

# Compact Vector Generation for Accurate Power Simulation

<sup>†</sup>Shi-Yu Huang, <sup>✦</sup>Kuang-Chien Chen, <sup>†</sup>Kwang-Ting Cheng, and <sup>✦</sup>Tien-Chien Lee

<sup>†</sup>Dept. of Electrical & Computer Engineering  
Univ. of California, Santa Barbara  
Santa Barbara, CA 93106

<sup>✦</sup>Fujitsu Labs of America  
3350 Scott Blvd. Bldg. 34  
Santa Clara, CA 95054

## *abstract*

Transistor-level power simulators have been popularly used to estimate the power dissipation of a CMOS circuit. These tools strike a good balance between the conventional transistor-level simulators, such as SPICE, and the logic-level power estimators with regard to accuracy and speed. However, it is still too time-consuming to run these tools for large designs. To simulate one-million functional vectors for a 50K-gate circuit, these power simulators may take months to complete. In this paper, we propose an approach to generate a compact set of vectors that can mimic the transition behavior of a much larger set of functional vectors, which is given by the designer or extracted from application programs. This compact set of vectors can then replace the functional vectors for power simulation to reduce the simulation time while still retaining a high degree of accuracy. We present experimental results to show the efficiency and accuracy of this approach.

## 1. Introduction

For a CMOS circuit, power dissipation is caused by three major types of currents: leakage current, short-circuit current, and dynamic transition current. The leakage current (or static current) is smaller than the other two types of currents by several orders of magnitude, and thus, is usually ignored. The short-circuit current occurs whenever a path from  $V_{dd}$  to ground is conducted at a device. In some cases, short-circuit current cannot be ignored. However, only the transistor-level simulators with continuous-time modeling of the device can take this part into consideration [3,4]. Logic-level power estimators completely ignore the short-circuit current and focus on the dynamic transition current, which is well recognized as the most dominating factor of power dissipation in a CMOS circuit.

Dynamic transition power is strongly related to the transition density of each internal signal [1]. Several methods [5,6,7,8,9,10,11,12,13] have been proposed to estimate the transition density of each internal signal at the logic level. However, these logic-level approaches suffer from the drawback of inaccuracy.

The inaccuracy of logic-level approaches is caused by the following reasons: (1) These approaches assumed each logic toggle count represents a full swing of  $V_{ss}$  to  $V_{dd}$  or vice-versa. This assumption may not be true for glitches (partial voltage swing). (2) The power consumed by charge/discharge at internal nodes of complex CMOS gates is ignored. (3) Short-circuit current is ignored. (4) The toggle power consumed by glitches or hazards is sensitive to the accuracy of delay models.

Stat-of-the-art transistor-level power simulators (e.g., PowerMill [14]), which will be referred to as *power simulators* for the rest of the paper, are approximately 2 to 3 orders of magnitude faster than SPICE. On the other hand, their accuracy is much higher than that of the logic-level power estimators because glitches and short-circuit current are also considered. These tools strike a good balance between speed and accuracy, and are more suitable for estimating the power dissipation of a design when high accuracy is required.

For power simulators, a set of simulation vectors which can characterize the typical behavior of the circuit is required. Such simulation vectors may be generated from a program which analyzes the circuit's high-level model or be provided by the designer. Also, in the cases of processors, a good set of vectors could be derived from a set of target application programs. Usually this set of simulation vectors is very large, hence the time of running it on a power simulator is still prohibitively high. For instance, a rough estimation shows that simulating 1,000,000 vectors for a 50K gate may take up to 3 months to complete [3]. In these cases, a designer cannot afford to run a power simulator using the original set of simulation vectors. One simple solution to resolve this problem is to select a small subset from the original set of vectors for power simulation. However, such vectors do not represent the *average* behavior of the circuit. Power estimation based on such randomly selected vectors could produce biased results.

In this paper, we propose a new methodology for power estimation that combines the advantages of the logic-level approaches and the power-simulator-based approaches. We first use a logic simulator to simulate the entire set of functional vectors and derive the *transition profile* of the internal signals. We then *generate* a new and compact set of vectors that would produce an identical or similar transition profile as the original set of vectors. Then we use such compact set of vectors for transistor-level power simulation. Since the compact set mimics the original (and large) set of functional vectors, it can be regarded as a good representative of the *typical* operations from the viewpoint of power dissipation. The generated set of vectors is much smaller than the original set of functional vectors, and thus, the transistor-level power simulation time is reduced significantly.

---

\*This work was supported by the National Science Foundation under grant MIP-9503651, California MICRO and Fujitsu Labs of America.

