High-Level Area and Power Estimation for VLSI Circuits[†]

Mahadevamurty Nemani and Farid N. Najm

ECE Dept. and Coordinated Science Lab.
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, Illinois 61801
Tel: (217) 333-7678, Email: najm@uiuc.edu

Abstract

High-level power estimation, when given *only* a high-level design specification such as a functional or RTL description, requires high-level estimation of the circuit average activity and total capacitance. Considering that total capacitance is related to circuit area, this paper addresses the problem of computing the "area complexity" of multi-output combinational logic given only their functional description, i.e., Boolean equations, where area complexity refers to the number of gates required for an *optimal* multi-level implementation of the combinational logic. The proposed area model is based on transforming the multi-output Boolean function description into an equivalent single-output function. The area model is empirical and results demonstrating its feasibility and utility are presented. Also, a methodology for converting the gate count estimates, obtained from the area model, into capacitance estimates is presented. High-level power estimates based on the total capacitance estimates and average activity estimates are also presented.

1. Introduction

Rapid increase in the design complexity and the need to reduce time-to-market have resulted in a need for CAD tools that can help make important design decisions *early* in the design process. To do so, these tools must operate with a design description at a high level of abstraction. One design criterion that has received increased attention lately is power dissipation. This is due to the increasing demand for low power mobile and portable electronics. As a result, there is a need for high level power estimation and optimization (as well as modeling for area, timing, noise, etc.).

There are two types of modeling approaches: bottom-up and top-down. In the bottom-up approach, one starts with a complete implementation of a circuit block (down to gates, transistors, and/or layout) and builds a simple and compact higher-level model that gives the power of the block for any specified input vectors or input switching statistics. Bottom-up models can be built with high accuracy because the circuit level implementation is available. Examples of bottom-up techniques include the power factor approximation (PFA) technique [23, 24], the dual bit type (DBT) method [25, 26], the look-up table based techniques [27, 28], the clustering based method [30], and the cycle-accurate macro-model [31]. However, bottom-up models are not enough. Certain parts of the design (typically 25% or more) will consist of application-specific logic blocks that have not been previously designed. During high-level design planning, we need to have some figure-of-merit for the power that these new functions would require, once implemented in a given gate library. This paper proposes a technique to address this problem.

Specifically, we propose an area and power estimation capability, given only a *functional* view of the design, such as when a circuit is described only with Boolean equations. In this case, no structural information is known - the lower-level (gate-level or lower) description of this function is not available. Of course, a given Boolean function can be implemented in many ways, with varying power dissipation levels. We are interested in predicting the nominal area and power dissipation of a minimal area implementation of the function that meets a given delay specification.

[†] This work was supported by Rockwell, by Intel Corp., by the National Science Foundation (MIP-9623237 & 9710235), and by the Semiconductor Research Corporation (SRC 97-DJ-484).

For a combinational circuit, since the only available information is its Boolean function, we consider that its power dissipation will be modeled as follows:

$$P_{avg} \propto \mathcal{D}_{avg} \mathcal{A} \mathcal{C}_{avg} \tag{1}$$

where $\mathcal{D}_{avg}$ is an estimate of the average node switching activity that a gate-level implementation of this circuit would have, $\mathcal{A}$ (also referred to as *area complexity*) is an estimate of the gate count (assuming some target gate library), and $\mathcal{C}_{avg}$ is an estimate of the average node capacitance. The estimation of $\mathcal{D}_{avg}$ was covered in [1-3]. The problem of estimating $\mathcal{A}$ from a high-level description of the circuit corresponds to the problem of high-level area estimation. This problem is also of independent interest, as the information it provides can be very useful, for instance, during floorplanning.

In an early work [4], Shannon studied area complexity, measured in terms of the number of relay elements used in building a Boolean function (switch-count). In that paper, Shannon proved that the *asymptotic* complexity of Boolean functions is *exponential* in the number of inputs (n), and that for large n, *almost every* Boolean function is exponentially complex. In [18], Muller demonstrated the same result for Boolean functions implemented using logic gates (gate-count measure). A key result of his work is that a measure of complexity based on gate-count is independent of the nature of the library used for implementing the function.

Several researchers have also reported results on the relationship between area complexity and entropy ($\mathcal{H}$) of a Boolean function (entropy will be introduced in the next section). These include [19,5,20]. More recently, Cheng et. al. [6] empirically demonstrated the relation between entropy and area complexity, with area complexity measured as literal-count. They showed that *randomly generated* Boolean functions (for $n = 8, 9, 10$) have a complexity *exponential* in n, and proposed to use that model as a area predictor for logic circuits. However, the circuits tested were very small, typically having less than 10 inputs. As one tries to apply that model to realistic VLSI circuits, it quickly breaks down due to the exponential dependence on n, leading to unrealistically large predictions of circuit area. For example, when applied to a circuit with 32 inputs (having been tuned to $n = 25$ inputs), this model predicts an area of ≈ 400 million gates, whereas the circuit can in reality be implemented with only 84 gates!

In this paper, we use "gate-count" as a measure of complexity, mainly due to the key fact observed by Muller [18], and also because of the popularity of cell-based or library-based design. As mentioned above, it is clear that a given Boolean function can be implemented in many different ways, with different resulting areas and gate-counts. For instance, a circuit may contain redundant logic, which artificially increases its area and is not reflected in the circuit function. Since redundant logic is undesirable anyway, we aim to estimate the gate-count of an *optimized* implementation of a Boolean function. Specifically, in our experiments, we have compared our estimated gate-counts to the gate-count for optimal circuit implementations that were obtained using the SIS synthesis system.

Our estimation technique is based on the novel concept of *complexity measure* of a Boolean function, to be defined later in the paper. Based on this, we will provide an area prediction model which gives reasonable results for realistic circuits, which is a significant improvement over traditional techniques. This will be demonstrated with experimental results on a large set of benchmarks, for which we compare our predicted gate-counts to those obtained from SIS. We will then combine the area estimates, provided by the area estimation tool, with the high-level activity estimates [1, 3] to obtain high-level power estimates for various circuits. This paper is an extended version of [17].

The proposed technique has two important limitations that one should be aware of. Firstly, it is limited (in its present form) to combinational circuits. We continue to work on this problem and will, in future, extend this approach to sequential circuits. Secondly, the method does not apply to circuits containing large arrays of exclusive-or gates. Such circuits are also the source of problems in many CAD applications, such as in BDD construction for verification. The failure of the area model on these circuits could be due to the failure of the *complexity measure* to capture the extreme regularity of the on-set and off-set in the Boolean space of the function. This regularity leads to area implementations which are small, however the complexity measure would indicate otherwise. One can argue that is not an important limitation of the model, because large xor arrays are typically arithmetic units, and it is natural for arithmetic blocks to be modeled bottom-up, not top-down. As observed by one of the reviewers, our technique is "better suited to relatively un-structured control-logic, whereas techniques such as the DBT method [a bottom-up approach] are better suited to data-path blocks." In any case, one way around the problem of exclusive-or arrays is to require that the Boolean function specification explicitly list exclusive-or gates. In that case, these can be identified up-front and excluded from the analysis, so that the proposed method is applied only to the remaining circuitry. In the remainder of this paper we will not consider circuits composed of large exclusive-or arrays.

