

High-Level Area and Power-up Current Estimation Considering Rich Cell Library*

Fei Li, Lei He, Joseph M. Basile[†], Rakesh J. Patel[†] and Hema Ramamurthy[†]

University of California, Los Angeles, CA 90095

[†]Intel Corporation, Santa Clara, CA 95054

Abstract— Reducing the ever-growing leakage power is critical to power efficient designs. Leakage reduction techniques such as power-gating using sleep transistor insertion introduces large power-up current that may affect circuit reliability as well as introduce performance loss. We present an in-depth study of high-level power-up current modeling and estimation in the context of a full custom design environment with a rich cell library. We propose a methodology to estimate the circuit area in terms of gate count and maximum power-up current for any given logic function. We build novel estimation metrics based on logic synthesis and gate level analysis using only a small number of typical circuits, but no further logic synthesis and gate level analysis are needed during our estimation. Compared to time-consuming logic synthesis and gate level analysis, the average errors for circuits from a leading industrial design project are 23.59% for area and 21.44% for maximum power-up current. In contrast, estimation based on quick synthesis leads to 11x area difference in gate count for an 8bit adder.

I. INTRODUCTION

As VLSI technology advances, leakage power becomes an ever growing power component. Dynamic power management via power gating at system and circuit levels is effective to reduce both leakage and dynamic power. Figure 1 (a) shows a system with a multi-channel voltage regulation module (VRM). The VRM channels can be configured to supply power *independently* for individual modules. Therefore, modules can be turned on or off at appropriate times for power reduction. Power gating at the circuit level is also called MTCMOS (see Figure 1 (b)). A PMOS sleep transistor with a high threshold voltage connects the power supply to the virtual Vdd. The sleep

transistor is turned on when the function block is needed, and is turned off otherwise.¹ We use MTCMOS to study power gating in this paper and the idea can be extended to VRM.

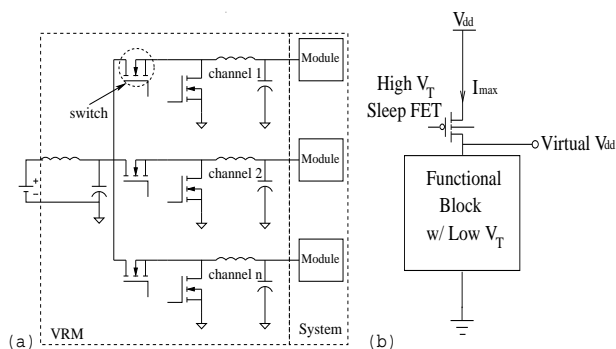


Fig. 1. Power gating at (a) system level and (b) circuit level.

Key questions in applying power gating include: (i) How to estimate the power-up current introduced by power gating? Power-up current [1] is large transient current that can greatly affect the circuit reliability. Estimation of power-up current can further decide the area overhead of power gating in order to satisfy the reliability constraints, and therefore determines whether power gating is worthwhile for a given design, and (ii) How to answer the above question at an early design stage without performing time-consuming logic synthesis and gate level analysis. Early decision making is crucial to deal with time-to-market pressure. Previous work [2, 3] address high-level estimation for capacitive power only. This paper presents an in-depth study of high-level power-up current modeling and estimation in the context of full custom design with a rich cell library and commercial synthesis tool such as Design Compiler.

In Section 2, we propose a method to estimate the gate count for a given logic function without performing logic

*This paper is partially supported by NSF CAREER award CCR-0306682, SRC grant HJ-1008, a UC MICRO grant sponsored by Analog Devices, Fujitsu Laboratories of America, Intel and LSI Logic, and a Faculty Partner Award by IBM. We used computers donated by Intel and SUN Microsystems. Address comments to lhe@ee.ucla.edu.

¹ Instead of the PMOS sleep transistor, an NMOS sleep transistor can be inserted between the ground and virtual ground and I_p to be presented later becomes the discharging current in this case. For simplicity of presentation, we assume PMOS sleep transistors in this paper.

synthesis. We show that the quick synthesis leads to 11x difference for a simple adder, and further validate and improve an area estimation technique that was originally developed for a library with limited number of cells [2]. The improved estimation method has an average error of 23.59%.

