High-Level Area and Current Estimation *

Fei Li, Lei He, Joe Basile†, Rakesh J. Patel† and Hema Ramamurthy†

EE Department, University of California, Los Angeles, CA
†Intel Corporation, San Jose, CA

Abstract. Reducing the ever-growing leakage current is critical to high performance and power efficient designs. We present an in-depth study of high-level leakage modeling and reduction in the context of a full custom design environment. We propose a methodology to estimate the circuit area, minimum and maximum leakage current, and maximum power-up current, introduced by leakage reduction using sleep transistor insertion, 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, 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.

1 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.

Key questions in applying power gating include: (i) How to estimate the leakage reduction by power gating and how to decide the area overhead of power

* This research was partially supported by the NSF CAREER Award 0093273, SRC grant 2002-HJ-1008 and a grant from Intel Design Science and Technology Committee. 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.

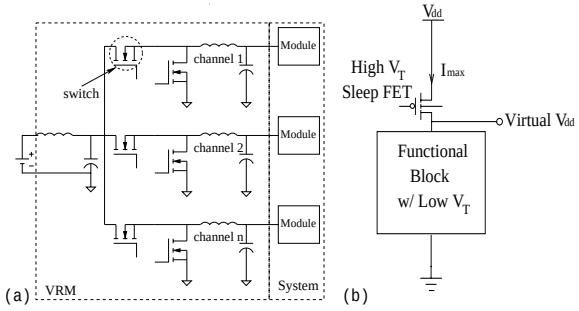

Fig. 1. Power gating at (a) system level and (b) circuit level.

gating? The answer 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 needed to deal with time-to-market pressure. This paper presents an in-depth study of high-level leakage modeling and reduction by using commercial synthesis tools such as Design Compiler.

In Section 2, we propose a method to estimate the gate count for a given logic function without performing logic 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 [1]. The improved estimation method has an average error of 23.59%. As shown in [2], 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. 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.

2 Area Estimation

2.1 Overview

Table 1 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

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

Table 1. Area count based on quick synthesis

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 in [1].

We summarize the estimation flow from [1] 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 [1] 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 [1]. 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 and our implementation details in the following sections.

2.2 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 $c_1, c_2, \ldots, c_N$ 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

Estimation of gate count $\mathcal{A}$ *in [1]*: Pre-characterization 1. Pre-characterize $\mathcal{A}$ as $\mathcal{F}(\mathcal{L}, \mathcal{H})$ using randomly generated SOFs. $\mathcal{H}$ is the output entropy. Mapping 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.
Estimation of gate count $\mathcal{A}$ *in this paper*: Pre-characterization 1. Pre-characterize $\mathcal{A}$ as $\mathcal{F}(\mathcal{L}, \mathcal{P})$ using randomly generated SOFs. $\mathcal{P}$ is the **output probability**. Mapping 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 $\boldsymbol{\mathcal{F}(\mathcal{L}, \mathcal{P})}$; 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 $\mathcal{A}$

that the intersection operation 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 [3]. 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.

2.3 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, the estimated gate count for the transformed SOF is subtracted by $\alpha \mathcal{A}_{mux}$. $\mathcal{A}_{mux}$ is the gate count of the complete multiplexer we have inserted, and α is the coefficient to get the *reduced* multiplexor gate count due to the logic optimization.

2.4 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

2.5 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 1. 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. High-level estimation costs over 100x less runtime compared to logic synthesis.

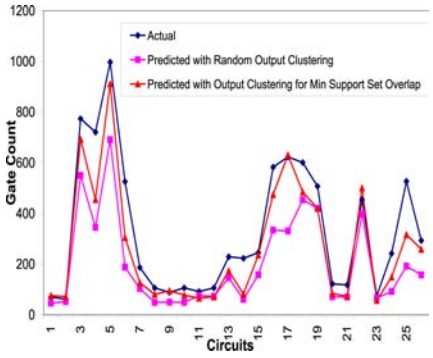

Fig. 4. Comparison between actual and predicted gate-count.

3 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 $I_p(f)$:

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

where $\mathcal{A}$ is the gate count estimated using the method in Section 2, and $\mathcal{I}_{avg}$ is the weighted average I_p to be discussed in Section 3.2. Because an accurate gate-level estimator is required for the calculation of $\mathcal{I}_{avg}$ and verification of $\mathcal{M}(f)$, we introduce our gate-level estimation in the next section.

3.1 Gate-Level Estimation

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 1, 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 2. 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.*

Circuit	vector1	vector2	difference
Adder8	1830	2260	23.50%

Table 2. Maximum I_p of an 8bit adder.

ATPG-based algorithms have been proposed in [2]. 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.

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. SPICE simulation is used to get the power-up current waveform and the waveform is linearized at different regions 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 [4] for maximum switching current estimation.

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 I_p value is obtained by waveform simulation with PWL current model.

Tournament selection 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 current 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 5.

```
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. 5. Generic algorithm for gate-level estimation.

We carry out experiments and compare the results of Genetic Algorithm to that of simulations with 5000 random vectors. Under the same PWL current model, GA achieves up to 27% estimation improvement to approach the upper bound of power-up current, The average improvement for all the circuits is 6%.

3.2 Calculation of $\mathcal{I}_{avg}$ and Experimental Results

$\mathcal{I}_{avg}$ 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 synthesis results for a few typical circuits (or random logic functions) are available. 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. We then increase n until the resulting value becomes a "constant".

We treat this constant value as $\mathcal{I}_{avg}$. In Figure 6 (a), we plot $\mathcal{I}_{avg}$ 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 6 (b). The average absolute error between $I_p(ckt)$ and $\mathcal{M}_p$ is 12.02%. Note that the circuits in Figure 6 (b) are different from those used to compute $\mathcal{I}_{avg}$ for the purpose of the verification of metric $\mathcal{M}_p$.

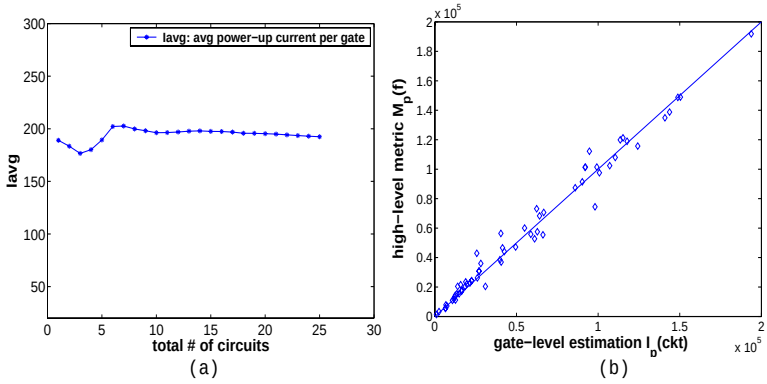

Fig. 6. (a) $\mathcal{I}_{avg}(PWL)$ w.r.t. number of circuits. (b) Comparison between gate-level estimation $I_p(ckt)$ and high-level metric $\mathcal{M}_p$.

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 3, shows that 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-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.

4 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 due to power gating for leakage reduction. Compared to time-consuming

PWL current model			
circuit id	gate-level est. $I_p(ckt)$	high-level est. $I_p(ckt)$	Abs. Err(%)
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

Table 3. Results of high-level I_p estimation.

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 is reliability constraint for sleep transistors, 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. 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.
2. F. Li and L. He, "Maximum current estimation considering power gating," in *Proc. Intl. Symp. on Physical Design*, pp. 106–111, 2001.
3. G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw Hill, 1994.
4. 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.