The rest of this paper is organized as follows. In Section 2, we describe the logic-level power dissipation model used in our approach. In Section 3, we formally describe the problem of generating the compact set of vectors and propose an algorithm to solve this problem. In Section 4, we present the experimental results. Section 5 gives the concluding remarks.

## 2. Logic-level Power Measure

Almost all logic-level power analysis approaches use the following formula to model the power dissipation:

$$P_d = \sum_i f \cdot V_{dd}^2 \cdot (T_i \cdot C_i),$$

where  $i$  is the index of an internal

signal,  $f$  and  $V_{dd}$  are the clock rate and the supply voltage, and  $T_i$  and  $C_i$  are the transition density and loading capacitance, respectively. It has been reported that glitches could be an important factor of the total dynamic power dissipation in some cases. The amount of glitches varies with different delay models. No glitch is considered if a zero-delay model is used. To estimate the glitches more accurately, a variable delay model should be used at the cost of longer simulation time. According to a delay model, we first run a logic-level simulator for the given set of vectors and measure the transition density at each node of the circuit. For the rest of this paper, we denote the original set of simulation vectors given by the user as  $V_{\text{original}}$  and the new compact set of simulation vectors to be generated as  $S$ .

We assume that a delay model is selected beforehand and used implicitly whenever the logic-level simulation is mentioned for the rest of this paper. The transition density of a signal  $g$  obtained by logic simulation on a set of vector  $V$  is denoted as  $T(V, g)$ . For each signal, transition density with respect to  $V_{\text{original}}$  is referred to as the *desired transition density*.

**Definition 1: (Transition measure)** The transition measure with respect to a vector set  $V$  is defined as follows:

$$\Phi(V) = \sum_i (T(V, i) \cdot C_i),$$

where  $i$  is the index of an

internal signal,  $T(V, i)$  is the transition density established by  $V$ , and  $C_i$  is the loading capacitance, respectively.  $\square$

**Definition 2: (Model error)** For a vector set  $S$ , the model error associated with this set is defined as the difference between the transition measure of  $S$  and  $V_{\text{original}}$ , i.e.,

$$\Delta\Phi(S) = (\Phi(S) - \Phi(V_{\text{original}})). \quad \square$$

## 3. The Algorithm

### 3.1 Problem formulation

Our goal is to generate a compact set of vectors  $S$  that can match  $V_{\text{original}}$  according to the above definition of logic-level transition measure, i.e., to find a small set of vectors  $S$  such that the model error  $\Delta\Phi(S)$  is minimized. Our vector generation algorithm generates one vector at a time. During the iterative process, we denote the accumulated set of vectors generated so far as  $S_{\text{current}}$ . In the following, we first introduce some notations and then discuss an algorithm to solve this problem.

**Definition 3: (Transition momentum)** For a signal  $g$ , the transition momentum, denoted as  $T_{\text{mmt}}(g)$ , is the difference between the *desired transition density* and the transition density with respect to  $S_{\text{current}}$ , i.e.,

$$T_{\text{mmt}}(g) = T(V_{\text{original}}, g) - T(S_{\text{current}}, g). \quad \square$$

The value of a signal's transition momentum is within  $[-1, 1]$ . If a signal has a positive transition momentum, then its desired transition density is higher than the transition density established by  $S_{\text{current}}$ . In other words, it is "under-transitioned" so far and the vectors generated in the following iterations should increase its transition activity. On the other hand, if a signal has a negative transition momentum, then it is "over-transitioned". Fig. 1 shows the profiles of the *desired density* and current transition density for

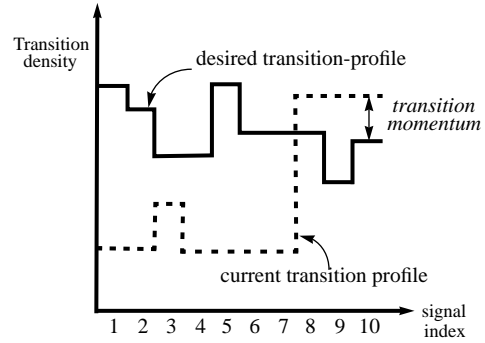


Fig 1: Illustration of the transition momentum during the process of generating power simulation vectors.

a set of internal signals. Signals 1-7 have positive transition momentums and signals 8-10 have negative transition momentums. To generate high quality vectors, the under-transitioned (with positive  $T_{\text{mmt}}$ ) signals should toggle in response to the next vector to reduce the overall model error. Similarly, the over-transitioned (with negative  $T_{\text{mmt}}$ ) signals should not toggle for the next vector. For each signal, the expected value for the next vector is a function of the transition momentum as well as the stable value at this signal produced by the last vector of  $S_{\text{current}}$ . In the following, we define signal momentum to reflect a signal's expected value for the next vector during the vector generation process.