As shown in [1], all nodes in a power-gated module are at logic “0” state, and must be brought to valid logic states by power-up current (I_p) before useful computation can begin. Further, I_p depends on the input vector. Its maximum value must be known to design reliable sleep transistors and VRM. In Section 3, we propose a high-level metric to estimate the maximum I_p without performing logic synthesis and gate-level I_p analysis. We verify this metric by a newly developed gate-level analysis for accurate I_p . Section IV concludes this paper. In all sections, we use the design environment of a leading industrial high-performance CPU design project. There are hundreds of cells with various sizes (1x to 65x) in the library. All experiments are carried out on a number of typical circuits. The circuits are specified in Verilog and synthesized by Design Compiler to verify our high-level estimations. Due to the need of IP protection, we report normalized current value in this paper.

II. AREA ESTIMATION

A. Overview

TABLE I
AREA COUNT BASED ON QUICK SYNTHESIS

Circuit	Synthesis 1	Synthesis 2
1bit adder	3	3
4bit adder	20	16
8bit adder	42	490

Table A presents synthesis results for adders where *synthesis 1* uses logic functions with intermediate variables, and *synthesis 2* uses equivalent logic functions without intermediate variables. 11x difference in gate count is observed for an 8bit adder. It shows that quick synthesis using Verilog specified at a higher abstraction level does not necessarily lead to a good estimation. Instead of using quick synthesis, we apply and improve the high-level area estimation presented in [2].

We summarize the estimation flow from [2] in Figure 2. It contains a one-time pre-characterization, where gate-count \mathcal{A} is pre-characterized as a function \mathcal{F} of the linear measure \mathcal{L} and output entropy \mathcal{H} . Then, a *multi-output function* (MOF) is transformed into a *single output function* (SOF) by adding a *m-to-1* MUX, where m is the number of outputs in the original MOF. \mathcal{L} and \mathcal{H} are calculated for the SOF to look up the pre-characterized table and obtain gate count. Removing MUX from this gate count leads to \mathcal{A} for the original MOF.

We improve the original estimation method in two ways. First, it is claimed in [2] that SOFs with the same output entropy \mathcal{H} and same linear measure \mathcal{L} have the same \mathcal{A} . However, we find that it may not be true for VLSI functions implemented with a rich cell library. Functions with smaller output probability of logic ‘1’ have fewer gate count under the same linear measure. Therefore, we have pre-characterized \mathcal{A} as a function $\mathcal{F}(\mathcal{L}, \mathcal{P})$, where \mathcal{P} is the output probability. Since complementary probabilities lead to the same entropy, our pre-characterization is more detailed compared to that in [2]. Further, we have developed an output clustering algorithm to partition the original MOF into sub-functions (called sub-MOFs) with minimum support set overlap, and have improved the efficiency and accuracy of the high-level estimation. We summarize our estimation flow with the difference highlighted in Figure 2, and describe each step in the following sections.

<p><i>Estimation of gate count \mathcal{A} in [2]:</i></p> <p>Pre-characterization</p> <ol style="list-style-type: none"> 1. Pre-characterize \mathcal{A} as $\mathcal{F}(\mathcal{L}, \mathcal{H})$ using randomly generated SOFs. \mathcal{H} is the output entropy. <p>Mapping</p> <ol style="list-style-type: none"> 1. Partition the MOF into sub-MOFs randomly; 2. Transform each sub-MOF into an SOF by adding MUX; 3. Compute \mathcal{L} and \mathcal{H}; 4. Obtain gate count \mathcal{A}' of transformed SOF from $\mathcal{F}(\mathcal{L}, \mathcal{H})$; 5. Remove MUX from \mathcal{A}' to get gate count of original MOF; 6. Obtain final estimate by adding up gate-count for all the sub-MOFs.
<p><i>Estimation of gate count \mathcal{A} in this paper:</i></p> <p>Pre-characterization</p> <ol style="list-style-type: none"> 1. Pre-characterize \mathcal{A} as $\mathcal{F}(\mathcal{L}, \mathcal{P})$ using randomly generated SOFs. \mathcal{P} is the output probability. <p>Mapping</p> <ol style="list-style-type: none"> 1. Partition the MOF to minimize support-set overlap; 2. Transform each sub-MOF into an SOF by adding MUX; 3. Compute output probability \mathcal{P} and linear measure \mathcal{L}; 4. Obtain gate count \mathcal{A}' of transformed SOF from $\mathcal{F}(\mathcal{L}, \mathcal{P})$; 5. Remove MUX from \mathcal{A}' to get gate count of original MOF. 6. Obtain final estimate by adding up gate-count for all the sub-MOFs.