The proposed technique can be combined with high-level top-down delay estimation methods [22, 29] to derive the power-area-delay trade-off curves of a Boolean function, thus enabling the designer to make useful design trade-offs early in the design. Such a capability is essential to do early design planning for system-on-a-chip designs.

Before leaving this section, we should mention some previous work on layout area estimation from an RTL level view. Wu et. al. [7] proposed a layout area model for datapath and control for two commonly used layout architectures. They used transistor count as a measure of area of datapath and control logic. For datapath units, the average transistor count was obtained by averaging the number of transistors over different implementations of the unit. For control logic, they calculate the number of transistors from the sum of products (SOP) expression for the next state and control signals. In addition to this, the wiring area for both datapath and control logic were estimated. Kurdahi et. al. [8] modified the above model to account for effects of floorplanning (effects of cell placement and interconnect on chip area). In [8], the area model for control logic is also based on SOP expressions, similar to that of [7]. However, each product term is implemented with AND gates (in the library) and the sum with OR gates (in the library). Since the product terms could be much larger than the gates in the library, the resulting implementation is a multi-level one. The advantage of these models ([7], [8]) is that they account for the effect of interconnect and placement on the layout area. In both these methods ([7], [8]), the number of AND gates for the SOP expression is computed by counting the number of AND gates required for each product term and summing over all product terms. The number of OR gates required to implement the SOP expression is computed by counting the number of OR gates required to form the sum of the product terms. The area estimate is equal to the sum of the number of AND and OR gates required. In reality however, the optimal number of gates required to implement the function would be much smaller than the above sum, because it is frequently possible to apply logic optimization (synthesis) algorithms to give a much better implementation of the circuit.

This paper is organized as follows. In section 2 we give a background discussion on the high-level power model for Boolean functions [1, 3] and a brief discussion of the activity prediction model of [1, 3]. In section 3 we discuss the issues pertaining to the complexity of randomly generated Boolean functions. In section 4, we define the complexity measure, *linear measure*, which would be used to estimate the area complexity and also present a model to compute the complexity measure of multi-output Boolean functions. In section 5 we present an area prediction model and in section 6 we present the overall flow of our high-level area estimation algorithm followed by empirical results, demonstrating the feasibility and utility of the proposed area estimation scheme. In section 7 we propose a methodology for estimating $\mathcal{C}_{avg}$ and present empirical results demonstrating its utility. In section 8 we present results showing the utility of the proposed area model in estimating the area complexity of a Boolean function at any feasible delay point. In section 9 we combine the high-level capacitance estimates and high-level activity estimates to obtain high-level power estimates and compare these with gate-level power estimates obtained using a zero-delay and a general-delay timing model. This is done for the minimum-area and minimum-delay implementations. We end the paper with some conclusions presented in section 10.

2. Background

In this section we briefly discuss previously published results pertaining to high-level power and activity estimation [1, 3]. These results are being summarized here for the convenience of the reader.

2.1. High-Level Power Estimation Model

We restrict ourselves to the common static fully complementary CMOS technology. Consider a combinational logic circuit, composed of N logic gates, whose gate output nodes are denoted x_i, $i = 1, 2, \ldots, N$. If $D(x_i)$ is the transition density [9] of node x_i (average number of logic transitions per second), then the average power consumed by the circuit is:

$$P_{avg} = \frac{1}{2} V_{dd}^2 \sum_{i=1}^{N} C_i D(x_i)$$

where C_i is the total capacitance at node i. This expression accounts only for the capacitive charging/discharging component of power, and not for the so-called short-circuit power which is known to be only around 10% of the total power in well-designed circuits. The transition density is a measure of circuit switching *activity*. We will be using the terms "density" and "activity" interchangeably.

Since we wish to accomplish power estimation at a level of abstraction where the circuit internal details are not known, certain approximations seem inevitable. The model proposed for high-level power estimation [1, 3] is given by:

$$P_{avg} \propto \mathcal{C}_{tot} \times \mathcal{D}_{avg}$$

where $\mathcal{D}_{avg}$ is a measure of the average node switching activity and $\mathcal{C}_{tot}$ is total circuit capacitance. Also,

$$\mathcal{C}_{tot} = \mathcal{C}_{avg}\mathcal{A} \tag{2}$$

where $\mathcal{C}_{avg}$ is an estimate of the average gate capacitance for a given target library and $\mathcal{A}$ is an estimate of the area complexity of the Boolean function. It must be noted that all the quantities, $\mathcal{D}_{avg}$, $\mathcal{C}_{avg}$ and $\mathcal{A}$, have to be estimated from a high level description of the function. In this paper we adopt the above model for estimating the power.

The above power approximation ($P_{avg} \propto \mathcal{C}_{tot}\mathcal{D}_{avg}$) was tested on several benchmark circuits from the ISCAS-89 [15] and MCNC [16] benchmark suites. These circuits (described at the gate level) were simulated under realistic gate delay models, for randomly generated vector sequences, for input probabilities ranging from 0.1 to 0.9. Average circuit activity and the total average power were computed from this simulation [10], and the area was computed as the total circuit capacitance (sum of output capacitance of all the gates). The results of this test are shown in Fig. 1, demonstrating the validity of this power approximation. For further details on this model, please refer to [1,3].

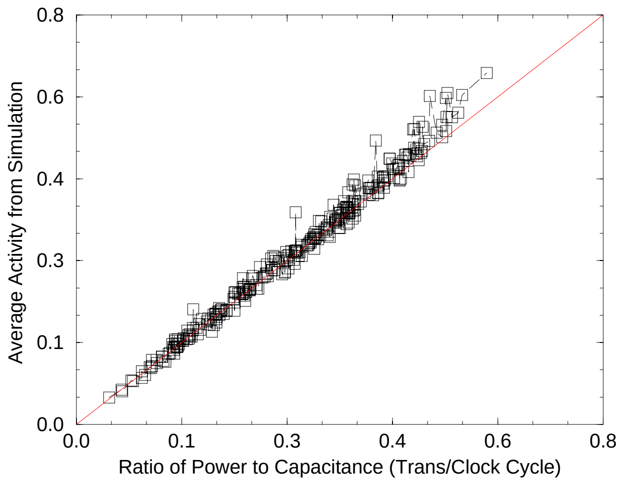

Figure 1. Accuracy of high-level power model.

2.2. High-Level Activity Prediction Model

It was observed in the previous sub-section that one would have to estimate the average switching activity of a combinational logic block in order to compute the power dissipated from a high level description. In this paper we use activity prediction model of [1, 3]. This model is based on the fact that the switching activity of a signal is related to its entropy. The model assumes that the primary inputs of the Boolean function are spatially and temporally independent, and is based on the empirical observation that the variation of cross-sectional entropy normalized to a linear width model falls quadratically with depth. From this observation it follows that:

$$\mathcal{D}_{avg} \approx \frac{2/3}{n+m}\left(D_i + 2D_o\right) \tag{3}$$

Here $\mathcal{D}_{avg}$ is the average activity of a node of the circuit, D_i is the sum of input activities, D_o is the sum of output activities, n is the number of primary inputs and m is the number of primary outputs of the Boolean function. For further information on the activity model, please refer to [1, 3].