**Definition 4: (Signal Momentum)** For a signal  $g$ , the signal momentum denoted as  $S_{\text{mmt}}(g)$ , is a number within  $[-1, 1]$ . It is computed by the following rules:

- (1) If the stable value at signal  $g$  produced by the last vector is '0', then  $S_{\text{mmt}}(g) = T_{\text{mmt}}(g)$ .
- (2) If the stable value at signal  $g$  produced by the last vector is '1', then  $S_{\text{mmt}}(g) = -T_{\text{mmt}}(g)$ .  $\square$

Intuitively, the signal momentum describes the expected value for a signal. If it is positive (negative), then the signal expects a '1' ('0') for the next vector. Once we have characterized the expected value for each signal, generating the next vector can be regarded as a search process for a vector that can produce the expected value for as many signals as possible. Since usually it is impossible to find a vector that produces expected values for all signals, priorities should be assigned. Signals with larger loading capacitance would have more influence on transition measure, therefore they should be assigned a higher priority. Also one of our goals is to match each internal signal's transition density, a signal with a larger absolute value of signal momentum ( $S_{\text{mmt}}$ ) should be given a higher priority in our algorithm. Therefore, to consider both effects, we use the product of a signal's momentum

and loading capacitance, i.e.,  $Weight(g) = (S_{mm}(g) * C_g)$  to reflect its priority during the search process of the next vector.

### 3.2 Backward weight propagation

Given a desired transition profile, we first compute the transition momentum and the signal momentum to decide the weight for each signal. After that, we use a technique, called backward weight propagation, to combine the weight of every signal to the primary inputs. Finally, we observe the combined weights at the primary inputs to decide an input pattern that can produce expected values at maximum number of signals. Therefore, in a single traversal of the circuit from the primary outputs towards the primary inputs, a new vector that attempts to achieve maximum reduction in the model error established by the current set of vectors is generated.

Consider a target signal that is visited during this back-propagation process. We add up the original weight of this signal with the weights propagated from each of its fanout signal(s). The resulting weight, referred to as *accumulated weight*, reflects the *combined* expectations at this signal and its fanout-cone. We further propagate this accumulated weight backward across the current gate/signal based on a rule derived from the following observations: Suppose the target node is an AND gate, and the accumulated weight is positive (expecting a '1' for the next vector). Because value '1' is the non-controlling value for an AND gate, all its fanins should be '1' to meet the expectation. Therefore, we assign this accumulated weight to each of its fanins. On the other hand, if the accumulated weight at this signal is negative (expecting a '0' for the next vector), then a '0' at any of its fanins will suffice to satisfy this condition because '0' is the controlling value for an AND gate. In this case, we *split* the accumulated weight evenly among its fanins. Similar rules can be derived for the other types of gates. We summarize the back-propagation rules as follows: (1) For a buffer: simply assign *weight* to its fanin. (2) For an inverter: simply assign ( $-weight$ ) to its fanin. (3) For an AND (OR) gate: if the expected value is the non-controlling value, then copy the accumulated weight to each fanin of the target signal. On the contrary, if the expected value is the controlling value, then divide the accumulated weight evenly among the target signal's fanins. The above discussion assumes that the network has been decomposed into simple primitive gates.

Once the accumulated weight reaches the primary inputs, we simply convert the weights into binary values by taking their signs. Since the generated vectors are for power simulation, we assume all flip-flops are controllable. That is, all outputs of flip-flops are regarded as primary inputs for the cases of sequential circuits.

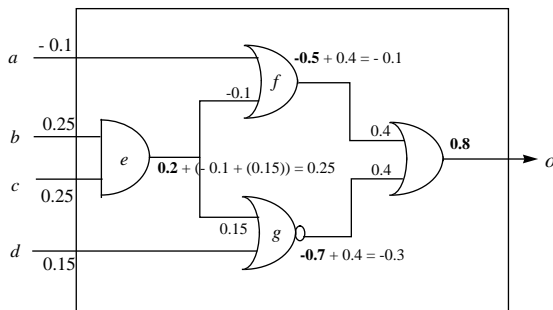


Fig. 2: An example to illustrate the backward weight propagation.