Fig. 2. Estimation of gate count \mathcal{A}

B. Linear Measure

Linear measure \mathcal{L} is determined by on and off-sets of an SOF as $\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_0$, where \mathcal{L}_1 and \mathcal{L}_0 are the linear measure for the on-set and off-set, respectively. \mathcal{L}_1 is further defined as $\mathcal{L}_1(f) = \sum_{i=1}^N c_i p_i$ (\mathcal{L}_0 can be defined similarly). N is the number of different sizes of all the prime implicants in a minimal cover of function f . The size of a prime implicant is the number of literals in it. c_i is one distinct prime implicant size. p_i is a weight of prime implicants with size c_i and can be computed in the following way. Suppose all the input vectors to the logic function can occur with the same probability. Let c_1, c_2, \dots, c_N be sorted in a decreasing order, and weight p_i be the probability that one random input vector matches all the prime implicants with size c_i but not by the prime implicants with size from c_1 to c_{i-1} , $1 < i \leq N$.

For $i = 1$, p_1 is just the probability that one random input vector matches prime implicants with size c_1 . Here “a matching” means that the intersection operation [4] between the vector and the prime implicant is consistent. Note that p_i satisfies the equation $\sum_{i=1}^N p_i = \mathcal{P}(f)$, where $\mathcal{P}(f)$ is the probability to satisfy function f .

The minimum cover of an SOF can be obtained by two-level logic minimization. To compute the weight p_i , a straightforward approach is to make the minimum cover disjoint and compute the probability exactly. However, in practice, this exact approach turns out to be very expensive. In our experiments, when the number of inputs is larger than 10, the program using the exact approach does not finish within reasonable time. But with p_i defined as the probability, $\mathcal{L}_1(f)$ can be viewed as a random variable $\mathcal{L}'_1(f)$ with certain probability distribution. For each random input vector, the variable $\mathcal{L}'_1(f)$ takes a certain value ‘randomly’. With probability of $1 - P(f)$, $\mathcal{L}_1(f)$ takes the value ‘0’. Then, $\mathcal{L}_1(f)$ becomes the *mean* of the random variable $\mathcal{L}'_1(f)$. By assuming that the variable $\mathcal{L}'_1(f)$ takes a Gaussian distribution, we use Monte Carlo simulation technique to estimate the mean value efficiently.

C. Output Probability and Gate-count Recovery

The output probability can be obtained as a by-product of Monte Carlo simulation. Since weight p_i satisfies $\sum_{i=1}^N p_i = P(f)$, we can keep record of all the p_i during the Monte Carlo simulation. When simulation process satisfies the stopping criteria, the output probability can be obtained easily. To recover the gate count of the original MOF, we use the same method in [2]. The number of gates to implement the added MUX in the MOF-to-SOF transformation are subtracted from the estimated gate count for the SOF, and the gate count for the original MOF is obtained.

D. Output Clustering

As the number of primary outputs increases, the time to calculate the minimum cover of a function increases non-linearly. To make the two-level optimization more efficient, one may partition the original MOF into sub-MOFs by output clustering, and then estimate for each sub-MOF individually. The gate-count of the original MOF is the sum of gate-counts for all the sub-MOFs. However, estimation errors may be introduced due to the overlap of the support sets of the sub-MOFs. We propose to partition the outputs with minimum support set overlap (see Figure 3). A PO-graph is constructed with vertices representing the Primary Outputs (POs). If two POs have support set overlap, there is an edge connecting the two corresponding vertices. The edge weight is the size of the common support set. The vertex weight is the sum of the weights of all edges connected to this vertex. There are two loops in the algorithm. In each iteration of the inner loop, the vertex with the minimum weight is deleted