3. Randomly Generated Boolean Functions

It was pointed out in [4] that for large n, Boolean functions have exponential complexity in n, based on a switch-count measure of complexity. In [6], Cheng et. al. point out a similar complexity behavior for randomly generated Boolean functions with n inputs, for $n = 8$, 9, and 10, using a literal-count measure (the same was observed by the authors when gate-count was used as a measure of complexity). By *randomly generated*, we mean that these functions were obtained by making a random choice for each point in the Boolean space, as to whether it belongs in the on-set or off-set of the function.

In [4], it was also pointed out that for sufficiently large n, all except a fraction δ of functions of n variables require at least $\frac{(1-\epsilon)2^n}{n}$ switch elements. The results of Cheng et. al. seem to indicate that this is also true for small n, so that "almost all" Boolean functions seem to have an exponential complexity. This suggests that the *average* area complexity of an n-input Boolean function (with the average taken over the set of *all* Boolean functions on n variables) varies exponentially with n. Perhaps based on the assumption that typical logic functions used in practice may be "average" (or close to average), the method in [6] applies this to every Boolean function, leading to the following area model:

$$\mathcal{A} \propto 2^n \mathcal{H}$$

Here n is the number of inputs, $\mathcal{A}$ is the area complexity measured as gate-count and $\mathcal{H}$ is the entropy of the output of the Boolean function (with independent inputs, each with probability 0.5) where entropy is the amount of information in a signal and can be easily computed from the signal probability using the following expression:

$$\mathcal{H} = p\log_2\left(\frac{1}{p}\right) + (1-p)\log_2\left(\frac{1}{1-p}\right)$$

The proportionality constant in the area expression depends on the library being used.

Risking abuse of terminology, we will refer to a Boolean function for which the above model holds true as an *average function*. Unfortunately, we have found that logic functions that are typically used in VLSI are far from being *average*, so that the above model breaks down very quickly for reasonable values of n. This is dramatically illustrated by the 32-input 84-gate circuit mentioned in the introduction, for which this model predicts an area close to 400 million gates. This behavior is typical of what we have seen.

Why is it that typical circuits are far from being average in terms of area complexity? We have investigated this by examining the structure of the on-sets for randomly generated functions, and found that their on-sets consist of points that are randomly scattered in the Boolean space, with no preferred direction. However, we have found that typical VLSI circuits have well structured distributions of their on-sets in the Boolean space, so that a function has certain preferred directions in which many of its cubes lie. This seems to translate to tremendous reduction in the area complexity relative to the (unstructured) randomly generated functions.

Thus typical VLSI circuits belong to the small minority of circuits whose area does not satisfy the model of Cheng et. al [6]. Finding an area model for such functions has remained an open problem. This paper utilizes the structure of the Boolean space, in addition to the entropy, to predict the area complexity.

4. The Area Complexity Measure

The problem of estimating the area complexity of a Boolean function pertains to estimation of the *minimum* number of gates ($\mathcal{A}$) required to implement the function, given only its high level description (Boolean equations) and a target technology library. It must be noted here that by implementation we mean an optimal multi-level implementation of the Boolean function. Let us, for now assume that the function at hand is a single-output function. For such functions it has been observed that the sizes of prime implicants of the on and off-sets may give us a hint about the area complexity. However to capture this dependence in a quantitative fashion, the notion of complexity measure will be introduced, which depends on the distribution of sizes of the prime implicants in the on and off-sets. This complexity measure will be referred to as the *linear measure* (one other complexity measure was introduced by the authors in [12], however it will not be discussed here).

The *linear measure* of a function is dependent on the complexity of the on and off-sets of the function. The linear complexity measure of the on-set (complexity of off-set can be defined similarly) is given by:

$$\mathcal{L}_1(f) = \sum_{i=1}^{N} c_i p_i \tag{4}$$

Here, $\mathcal{L}_1(f)$ is the linear complexity measure of the on-set of f, N is the number of distinct sizes (size of a cube is the number of literals in it) of prime-implicants in a minimal cover [13] of f, $\{c_1, c_2, \ldots, c_N\}$ are the distinct sizes of these prime implicants, and p_i is a weight on the prime implicants of size c_i such that $\sum_{i=1}^{N} p_i = \mathcal{P}(f)$ where $\mathcal{P}(f)$ is the probability that $f = 1$ when each point of the Boolean space has the same probability value and the probability of the entire space is 1. The weights p_i constitute a weighting function, defined as follows. For $\mathcal{L}_1(f)$, let the c_i be ordered such that $c_1 > c_2 > \cdots > c_N$. Let f_i refer to a Boolean sub-function of the original function f, defined so that its on-set consists only of the prime implicants of sizes $c_1, c_2, \ldots, c_i$, where $1 \leq i \leq N$. We define the weight p_i as follows:

$$p_i = \begin{cases} \mathcal{P}(f_i) - \mathcal{P}(f_{i-1}), & \text{if } i > 1; \\ \mathcal{P}(f_1), & \text{if } i = 1. \end{cases} \tag{5}$$

where $\mathcal{P}()$ denotes probability. Thus, p_i is the probability of the set of all min-terms in the on-set of f that are covered by the prime implicants of size c_i, but not by prime implicants of any larger size. With p_i thus defined, as probabilities, the expression (4) becomes equal to the *mean* of $\mathcal{L}_1(f)$ (when $\mathcal{L}_1(f)$ is assumed to take the value 0 with probability $1 - \mathcal{P}(f)$), $\mathcal{L}_1(f) = E[\mathcal{L}_1(f)]$, and hence can easily be computed using Monte-Carlo mean estimation techniques such as the one used in [10]. Using a similar development, $\mathcal{L}_0(f)$ can also be computed using Monte-Carlo simulation. Using $\mathcal{L}_1(f)$ and $\mathcal{L}_0(f)$ we can define the *linear measure* of f as:

$$\mathcal{L}(f) = \mathcal{L}_1(f) + \mathcal{L}_0(f) \tag{6}$$

We have observed that the presence of cubes of size one (cubes consisting of a single literal) can adversely affect the accuracy of the area estimation. This is because these cubes have a negligible effect on the gate count (a single OR gate) but have a big effect on the output probability value. Their presence also skews the probability distributions and makes the Monte Carlo estimation much more expensive. We have found that the best practical method for accounting for these cubes is to in effect exclude them from the summation (4) used to compute $\mathcal{L}_1(f)$, and similarly for $\mathcal{L}_0(f)$. This leads to improved estimation speed and much improved accuracy. Thus, the results to be presented in this paper make this modification to the cube distribution before carrying out the area prediction.