Fig. 2 shows an example of this back-propagation process. It starts from the primary output signal  $o$  whose weight is assumed to be 0.8 (computed by taking the product of the signal momentum and its loading capacity). Since this corresponds to an expected value of '1', which is a controlling value for an OR gate, the weight 0.8 is divided evenly and we assign 0.4 to its fanins  $f$  and  $g$ . Similar operations are executed for signals  $f$ ,  $g$ ,  $e$ . In Fig. 2, the notation  $w + w' = w''$  associated with each signal means that the individual weight derived before back-propagation is  $w$ , the weight that propagated from its fanout-cone is  $w'$ , and the final accumulated weight is  $w''$ . Signal  $e$  is an arbitration point because its two fanout branches have different expected values (-0.1 from the branch to  $f$  and 0.15 from the branch to  $g$ ). In this case, the fanout branch from  $e$  to  $g$  dominates, and thus, the expected value of signal  $f$  is sacrificed. Note that signal  $o$  tries to assert an expected value '1' by propagating a split weight 0.4 to both fanins  $f$  and  $g$ , and indeed this goal is satisfied by the fanin signal  $g$ . Therefore, the only signal that is not satisfied in this example is signal  $f$ . After all signals' weights are combined and propagated to the primary inputs, an input vector is decided by the rule of translating a weight to an expected value. For the example in Fig. 2, the vector generated is  $(abcd) = (0111)$ .

The complete algorithm of power vector generation is summarized as follows. Given a large set of original simulation vectors, we first derive its transition profile by a logic simulator using an appropriate delay model. Then we assign one starting vector and then iteratively generate the power vectors one at a time using the back propagation technique we just described. This algorithm continues until one of the following 2 stopping criteria is satisfied: (1) The model error is smaller than a user-specified threshold. (2) A limit on the number of generated vectors is reached.

## 4. Experimental Results

We implemented the proposed algorithm and tested it on a set of ISCAS89 benchmark circuits. We assume a fixed probability at each primary input to generate 1000 original vectors (the reason for generating only 1000 vectors as  $V_{original}$  will be discussed later). We regard this set of vectors as the original set of functional vectors. The desired transition profile of internal signals are derived by logic simulation. Then we run our program to generate a compact set of vectors. As mentioned earlier, for sequential circuits, we assume each flip-flop is fully controllable, and thus, we can explore the freedom of assigning values to the present state lines in our algorithm. We use a zero-delay model in the current experiment.

The results are shown in Table 1. It can be seen that the model errors can be reduced to below 2 or 3% with only 200 vectors for all the benchmark circuits. Note that power vector generation times listed in the last column (seconds on a Sun-Sparc5) are small as compared to the times of running PowerMill as shown in Table 2. Table 1 also shows the average of the absolute transition density error between the original set and the generated vectors, which is computed by  $\sum_i (|T(S, i) - T(V_{original}, i)|) / n$ , where  $n$  is the total number of signals. It is only slightly higher than the model error.

To run PowerMill, we translate each primitive gate to its transistor level counterpart using standard device parameters. For example, an AND gate is converted to a NOR gate with inverting

input signals. The width and length dimensions of p-channel and n-channel transistors used are (1000, 100) and (500, 100)  $\mu\text{m}$ , respectively.

We run PowerMill on these 200 vectors for each circuit. For comparison, we also run PowerMill on the original vectors. Note that in this experiment we only generate 1000 vectors as the reference vector set. We have also generated a much larger set of vectors and found that our program produces very similar results to those presented in Table 1. However, due to the long simulation time of PowerMill, we could not afford to simulate such long vector set for comparison. Table 2 shows the results of running PowerMill on  $V_{\text{original}}$  (1000 vectors) and  $S$  (200 vectors). The average ground current are obtained by assuming the clock rate is 1MHz. The last column shows the ratio of current obtained by running PowerMill on  $S$  versus that obtained by running PowerMill on  $V_{\text{original}}$ , i.e.,

$$\text{ratio} = I_{\text{gnd}}(S) / I_{\text{gnd}}(V_{\text{original}}).$$

Table 1: Results of generating 200 power vectors.  
(The size of  $V_{\text{original}}$  is 1000)

Circuit	Model error	abs. transition error	Time (seconds)
s1196	0.4%	2.0%	66
s1238	1.3%	2.5%	78
s1423	1.5%	2.5%	78
s1488	1.2%	3.5%	162
s1494	0.9%	3.6%	112
s5378	2.2%	3.2%	170
s9234	1.5%	2.4%	290
s13207	0.8%	1.2%	682
s15850	1.3%	1.5%	770
s35932	1.1%	4.4%	2006
s38417	1.3%	3.7%	1567
s38584	1.1%	2.8%	1910
Average	1.2%	2.8%	---