and the weights are updated for edges and vertices that connect the deleted vertex. It continues until the number of remaining vertices is less or equal to the pre-specified cluster size. The PO-graph is then re-constructed with all the POs that have not been clustered. The algorithm continues until all the outputs are clustered and the PO-graph becomes empty.

```

Output Clustering Algorithm:
Construct the PO-graph;
While (PO-graph is not empty)
begin
  While (# of remaining vertices > CLUSTER_SIZE)
    begin
      Delete the vertex  $v$  with the minimum weight;
      Update weights for edges and vertices connecting  $v$ ;
    end
  Obtain one cluster of POs using remaining vertices;
  Re-construct PO-graph excluding POs already in clusters;
end

```

Fig. 3. Output clustering algorithm

E. Experimental Results

We compare area estimation methods in Figure 4, where x-axis is the circuit ID number and y-axis is the gate count. During the Monte Carlo simulation to calculate the linear measure, we choose the parameters of confidence and error as 96% and 3%, respectively. The actual gate count is obtained by the synthesis using Design Compiler. The method with random output clustering has an average absolute error of 39.36%. By applying our output clustering algorithm to minimize support set overlap, we reduce the average absolute error to 23.59%. Such estimation errors are much smaller compared to the 11x gate-count difference in Table A. Note that different descriptions of a given logic function do not change the \mathcal{L} and \mathcal{P} , and therefore do not affect the estimation results by our approach. The area estimation costs over 100x less runtime compared to logic synthesis.

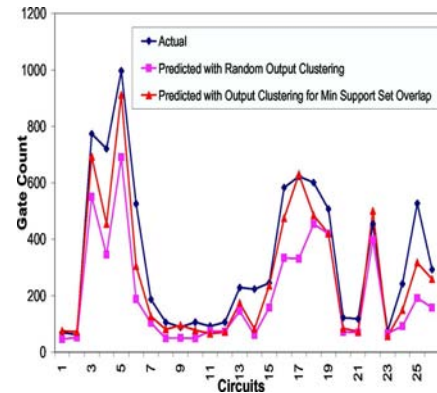


Fig. 4. Comparison between actual and predicted gate-count.

III. POWER-UP CURRENT ESTIMATION

Given the Boolean function f of a combinational logic block and the target cell library, our high-level estimation finds the maximum power-up current $I_p(f)$ when the logic block is implemented with the given cell library for power gating. A Boolean function can be implemented under different constraints, but we assume the min-area implementation in this paper.

We propose the following high-level metric \mathcal{M}_p for $I_p(f)$:

$$I_p(f) \propto \mathcal{M}_p(f) = \mathcal{I}_{avg} \cdot \mathcal{A} \quad (1)$$

where \mathcal{A} is the gate count estimated using the method in Section II, and \mathcal{I}_{avg} is the weighted average I_p to be discussed in Section B. Because an accurate gate-level estimator is required for the calculation of \mathcal{I}_{avg} and verification of $\mathcal{M}(f)$, we first introduce our gate-level estimation in section A.

A. Gate-Level Estimation

A.1 Background Knowledge

Power-up current (I_p) is different from the normal switching current (I_s). I_s depends on two successive circuit states S_1 and S_2 , which are determined by two successive input vectors V_1 and V_2 for combinational circuits. As discussed in Section I, I_p can be viewed as a special case of I_s where the state S_1 before power-up is logic “0” for all the nodes. Because no input vector leads to a circuit state with all nodes at logic “0” for non-trivial circuits, the maximum I_p is in general different from the maximum I_s . Moreover, the I_p of a circuit is *solely* decided by the circuit state S_2 , and therefore decided by a single input vector when the circuit is powered up. To illustrate that I_p depends on the input vectors, we present the I_p obtained by SPICE simulation for an 8-bit adder under two different input vectors in Table A.1. The difference of the maximum I_p is about 24%. It is obvious that I_p is greatly affected by the input vector when the circuit is powered up. We refer I_p element to be the power-up current generated by an individual gate, and give the following observation related to timing:

Observation 1 *If a set of gates are controlled by one single sleep transistor, all these gates are powered up simultaneously. I.e., all the I_p elements for these gates have the same starting time.*

TABLE II
MAXIMUM I_p OF AN 8BIT ADDER.

