle R_a\rangle \neq 0,\ PC \leftarrow EA$

TABLE I
α_0 INSTRUCTION SET

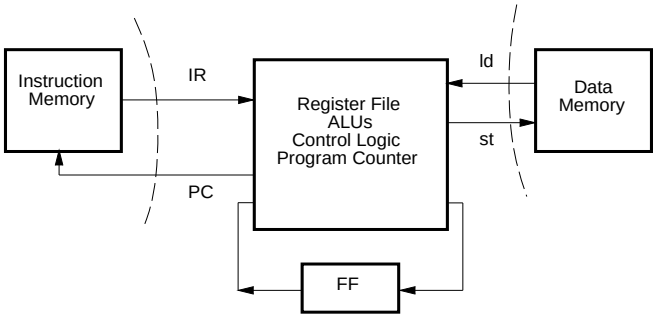

Fig. 6. Processor Model

on the processor. Our model for the processor is illustrated in Figure 6. The processor is a sequential circuit consisting of a register file, arithmetic units, and control logic. It receives as input an instruction stream and reads and writes an external memory.

A key assumption that we make is that data values loaded *from* memory are random and uncorrelated. Therefore, the effect of a sequence of loads to, and stores from the same location in memory is not modeled. If we did not make this assumption then we would have to deal with the entire state space of the memory – a very difficult task. Note that in this paper we are also not concerned with the power dissipated in the external memory.

We will now describe how to generate a IMFSM given an arbitrary program comprised of a sequence of assembly instructions. Let the program P be a sequence of instructions $P = \{r_1,\ r_2,\ \ldots,\ r_N\}$. The STG of the Moore IMFSM Q has N states. For each of the different classes of instructions in Table I we show how to derive the STG of Q.

- Operate: If r_i is an Operate instruction (e.g., add, cmplt) we assign r_i as the output of state s_i. s_i makes an unconditional transition to s_{i+1}.
- Branch: If r_i is a branch instruction, we determine the branch target instruction, call it r_j. State s_i makes a transition to state s_j if variable $v_i = 1$, and a transition to state s_{i+1} if $v_i = 0$. The probability of v_i being a 1 will be determined by preprocessing the program P as described later in the section. The output associated with s_i is r_i.
- Memory: If r_i is a Memory instruction, the output associated with s_i is r_i. On a load instruction (ld), R_a is loaded with a random value from memory. The inputs to the processor from memory will have certain probabilities associated with 0 or 1 values and we elaborate on this next. Since we are treating the data memory as an external memory, a store instruction (st) is essentially a null operation.

We now elaborate on the probabilities of the data inputs from memory and the probabilities of the branch variables (v_i's). Branch prediction is a problem that has received some attention in the compiler world [1]. The probabilities of the branch variable $v_i = 1$ corresponds to the probability that a branch is taken on the execution of instruction r_i, and this probability can be determined, at least approximately, by preprocessing the program P.

For example, if we have a constant iteration loop with N iterations, the probability of staying in the loop is computed as $\frac{N}{N+1}$ and the probability of exiting the loop as $\frac{1}{N+1}$. If comparisons between data operands are used to determine branch conditions, the probability of

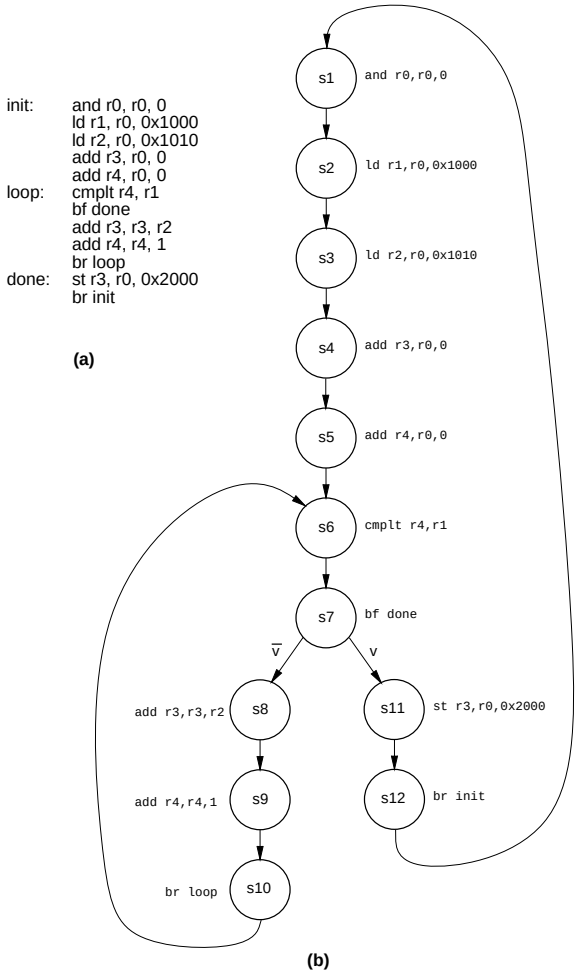

Fig. 7. Example of Mealy IMFSM for Assembly Program

the comparison evaluating to a 1 assuming random data operands can be calculated. For example, the probability that $a \geq b$ is 0.5, and the probability that $a + b > c$ is 0.75. These probabilities can be computed using Binary Decision Diagrams [2].