Table 2: Results of running PowerMill.  
( $S$ : compact set)  
( $V_{\text{orig}}$ : original set)

Circuit	$I_{\text{gnd}}(V_{\text{orig}})$ (mA)	$I_{\text{gnd}}(S)$ (mA)	ratio ( $S/V_{\text{orig}}$ )	Time ( $V_{\text{orig}}$ ) (sec)	Time ( $S$ ) (sec)
s1196	0.052	0.052	1.01	773	162
s1238	0.050	0.051	1.02	737	156
s1423	0.047	0.050	1.06	601	134
s1488	0.053	0.052	0.97	713	141
s1494	0.188	0.187	0.99	2413	489
s5378	0.874	0.848	0.97	1253	266
s9234.1	1.821	1.759	0.96	2470	484
s13207.1	0.975	0.922	1.06	1410	269

## 5. Conclusions

Power simulators offer an accurate solution to estimate power dissipation for CMOS circuits. However, for larger designs, it is common that the given set of functional vectors which characterizes the typical operations of a circuit is too long for such tools to handle. On the other hand, logic-level approaches, though efficient, may not be able to provide accurate estimation due to some inherent limitations. In this paper, we propose a method that combines the advantages of the logic-level and the transistor-level

approaches. Our approach generates a compact set of vectors that reflects a similar transition profile as the one produced by the given set of simulation vectors. We developed an iterative algorithm to generate this compact set of vectors. In each iteration, we compute the expected value of each signal based on a guidance called transition momentum. Then we perform the backward weight propagation to generate a high quality vector. Our experimental results show that this is a promising approach for using transistor-level power simulators for large designs to obtain accurate power estimation within reasonable time budget.

## References

- [1] M. A. Cirit, "Estimating Dynamic Power Consumption of CMOS Circuits," Proceedings of *ICCAD*, pp. 534-537, Nov. 1987.
- [2] S. M. Kang, "Accurate Simulation of Power Dissipation in VLSI Circuits," *IEEE Journal of Solid State Circuits*, vol. SC-21, no. 5, pp. 889-891, Oct. 1986.
- [3] A. C. Deng, Y. C. Shiau, and K. H. Loh, "Time Domain Current Waveform Simulation of CMOS Circuits," Proceedings of *ICCAD*, pp. 208-211, Nov. 1988.
- [4] G. Y. Yacoub, and W. H. Ku, "An Accurate Simulation Technique for Short-Circuit Power Dissipation Based on Current Component Isolation," *IEEE International Symposium on Circuits and Systems*, pp. 1157-1161, 1989.
- [5] F. Najm, "Transition Density: A Stochastically Measure of Activity in Digital Circuits," Proceedings of the *Custom Integrated Circuits Conference (CICC)*, pp. 19.7.1-10.7.6, May 1990.
- [6] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits," *IEEE/ACM Design Automation Conference*, pp. 253-259, June 1992.
- [7] A. Shen, A. Ghosh, S. Devadas, K. Keutzer, "On Average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks," Proceedings of *ICCAD*, pp. 403-407, Nov. 1992.
- [8] R. Burch, F. Najm, P. Yang, T. Trick, "McPower: A Monte Carlo Approach to Power Estimation," Proceedings of *ICCAD*, pp. 224-228, Nov. 1992.
- [9] C. Y. Tsui, M. Pedram, A. M. Despain, "Efficient Estimation of Dynamic Power Consumption under a Real Delay Model," Proceedings of *ICCAD*, pp. 90-97, Nov. 1992.
- [10] A. M. Hill and S. M. Kang, "Determining Accuracy Bounds for Simulation-Based Switching Activity Estimation," *Int'l Workshop on Lower-Power Design*, pp. 215-220, 1994.
- [11] J. Monterio and S. Devadas, "Techniques for Power Estimation of Sequential Logic Circuits under User-Specified Input Sequence and Programs," *Int'l Workshop on Lower-Power Design*, pp. 33-38, 1994.
- [12] D. I. Cheng, M. Marek-Sadowska, and K.-T. Cheng, "Speeding Up Power Estimation by Topological Analysis," Proceedings of *Custom Integrated Circuit Conference (CICC)*, pp. 623-626, May 1995.
- [13] D. I. Cheng, K.-T. Cheng, D. C. Wang, and M. Marek-Sadowska, "A new Hybrid Methodology for Power Estimation," to appear in Proceedings of *Design Automation Conference*, June 1996.
- [14] PowerMill Reference Manual, Version 2.7.1, *EPIC Design Technology*, Aug. 1992.