Circuit	vector1	vector2	difference
Adder8	1830 (uA)	2260 (uA)	23.50%

ATPG-based algorithms have been proposed in [1], where it is assumed that the power-up current is proportional to the total charge in the circuit after power-up, and the charge for one single gate with output value “1” is proportional to its fanout number. Therefore, the gate fanout number is used as the figure of merit of the power-up current (I_p) for the gate with output value “1”. ATPG algorithm is performed to find the logic vector that maximizes the figure of merit. However, this algorithm does not take the current waveform in the time domain into account. The vector obtained by ATPG algorithm has to be further used in SPICE simulation to obtain the I_p value.

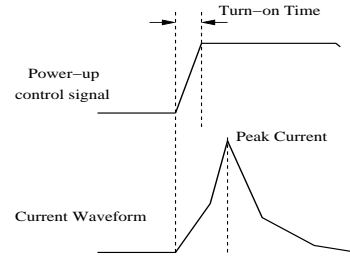


Fig. 5. PWL current model.

To achieve more accurate estimation and obtain I_p value directly, we need a current model that can capture the current waveform. We apply the piece-wise linear (PWL) function to model the I_p element (see Figure 5). We use SPICE simulation to build the PWL model for each cell in the library. Our PWL model considers the following four dimensions: gate type, input pin number, gate size, and post-powerup output logic value. Note that a much simplified PWL model, the right-triangle current model has been successfully used in [5] for maximum switching current estimation.

A.2 Genetic Algorithm

Since exhaustive search for the input vector that generates the maximum I_p is infeasible, we apply Genetic Algorithm (GA) in our gate-level estimation. We encode the solution, input vector, into a string so that the length of the string is equal to the number of primary inputs. Each bit in the string is either ‘1’ or ‘0’. The initial population is randomly generated. The population size is proportional to the number of primary inputs. The fitness value is chosen as the maximum I_p value under the input vector represented by the string. The maximum I_p value is obtained by the waveform simulation with PWL current model.

Tournament selection [6] is used in our selection process. From the current generation, we randomly pick two strings and select the one with the higher fitness value. After that, the two strings are removed from the current generation. We repeat this procedure until the cur-

rent generation becomes empty. By doing this, we divide the original strings into inferior and superior groups. We keep record of the strings in the superior group and put these two groups together to carry out tournament selection again. The two superior groups generated in the two tournaments are combined to go through crossover and mutation, and produce the new generation. The string with the highest fitness will be selected twice so that the best solution so far will stay in the next generation. Since strings with lower fitness have higher probability of being dropped, the average fitness tends to increase by each generation.

The crossover scheme we use is the one-point crossover algorithm. One bit position is randomly chosen for two parent-strings and they are crossed at that position to get the two child-strings. After crossover, we further use a simple mutation scheme that flip each bit in the string with equal probability. The new generation is produced after crossover and mutation, and is ready to go through a new iteration of natural selection. The algorithm stops after the number of generations exceeds a pre-defined number. We summarize the algorithm in Figure 6.

```

GenerationNumber = 1;
Randomly generate the initial generation;
While (GenerationNumber < MAX_GENERATION)
begin
    Evaluate the fitness value of each string;
    Apply tournament selection to get parent-strings;
    Apply crossover and mutation to get child-strings;
    Produce the new generation with the child-strings;
    GenerationNumber++;
end
return the peak  $I_p$  and its correspondent input vector

```

Fig. 6. Genetic algorithm for gate-level estimation of maximum power-up current I_p .

A.3 Experimental Results

In Table A.3, we compare Genetic Algorithm to simulation with random vectors (*random* in the table) under the same PWL current model. The number of vectors in random simulation is 10x more than the number of vectors used in GA for each circuit. Since we estimate the maximum I_p in the worst case, the larger estimated value under the same current model means that it is closer to the upper bound of the maximum I_p . Compared to random simulation, GA can achieve up to 27% estimation improvement to approach the upper bound as highlighted in the table. The average improvement for all the circuits is 6%. GA tries much fewer number of input vectors than random simulation and therefore it runs much faster.