4.1. Complexity Measure for Multi-Output Functions

The complexity measure proposed in the previous section is based on the notion of complexity of the on and off-sets of a Boolean function. However, no such notion exists for multi-output Boolean functions. Moreover any notion of area complexity of a multi-output function should implicitly account for the fact that there is sharing of logic between the outputs of the Boolean function. In this section we propose a method by which the previously defined complexity measure can be extended to measure the complexity of multi-output functions. Our approach is inspired by the multi-valued logic approach to address the problem of two-level minimization of multi-output Boolean functions [13]. The approach is based on transforming a binary-valued, multi-output Boolean function into an equivalent multi-valued-input single-output (binary-valued) Boolean function. The transformation is accomplished by adding an m-valued input to the Boolean function, i.e., given:

$$f : \{0,1\}^n \rightarrow \{0,1\}^m$$

where f is an n-input, m-output Boolean function, f can be transformed to:

$$g : \{0,1\}^n \times \{0,1,\ldots,m-1\} \rightarrow \{0,1\}$$

where g is a binary-valued, single-output function with n binary inputs and one m-valued input. It must be noted here that each value of the multi-valued input corresponds to one of the m outputs. It has been shown that, for two-level minimization, minimizing a binary-valued, n-input, m-output Boolean function is equivalent to minimizing the corresponding multi-valued-input, singled output function. In our approach we perform a similar transformation on f, except that we use $\lceil \log_2(m) \rceil$ binary-valued inputs to implement a m-valued input. An equivalent way of representing the transformation is to think of the additional $\lceil \log_2(m) \rceil$ binary-valued inputs as control signals of a multiplexor, and that the value of the control word corresponds to the output being selected. This corresponds to multiplexing the m outputs of a m-output Boolean function, as shown if Fig. 2.

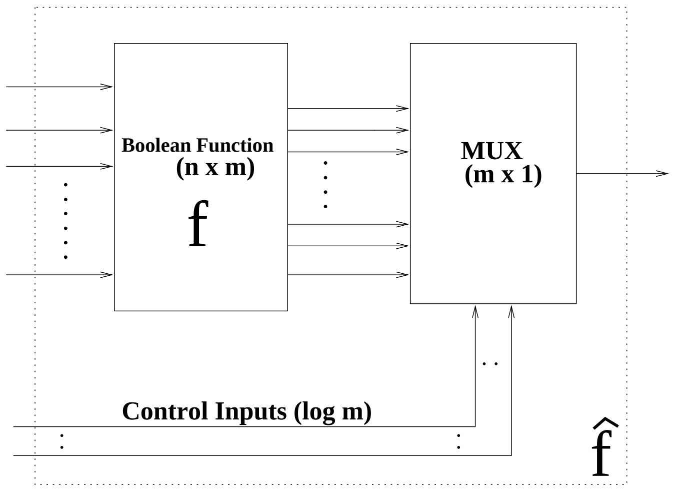

Figure 2. Transformation of an m-output Boolean Function into a single output Boolean function.

This multiplexing of the outputs of a n input, m output Boolean function f, gives rise to a $(n + \lceil \log_2 m \rceil)$ input, single-output Boolean function $\hat{f}$, shown in Fig. 2. Since $\hat{f}$ is a single-output function, its complexity measure can be computed, as presented in the previous section. It must be noted that by estimating the complexity of $\hat{f}$, which is made up of all the outputs, we are in effect dealing with all the outputs at the same time and thus automatically accounting for the effect of sharing. However, we must remember that complexity of $\hat{f}$ was computed by adding a multiplexor to f. Thus in order to compute the area complexity of f from the area complexity of $\hat{f}$, we must be able to compute the influence of the multiplexor on the area complexity of $\hat{f}$. This problem of recovery of area of f from that of $\hat{f}$ will be discussed in section 5.1.

5. The Area Complexity Model

In this section we present the area model to compute the area complexity $\mathcal{A}(f)$ of Boolean functions. The area model uses the complexity measure of the Boolean function along with its output entropy to estimate the

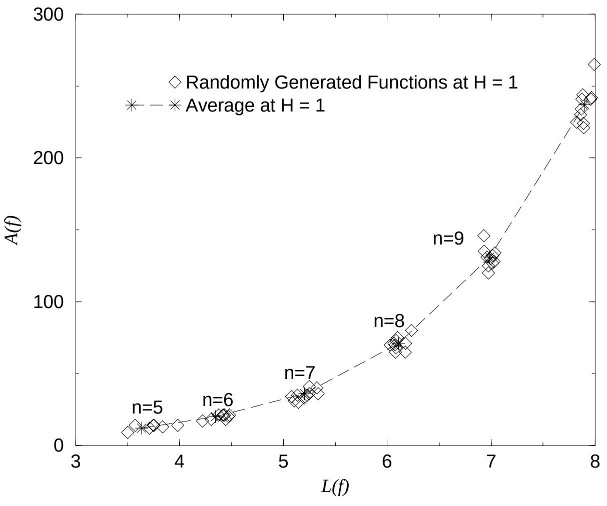

Figure 3. $\mathcal{A}(f)$ versus $\mathcal{L}(f)$ for Randomly Generated Boolean functions with $\mathcal{H}(f) = 1$.

area complexity. Since the approach adopted to estimate the area complexity of multi-output functions is to transform them into equivalent single-output functions, we will start by considering single-output functions, and then discuss the area recovery for the general case of multi-output functions.

5.1. Area Estimation for Single Output Functions

To start with, we will discuss the data shown in Fig. 3, which was generated as follows. For a given n, consider the set of all Boolean functions on n inputs *and* whose output entropy is $\mathcal{H}(f) = 1$, based on all inputs being independent and with 0.5 probability. For a number of randomly generated Boolean functions from this set, we computed their linear measure, $\mathcal{L}(f)$, using our algorithm and obtained an estimate of the gate-count $\mathcal{A}(f)$ from an optimized implementation of the function using SIS. We then plotted these points and it can be seen from the figure that the set of randomly generated functions for each n is clustered around specific points in the plot. This means that the distribution of $\mathcal{A}(f)$ of randomly generated Boolean functions (given n and $\mathcal{H}$) is *tight*, as observed by many others (see section 3). It also illustrates that the distribution of $\mathcal{L}(f)$ is also tight. The dotted curve shown in the figure is one which joins the center (average values) of each cluster and is close, but not exactly equal, to an exponential.

This almost-exponential $\mathcal{A}$ versus $\mathcal{L}$ curve is very important and is in fact the essence of our area prediction model. This is because we have found that not only do randomly generated Boolean functions fall on this curve, but also typical VLSI functions fall on it or close to it, as shown in Fig. 4. The data points shown in Fig. 4 correspond to test cases obtained from the benchmark suite presented in Table 1. It is noteworthy that the points are not clustered at specific points, but spread all over the curve. This illustrates the point made earlier about typical VLSI functions not being *average*. Further results will be given in the empirical results section, where we will use this curve to predict $\mathcal{A}(f)$, having first computed $\mathcal{L}(f)$. In fact, we use a family of such curves, corresponding to different entropy values, as shown in Fig. 5. Additional curves can be easily generated for other entropy values. These curves need to be generated only once, which is an up-front once-only cost, and they can then be used to predict the area of various functions.