Additionally, we can run the program P with several different inputs, and obtain the information regarding the relative frequency with which each conditional branch is being taken versus not being taken. This relative frequency is easily converted into the probabilities for the v_i's.

As before once the STG of the IMFSM has been derived and encoded, estimation can be carried out using the topology of Figure 4. An example assembly program for the processor α_0 is given in figure 7(a) and the STG of its corresponding IMFSM is shown in figure 7(b). The average power dissipation of the processor when executing the program is computed by the estimation method.

VI. Preliminary Experiments

In this section we present preliminary experimental results obtained using the methods of Sections IV and V.

We will decouple ourselves from the particular combinational power estimation strategy used. As described in Section III, any strategy has to be able to:

1. model the correlation between applied vector pairs due to the next state logic as shown in Figure 2,
2. use present state probabilities or approximate using line probabilities, and
3. model the correlation in input sequences or programs while computing primary input probabilities.

Item 3 has been the focus of this paper.

In Table II we present power estimation results on sequential circuits, of three different types, small machines synthesized from State Transition Graph descriptions, larger controller circuits, and a small processor similar to the α_0. For each given sequential circuit or processor, assuming uniform primary input probabilities, we compute the power dissipation using the techniques of [9]. The power estimation values assuming a clock frequency of 20MHz, a supply voltage of 5V and a unit delay model are given in the column UNIFORM-PROB, together with the CPU time in seconds required for the computation on a DEC-AXP 3000/500. For the first type of circuits (for which we have a STG available) we built a transfer input sequence, i.e., an input sequence that will traverse all states in the STG. For all sequential circuits we generated a random input sequence. Given these input sequences we construct a IMFSM using the methods of Section IV. The corresponding power values and CPU time are given in columns IMFSM-TRANSFER-SEQ and IMFSM-RAND-SEQ respectively. Similarly, we use the techniques of Section V to obtain a IMFSM for 2 different input programs for the α_0 processor.

We compute the power dissipation of the cascade circuit consisting of the IMFSM driving the sequential circuit or processor (cf. Figure 4) using the techniques of [9].

In Table III we give percentage errors of the present line probabilities. For each sequential circuit and each random/transfer input sequence we compute the static probabilities of the present state lines and compare them with the static probabilities obtained by assuming uniform primary input probabilities. Under *min*/*max* columns we give the percentage error of the state line with minimum/maximum static probability error. Under *avg* we give the average error over all present state lines.

As can be seen from table II, the CPU time required to compute the power for the cascaded circuit is not much more than for the original circuit. However, the power estimation error for the first set of circuits can be as high as 44%, implying that the uniform probability assumption is unrealistic. Obtaining more accurate line probabilities allows the final combinational power estimation to be more accurate. Once accurate present state line probabilities have been computed a variety of methods can be applied to estimate the power dissipated in the logic.

For the processor example, huge errors occur. The first program is a simple program which does not cause any activity in the majority of the registers and in a large fraction of the combinational logic in the processor. The difference between the average power dissipated when this program is run, and when random inputs are assumed is therefore very high. The second program is more complex, and it causes greater activity and greater power dissipation. Note that for the input programs to the processors we have assumed a random distribution for data values, an assumption critiqued in Section VII.

VII. Conclusions, Limitations and Future Work

Average power dissipation estimation for sequential circuits is a difficult problem both from a standpoint of computational complexity, and from a standpoint of modeling the correlation due to feedback and correlation in input sequences. Previous approaches to sequential circuit power estimation are limited by the fact that the input sequences to the sequential circuit are assumed to be uncorrelated.