B. Calculation of \mathcal{I}_{avg} for High-Level Estimation

\mathcal{I}_{avg} is not simply the average I_p element for all cells in a library. The frequency of cells used in logic synthesis should be taken into account. We assume that the logic

TABLE III
EXPERIMENTS FOR GATE-LEVEL ESTIMATION OF MAXIMUM POWER-UP CURRENT I_p .

circuit ID	Random Sim (uA)	Genetic Alg (uA)	Upper-bound Estimate Improvement(%)
1	12234.00	12234.00	0.00
2	11756.12	11756.12	0.00
3	147989.14	150437.28	+1.65
4	138497.98	143767.61	+3.80
5	193818.20	193818.20	+0.00
6	80758.16	92024.79	+13.95
7	26190.60	28170.35	+7.56
8	13124.87	13975.08	+6.48
9	15205.17	16349.61	+7.53
10	30649.70	30964.78	+1.03
11	16687.65	16687.65	+0.00
12	18042.84	18577.41	+2.96
13	42066.13	42565.20	+1.19
14	20261.80	25710.94	+26.89
15	42982.51	49502.92	+15.17
16	84997.98	94697.20	+11.41
17	105486.86	113573.73	+7.67
18	123394.24	124298.63	+0.73
19	100145.88	100798.74	+0.65
20	18250.63	18907.54	+3.60
21	20560.78	21486.25	+4.50
22	85656.30	85981.01	+0.38
23	11911.55	12599.41	+5.78
24	38598.95	41482.89	+7.47
25	81358.29	92199.21	+13.32
26	35530.71	40449.54	+13.84
Average Improvement			+6.06

synthesis results using the target library are available for a few typical circuits. This is a reasonable assumption for an industry design environment. We calculate \mathcal{I}_{avg} in a regression-based way as follows: We compute the average maximum I_p per gate for n typical circuits by applying the gate-level estimation. As the number of circuits n increases, the calculated \mathcal{I}_{avg} value becomes more stable. We use this stable value as the \mathcal{I}_{avg} in our high-level estimation. Figure 7 shows the \mathcal{I}_{avg} with respect to the number of circuits used to calculate \mathcal{I}_{avg} . The figure shows that the change of the \mathcal{I}_{avg} value is relatively large when the number of circuits is small (less than 10 in the figure). After the number of circuits increases to 20, the value of \mathcal{I}_{avg} becomes very stable and can be used as our high-level metric \mathcal{M}_p .

To validate our regression-based \mathcal{I}_{avg} , we use the computed value of \mathcal{I}_{avg} under PWL model and the accurate gate count to get the high-level metric \mathcal{M}_p . We compare the gate-level estimation $I_p(ckt)$ by Genetic Algorithm to the metric \mathcal{M}_p in Figure 8. The average absolute error between $I_p(ckt)$ and \mathcal{M}_p is 12.02%. Note that the circuits in Figure 8 are different from those used to compute \mathcal{I}_{avg} for the purpose of the verification of metric \mathcal{M}_p .

Furthermore, we compare the maximum I_p using estimated \mathcal{I}_{avg} and \mathcal{A} to the maximum I_p obtained via logic synthesis followed by gate-level analysis. As shown in Table B, the average estimation error is 21.44%. We measure gate-level analysis runtime as the time for logic synthesis and Genetic Algorithm, and measure the high-level estimation runtime as the time for area estimation and application of the formula $\mathcal{M}_p(f) = \mathcal{I}_{avg} \cdot \mathcal{A}$ (pre-

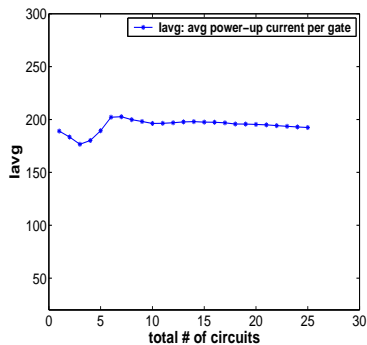


Fig. 7. $\mathcal{I}_{avg}(PWL)$ w.r.t. number of circuits.