An important consideration is what the largest n should be for which these curves need to be generated. Obviously, the curves are going to be more difficult to generate for larger n because of the cost of running synthesis to obtain the $\mathcal{A}(f)$ values. Luckily, there are two reasons why this is not a problem so that considering $n \leq 12$ as in Fig. 5 is sufficient. Firstly, we have found that for typical VLSI functions, the value of $\mathcal{L}(f)$ turns out to be much smaller than n in most cases. Indeed, all the test cases that we will present (for which n ranges from 4 to 70) had $\mathcal{L}(f) \leq 8$, so that the curves in Fig. 5 were sufficient. This fact is key

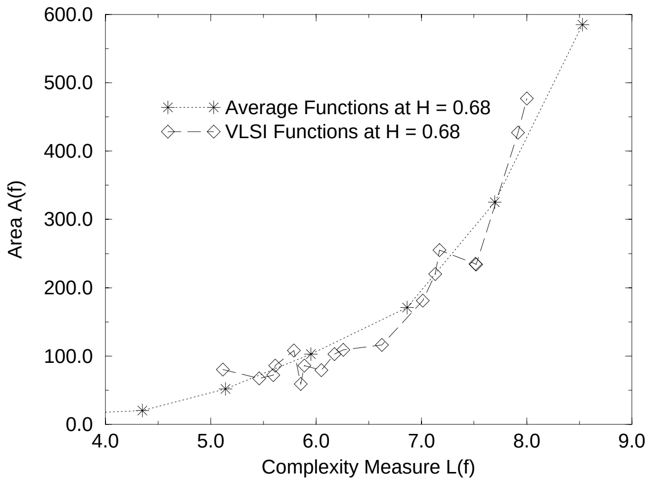

Figure 4. Typical VLSI functions fall close to the $\mathcal{A}$ vs. $\mathcal{L}$ curve

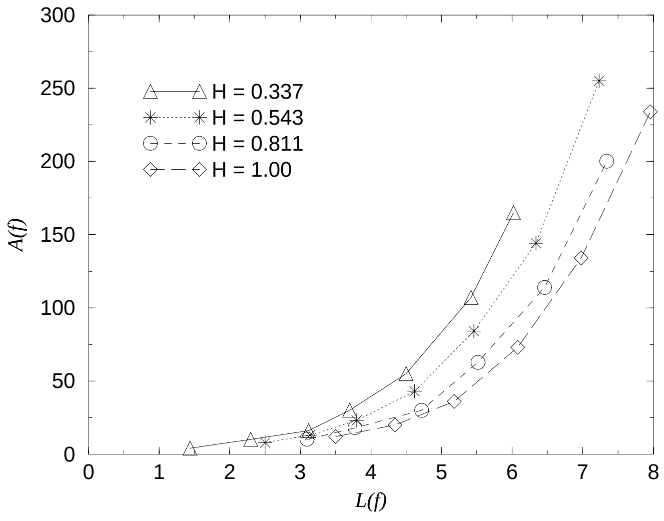

Figure 5. $\mathcal{A}(\cdot)$ versus $\mathcal{L}(\cdot)$ for different values of entropy

because it illustrates why the traditional (exponential in n) model breaks down while our (almost-exponential in $\mathcal{L}$) model gives reasonable results for typical VLSI functions.

The second reason why generating the curves only for small n is sufficient is that for larger values of n the curves become closer to the exponential and can be modeled analytically, as can be seen in the logarithmic plot in Fig. 6. For large n values, one can simply compute the area complexity as:

$$\mathcal{A}(f) = 2^{\mathcal{L}(f)}k(\mathcal{H})$$

where $k(\mathcal{H})$ is a proportionality constant that depends on the entropy $\mathcal{H}$, and can be computed using a least squares approach.

5.2. Area Recovery for Multi-Output Functions

We have seen previously that in order to compute the complexity measure associated with a multi-output function, we transform it into an equivalent single-output function by appending to it a multiplexor. However,

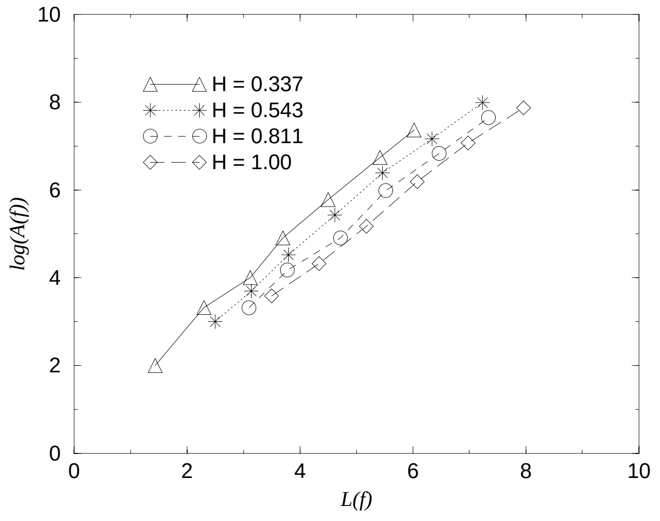

Figure 6. $\log_2 \mathcal{A}(\cdot)$ versus $\mathcal{L}(\cdot)$ for different values of entropy.

this transformation poses the problem of recovering the area of f from that of $\hat{f}$. In this section we address this problem.

A natural question to ask is, what is the relation between the (optimal) area of f and that of $\hat{f}$. To answer this question, consider the following two scenarios. In the first scenario, let all the outputs of the multi-output Boolean function be the same. In this case the area of the multi-output Boolean function is equal to the area of any of its outputs. Also note that the prime implicants of the on and off-sets of $\hat{f}$ are independent of the control inputs. Hence the complexity measure of $\hat{f}$ is equal to the complexity measure of any of the outputs of f. Also, as all the outputs of the function are the same, there is no need for the multiplexor. Thus the area contribution of the multiplexor to the overall area of a minimized $\hat{f}$ is zero.

Now consider the second scenario. Here assume that all the outputs of the multi-output Boolean function have disjoint support sets. It then follows that the optimal area of $\hat{f}$ is equal to the sum of optimal area complexity of f and the area complexity of the multiplexor. Thus one has to subtract the area of the multiplexor from the area complexity of $\hat{f}$ in order to get the area complexity of f. Moreover every prime implicant in the on and off-sets of $\hat{f}$ contains all the control inputs.

In the first scenario, when the contribution of the multiplexor to the area of $\hat{f}$ was zero, we saw that the control inputs were absent from all the prime implicants, while in the second scenario when the contribution of the multiplexor to the area of $\hat{f}$ is maximum, we saw that all the control inputs are present in every prime implicant of $\hat{f}$. Thus there seems to be a correlation between the influence of the multiplexor on the area of $\hat{f}$ and the number of control inputs in the prime implicants of $\hat{f}$.

From the above considerations, we propose that an appropriate area model for a multi-output function f, in terms of the area of $\hat{f}$ and the area of a m to 1 multiplexor is given by

$$\mathcal{A}(f) = \mathcal{A}(\hat{f}) - \alpha \mathcal{A}_{mux} \tag{7}$$

where $\mathcal{A}_{mux}$ is the area complexity of an m to 1 multiplexor, and $0 \leq \alpha \leq 1$ is a coefficient that represents the contribution of the multiplexor to the area complexity of $\hat{f}$. In the following, we present an approach for estimating α.