Circuit Name	#gate	#ff	UNIFORM-PROB		IMFSM-RAND-SEQ			IMFSM-TRANSFER-SEQ		
			power	cpu	power	% diff	cpu	power	% diff	cpu
bbtas	26	3	134	0.4	142	6.0	1.4	117	12.7	0.8
cse	136	4	454	13.5	473	4.2	15.5	510	12.3	15.6
keyb	174	5	587	17.5	479	18.4	23.4	577	1.7	21.7
kirkman	171	4	734	6.7	826	12.5	15.5	409	44.3	4.8
planet	333	6	2359	33.7	2158	8.5	129.7	2147	9.0	83.5
styr	318	5	1195	31.5	1175	1.7	46.6	1317	10.2	46.5
tbk	483	5	1835	81.7	1705	7.1	94.1	2084	13.6	101.0
train4	15	2	85	0.3	54	36.5	0.4	52	38.9	0.4
s298	119	14	441	2.5	331	24.9	8.1	N/A		
s444	181	21	411	6.7	348	15.3	17.8	N/A		
s526	193	21	529	5.3	423	20.0	13.9	N/A		
s713	393	19	1176	333.7	1096	6.8	513.0	N/A		
s1196	529	18	2674	174.2	2313	13.5	197.3	N/A		
α_0-prog1	144	75	965	4.3	N/A			26	97.5	13.4
α_0-prog2					N/A			918	4.9	59.1

TABLE II
COMPARISON OF POWER DISSIPATION UNDER UNIFORM INPUT ASSUMPTION AND IMFSM COMPUTATION

Circuit Name	IMFSM-RAND-SEQ			IMFSM-TRANSFER-SEQ		
	min	avg	max	min	avg	max
bbtas	8.1	15.7	22.4	7.3	20.3	31.7
cse	9.8	20.3	27.1	9.8	16.3	20.6
keyb	0.0	5.9	10.1	0.9	12.5	20.0
kirkman	26.9	39.6	49.3	27.1	39.7	49.3
planet	0.2	1.7	3.7	0.4	0.9	2.0
styr	8.0	20.2	29.7	13.8	19.0	22.5
tbk	1.3	3.7	5.4	0.4	12.6	17.7
train4	0.0	8.3	16.7	6.9	10.6	14.2
s298	0.0	4.9	9.5	N/A		
s444	0.0	1.6	7.4	N/A		
s526	0.0	2.9	12.8	N/A		
s713	0.0	3.3	18.3	N/A		
s1196	0.0	5.7	15.2	N/A		
α_0-prog1	N/A			0.0	2.3	49.2
α_0-prog2	N/A			0.0	0.2	1.5

TABLE III
PRESENT STATE LINE PROBABILITY ERRORS

We showed how user-specified sequences and programs can be modeled using a finite state machine, termed an input-modeling finite state machines or IMFSM. Power estimation can be carried out using existing sequential circuit power estimation methods on a cascade circuit consisting of the IMFSM and the original sequential circuit.

Given input sequences or programs, we need to keep the IMFSM description reasonably compact, in order to manage the computational complexity of estimation. This implies that we need to make certain assumptions, the primary one being that data values are assumed to be uncorrelated. This assumption can be relaxed by using empirical data for particular applications such as voice and video, and we are currently looking at methods to derive this information automatically. Finally, more work is required to improve the efficiency of available sequential power estimation methods.

VIII. ACKNOWLEDGEMENTS

This research was supported in part by the Advanced Research Projects Agency under contract DABT63-94-C-0053, in part by the Portuguese "Junta Nacional de Investigação Científica e Tecnológica" under project "Praxis" and in part by a NSF Young Investigator Award with matching funds from Mitsubishi Corporation.

REFERENCES

[1] A. Aho, R. Sethi, and J. Ullman. *Compilers Principles, Techniques and Tools*. Addison-Wesley, 1986.

[2] R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[3] A. Ghosh, S. Devadas, K. Keutzer, and J. White. Estimation of Average Switching Activity in Combinational and Sequential Circuits. In *Proceedings of the* 29^{th} *Design Automation Conference*, pages 253–259, June 1992.

[4] L. Glasser and D. Dobberpuhl. *The Design and Analysis of VLSI Circuits*. Addison-Wesley, 1985.

[5] F. Najm. Transition Density, A Stochastic Measure of Activity in Digital Circuits. In *Proceedings of the* 28^{th} *Design Automation Conference*, pages 644–649, June 1991.

[6] F. Najm. A Survey of Power Estimation Techniques in VLSI Circuits (*Invited Paper*). *IEEE Transactions on VLSI Systems*, 2(4):446–455, December 1994.

[7] A. Papoulis. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 3^{rd} edition, 1991.

[8] V. Tiwari, S. Malik, and A. Wolfe. Power Analysis of Embedded Software: A First Step Toward Software Power Minimization. *IEEE Transactions on VLSI Systems*, 2(4):437–445, December 1994.

[9] C-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. Despain, and B. Lin. Power Estimation for Sequential Logic Circuits. *IEEE Transactions on VLSI Systems*, 3(1), June 1995. to appear.