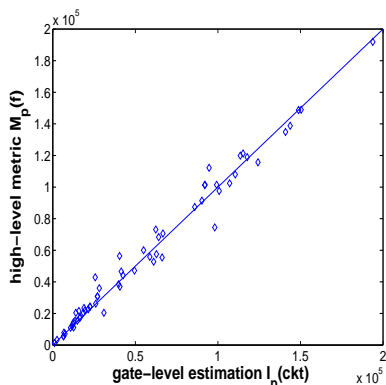


Fig. 8. Comparison between gate-level estimation $I_p(ckt)$ and high-level metric \mathcal{M}_p .

characterization only has one-time cost and is ignored in the runtime comparison). Our high-level estimation achieves more than 200x run-time speedup for large test circuits.

IV. CONCLUSIONS AND DISCUSSIONS

Using design examples and design environment of a leading industrial CPU project, we have presented an improved high-level area estimation method. The estimation has an average error of 23.59% for designs using a rich cell library. We have also proposed a high-level metric to estimate the maximum power-up current introduced power gating for leakage reduction. Compared to time-consuming logic synthesis followed by gate-level analysis, our high-level estimation has an average error of 21.44% for power-up current.

Our high-level estimation method can be readily applied to estimating the area overhead due to the sleep transistor insertion in power gating. There are two primary constraints for the sleep transistor. One constraint is the IR voltage drop that introduces performance penalty. Appropriate sizing of the sleep transistor can be performed to satisfy this constraint based on the maxi-

TABLE IV
RESULTS OF HIGH-LEVEL ESTIMATION FOR I_p .

circuit ID	PWL current model		Abs. Error(%)
	gate-level est. $I_p(ckt)$ (uA)	high-level est. $I_p(ckt)$ (uA)	
1	12234.00	14626.98	19.56
2	11756.12	13857.14	17.87
3	150437.28	133374.96	11.34
4	143767.61	87569.42	39.09
5	193818.20	175523.76	9.44
6	92024.79	58700.38	36.21
7	28170.35	24442.45	13.23
8	13975.08	15589.28	11.55
9	16349.61	18283.73	11.83
10	30964.78	15011.90	51.52
11	16687.65	12702.38	23.88
12	18577.41	13664.68	26.44
13	42565.20	33488.08	21.33
14	25710.94	15781.74	38.62
15	49502.92	45420.62	8.25
16	94697.20	91418.62	3.46
17	113573.73	121442.42	6.93
18	124298.63	93343.23	24.90
19	100798.74	80833.31	19.81
20	18907.54	16166.66	14.50
21	21486.25	14049.60	19.81
22	85981.01	96422.59	12.14
23	12599.41	10970.23	12.93
24	41482.89	28676.58	30.87
25	92199.21	61009.90	33.83
26	40449.54	49847.21	23.23
Average			21.44

imum switch current. The other constraint is the reliability of the sleep transistor, i.e., avoidance of damaging the sleep transistor by a large transient current. We can obtain the maximum transient current as the bigger one between the maximum power-up current and maximum switching current, and size the sleep transistor to satisfy the reliability constraint.

REFERENCES

- [1] F. Li and L. He, "Maximum current estimation considering power gating," in *Proc. Intl. Symp. on Physical Design*, pp. 106–111, 2001.
- [2] M. Nemani and N. Najm, "High-level area and power estimation for VLSI circuits," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 697–713, June 1999.
- [3] K. M. Buyuksahin and F. N. Najm, "High-level power estimation with interconnect effects," pp. 197–202, July 2000.
- [4] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. IEEE PRESS, 1990.
- [5] A. Krstic and K.-T. Cheng, "Vector generation for maximum instantaneous current through supply lines for CMOS circuits," in *Proc. Design Automation Conf.*, pp. 383–388, June 1997.
- [6] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection and effects of noise," in *Complex Systems*, pp. 193–212, June 1995.