Note that the complexity measure of a m to 1 multiplexor is given by $\lceil \log_2 m \rceil + 1$, i.e., the complexity of a m to 1 multiplexor is proportional to the number of control inputs. This is true because every prime implicant of a m to 1 multiplexor has a size given by $\lceil \log_2 m \rceil + 1$. It is well known that an m to 1 multiplexor has a balanced tree decomposition such that the height of the tree is equal to $\lceil \log_2 m \rceil$ and the number of nodes in the tree is equal to $2^{\lceil \log_2 m \rceil}$. From this observation it follows that the area complexity ($\mathcal{A}_{mux}$) of a m to 1 multiplexor is given by:

$$\mathcal{A}_{mux} \propto 2^{\lceil \log_2 m \rceil} \tag{8}$$

Also we can re-write equation (7) as,

$$\alpha = \frac{\mathcal{A}(\hat{f}) - \mathcal{A}(f)}{\mathcal{A}_{mux}}$$

Here $\mathcal{A}(\hat{f}) - \mathcal{A}(f)$ represents the area contribution of the multiplexor to an optimal area implementation of $\hat{f}$. Note that after optimization it might so happen that certain control inputs become redundant for certain outputs. This manifests itself as some control inputs being absent in some prime implicants of on and off-sets of $\hat{f}$. Thus, we may think of $\mathcal{A}(\hat{f}) - \mathcal{A}(f)$ as representing the area of a *reduced* multiplexor resulting from the optimization. This reduced multiplexor area is related to the number of remaining control signals, which leads us to a method for estimating this area, as follows.

Let C_i denote the number of control inputs in a prime implicant P_i. Then define C_{on} to be the average number of control inputs in a prime implicant belonging to the on-set of $\hat{f}$, so that:

$$C_{on} = \frac{\sum_{i=1}^{K_{on}} C_i}{K_{on}} \tag{9}$$

where K_{on} is the number of prime implicants in the on-set of $\hat{f}$. Similarly, one can define C_{off}. From the above discussion it follows that C_{on} and C_{off} can be used to measure the area contribution of the multiplexor to an optimal area implementation of $\hat{f}$. Notice that the optimal implementation of $\hat{f}$ will contain a (implicit) reduced multiplexor whose area depends on the smaller of C_{on} and C_{off}. Thus, we can model this area contribution, in a fashion analogous to equation (8), as:

$$\mathcal{A}(\hat{f}) - \mathcal{A}(f) \propto 2^{\min\{C_{on}, C_{off}\}} \tag{10}$$

It then follows from equations (8) and (10) that:

$$\alpha = 2^{\min\{C_{on}, C_{off}\} - \lceil \log_2 m \rceil} \tag{11}$$

It must be noted that α can be computed with minimal effort from the prime implicants of $\hat{f}$, and once α is available, $\mathcal{A}(f)$ can be computed using (7).

6. High-Level Area Estimation Flow

The transformation, as stated in the previous section, does not place any restriction on the number of outputs that can be dealt with at a time (m). However, we have observed that in practice there is a trade-off between run time of the area estimation procedure and m. As the value of m increases we observed that the time taken to generate the prime implicants increases. Also, the size of the table capturing the variation of area of single-output Boolean functions with the *linear measure* increases. However, using too small a value of m can affect accuracy by overestimating the area, as the sharing between all outputs is not captured. Keeping these reasons in mind, after experimenting with different values of m, it was found that a reasonable choice for the value of m was 16.

Typically, a multi-output Boolean function has outputs with varying support set sizes. Outputs whose support set size is very small, for instance 1, 2 or 3, consume very little area. For these outputs very little area optimization can be done. One can make a reliable area prediction for such outputs without having to resort to the aforementioned approach. In fact it was found that an area estimate of two gates for outputs whose support set size is two, and an estimate of three gates for outputs with support set size of three, works very well in practice. As far as outputs with support set size of one are concerned, their contribution to an optimal area implementation depends on whether or not they are realized by inversion of a primary input signal. Those which are realized by inversion are assumed to contribute an area of one gate while the rest are assumed not to contribute to the area. The above approach yields benefits in terms of both run time

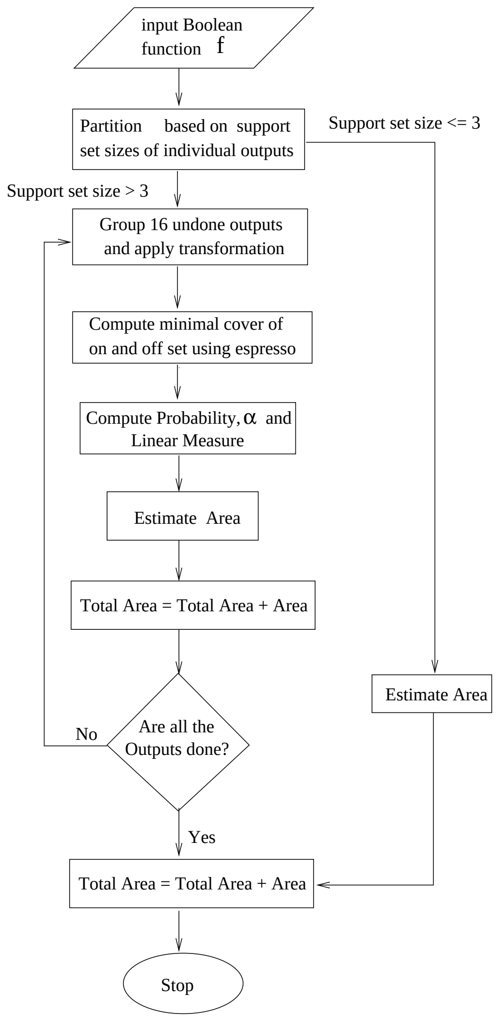
input Boolean function f
Partition based on support set sizes of individual outputs
Support set size <= 3
Support set size > 3
Group 16 undone outputs and apply transformation
Compute minimal cover of on and off set using espresso
Compute Probability, α and Linear Measure
Estimate Area
Total Area = Total Area + Area
Estimate Area
No
Are all the Outputs done?
Yes
Total Area = Total Area + Area
Stop

Figure 7. Flow Diagram of the overall Area estimation Procedure.

and accuracy, and has been adopted in our area estimation procedure. The flow diagram for the overall area estimation procedure is given in Fig. 7.

The area estimation tool reads an input description of f and partitions the function into two sub-functions. One sub-function (f_1) comprises of all outputs whose support set size is less than or equal to three, while the other (f_2), comprises of all outputs whose support set size is greater than three. The partitioning of the network into f_1 and f_2 can be performed by a breadth first search and is fairly inexpensive. We estimate the area of f_1 in the following fashion:

$$\mathcal{A}(f_1) = \beta|f_1^1| + 2|f_1^2| + 3|f_1^3| \tag{12}$$

Here, $|f_1^1|$ is the number of outputs in f_1 with support set size equal to 1, β is a fraction of these outputs which are realized by inversion of a primary input signal, $|f_1^2|$ is the number of outputs in f_1 with support set size equal to 2, and $|f_1^3|$ is the number of outputs in f_1 with support set size equal to 3. For estimating the area of f_2 we use the transformation based approach described above. Let the outputs of f_2 be grouped into I groups of size sixteen each, except perhaps for one group which may have fewer than 16 outputs. Let the Boolean function comprising of the i^{th} group of outputs be g_i. We apply the multiplexor transformation to g_i, and compute α, probability and the *linear measure* of the resultant $\hat{g}_i$. We then compute the area complexity of g_i using (7) and (11). This procedure is repeated until all the outputs have been used up, and the area of f_2 is estimated as:

$$\mathcal{A}(f_2) = \sum_{i=1}^{I} \mathcal{A}(g_i) \tag{13}$$

Finally, the area of f is computed as:

$$\mathcal{A}(f) = \mathcal{A}(f_1) + \mathcal{A}(f_2) \tag{14}$$

It must be noted that the proposed area model does not account for area sharing across groups. Also it must be mentioned that in our implementation no particular effort was made in grouping the outputs, as any specific style of grouping would require additional computational effort.

6.1 Empirical Results

The above proposed area model for multi-output functions was tested on several ISCAS-89 [15] and MCNC [16] benchmark circuits. These circuits are listed in Table 1 which, in addition to primary input and output count, shows the functionality of these benchmarks.

Table 1. Benchmark circuits and execution times.

CIRCUIT Name	Circuit Function	Inputs	Outputs	CPU Time sec
b9	Logic	41	21	5.7
c8	Logic	28	18	4.9
example2	Logic	85	66	28
frg2	Logic	143	139	268
i7	Logic	199	67	23.1
i8	Logic	133	81	81.5
i6	Logic	138	67	17.5
cht	Logic	47	36	6.5
alu2	ALU	10	6	12.8
alu4	ALU	14	8	104
term1	Logic	34	10	17.4
ttt2	Logic	24	21	6.25
apex6	Logic	135	99	45.3
apex7	Logic	49	37	20.3
x1	Logic	51	35	12.8
x3	Logic	135	99	53
x4	Logic	94	71	28.6
vda	Logic	17	39	39.3
k2	Logic	45	45	170.1
s298	Controller	17	20	4.4
s386	Controller	13	13	4.2
s400	Controller	24	27	8.5
s444	Controller	24	27	8.5
s510	Controller	25	13	6.9
s526	Controller	24	27	10.4
s526n	Controller	24	27	10
s641	Controller	59	43	41.4
s713	Controller	58	42	42.3
s820	Controller	37	24	16.3
s832	Controller	37	24	16.5
s953	Controller	39	52	38.8
s1196	Logic	28	32	163
s1238	Logic	28	32	141
s1494	Controller	27	25	26.8
s1488	Controller	27	25	29.3
s13207	Logic	152	783	212.8
s35932	Logic	1763	1728	942.4

These circuits were optimized in SIS using the script *script.rugged* for optimization, and mapped using the library *lib2.genlib*. The area predicted using the area model was compared with the SIS optimal area. The performance of the model on all the benchmarks in Table 1, except $s13207^*$ and $s35932$, is shown in Fig. 8. The circuit $s13207^*$ is a modified version of $s13207$, obtained by deleting the primary outputs which contain exclusive-or arrays in them. The SIS-optimal area of $s13207^*$ was 1367. The estimated area for this circuit was 1045. The circuit $s35932$ could not be optimized in SIS in one piece. Hence the circuit was partitioned based on the support set sizes (in a fashion similar to the above discussion) and optimized separately in SIS. The resulting SIS-area that was obtained was 7252. The area estimated by the area estimation tool was 8492. The area estimation results obtained have also been tabulated in Table 2. As indicated in Table 2, the average absolute error of our estimation approach on the benchmarks is 21.65%.

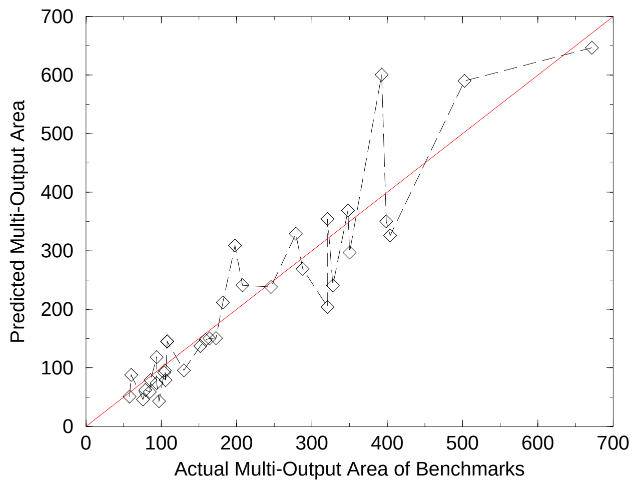

Figure 8. Comparison between actual versus predicted area at minimum-area point.

The execution time required by our area estimation tool is also given in Table 1, in cpu seconds on a SUN sparc5 with 24 MB RAM. We compared these run times, on the above benchmarks, with one run of SIS using *script.rugged* followed by SIS technology mapping. The speedup obtained is shown in Fig. 9. The figure shows a speedup between 2x and 36x. Two important observations are in order. The proposed area model is implementation independent. Hence for a given function only one run is required to estimate its area. However, in practice, several runs of SIS might be required to build a reasonable confidence that the implementation is in fact near area-optimal. Hence the speedup obtained in practice could be significantly larger. Also, speedup of 10x was obtained on large benchmarks like $s35932$ and $s13207^*$. It must be kept in mind that the reported time for $s35932$ was obtained after the circuit was partitioned. Strictly speaking the circuit was not completed in SIS. Hence we believe that on large benchmarks the speedups that can be obtained in practice can be significant. A side observation to be made is that a significant portion of the run time of the area estimation tool was spent on computation of the prime implicants in a minimal cover using *espresso* [11].

7. Estimation of $\mathcal{C}_{avg}$

In order to estimate the power, one needs to estimate not only the area complexity but also $\mathcal{C}_{avg}$, which is the average node capacitance in a circuit. If $\mathcal{C}_{tot}$ is the total circuit capacitance of an optimal area implementation and $\mathcal{A}$ is the number of gates, then:

$$\mathcal{C}_{avg} = \frac{\mathcal{C}_{tot}}{\mathcal{A}} \tag{15}$$

Table 2. Actual versus predicted areas
for benchmark circuits at minimum-area point.

CIRCUIT Name	Act. area Gate count	Pred. area Gate count	Abs. error %
s298	58	51	12.1
c8	60	88	46.67
s386	76	46	39.50
b9	79	62	21.5
s444	85	59	30.5
s400	86	79	8.1
cht	94	118	25.5
s510	94	74	21.2
term1	97	43	55.6
s526	104	92	11.5
s526n	105	96	8.6
ttt2	106	79	25.4
s641	108	145	35.2
s713	108	145	35.2
apex7	130	96	26.1
s832	152	137	9.86
s820	159	148	6.90
x1	164	150	8.5
alu2	173	151	12.7
i6	182	212	16.5
example2	198	302	56.1
x4	208	241	15.8
s953	246	238	3.20
s1196	279	329	17.9
s1238	288	269	6.60
s1494	321	204	36.4
i7	321	354	10.3
s1488	328	241	26.5
vda	348	369	6.03
alu4	350	297	15.1
frg2	393	601	52.9
x3	399	350	12.3
apex6	404	326	19.3
i8	503	590	17.3
k2	672	647	3.70
s13207*	1367	1045	23.56
s35932	7252	8761	20.81
Average			21.65

This quantity depends on the target gate library and on the fan-out structure of the circuit. Conceivably, one can make a rough estimate $\mathcal{C}_{avg}$ by averaging the intrinsic output capacitance of gates in the target library. In order to make this estimate more accurate, the averaging would have to be weighted according to the frequency of use of the gates in typical designs. It is not unreasonable to consider that several prior designs may be available from which to obtain this data. To make the estimation even more accurate, one needs to consider the fanout structure of the circuit and to add to the output capacitance of each gate the capacitance due to the fanout branches. This is the method which we use. In this case, $\mathcal{C}_{avg}$ becomes truly node capacitance and not just logic-gate capacitance. In order to estimate this, it is assumed that one has access to a few area optimal circuit implementations in the desired target library. This does not appear to be an unreasonable assumption. In this case, an estimate of $\mathcal{C}_{avg}$ can be obtained by performing an average of the $\mathcal{C}_{avg}$ estimates obtained from the area optimal circuit implementations.

In order to test the accuracy of this approach, only a few benchmarks from the benchmark set listed in Table 1 were used to obtain an estimate of $\mathcal{C}_{avg}$. These benchmarks were $s13207^*$, $s35932$ (without outputs with support set size less than or equal to three), $k2$ and $i8$. This estimated value of $\mathcal{C}_{avg}$ was used to compute $\mathcal{C}_{tot}$, assuming that the exact value of $\mathcal{A}$ was available. The estimated value of $\mathcal{C}_{tot}$ was compared

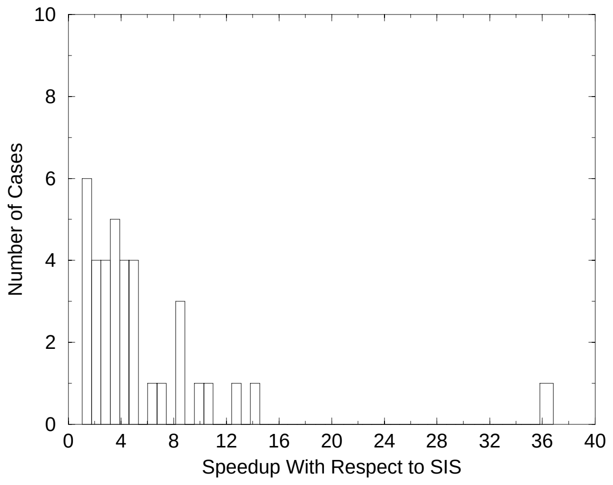

Figure 9. Speed-up versus Number of Cases.

with the true value of $\mathcal{C}_{tot}$, for the benchmark set, and the results are shown in Fig. 10, which validates the above estimation procedure for $\mathcal{C}_{avg}$.

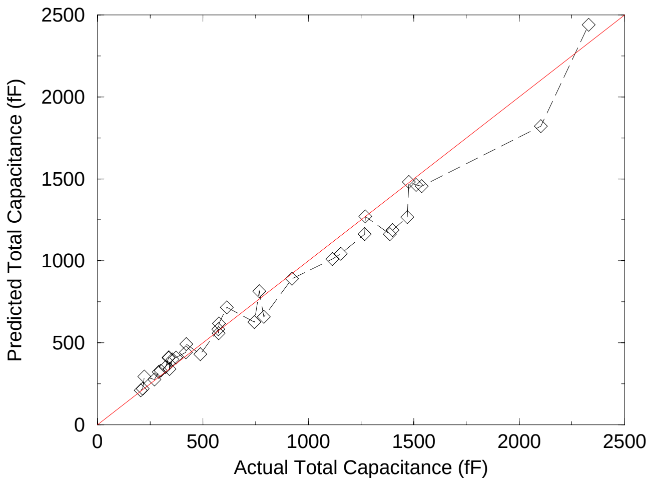

Figure 10. Error between actual and estimated values of $\mathcal{C}_{tot}$ assuming $\mathcal{A}$ is known.

The above computed estimate of $\mathcal{C}_{avg}$ works well in general. However, for circuits containing a large number of outputs with support set size less than or equal to three, the above value of $\mathcal{C}_{avg}$ can lead to an over-estimation of $\mathcal{C}_{tot}$. This over-estimation problem can be fixed by using smaller values of $\mathcal{C}_{avg}$ for estimating the capacitance of the various sub-functions of f_1 (sub-function comprising of all outputs with support set size less than or equal to three), namely functions with support set size of one, two and three. As an example, the $\mathcal{C}_{avg}$ for functions with support set size three can be determined by performing an average of this quantity over several randomly generated three input functions. The value of $\mathcal{C}_{avg}$ for functions with support set size of one and two can be similarly determined. Thus we have the following:

$$\mathcal{C}_{tot} = \mathcal{A}(f_2)\mathcal{C}_{avg} + \beta|f_1^1|\mathcal{C}_{avg}^1 + 2|f_1^2|\mathcal{C}_{avg}^2 + 3|f_1^3|\mathcal{C}_{avg}^3 \tag{16}$$

Here $\mathcal{C}_{avg}$ can be thought of as the average node capacitance of a function whose outputs have a support set size greater than three, and $\mathcal{C}^{i}_{avg}$ is the average node capacitance of a function with support set size of i, where $i \in \{1, 2, 3\}$.

The estimated value of $\mathcal{C}_{avg}$ was combined with the estimated area complexity of Boolean functions to obtain an estimate of the total capacitance of the Boolean function, $\mathcal{C}_{tot}$. The plot comparing the actual versus predicted values of $\mathcal{C}_{tot}$, when both $\mathcal{A}$ and $\mathcal{C}_{avg}$ are estimated, is shown in Fig. 11.

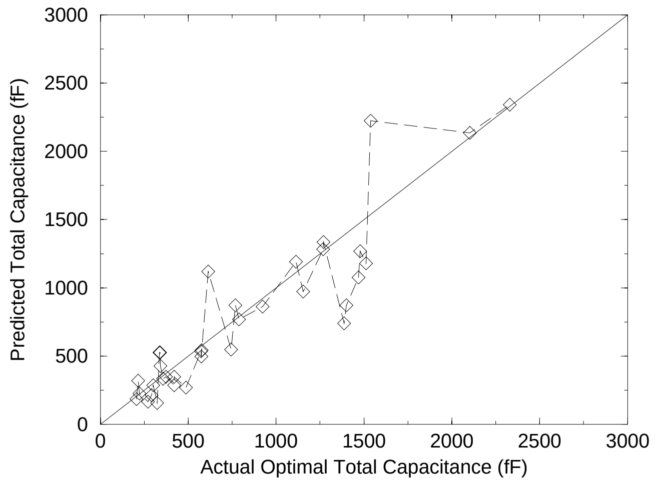

Figure 11. Actual versus Estimated values of $\mathcal{C}_{tot}$

The above approach can be adopted to estimate $\mathcal{C}_{avg}$ at any delay point on the area-delay curve (see Fig. 12). We observed that the value of $\mathcal{C}_{avg}$ decreases as we move from the minimum-area to minimum-delay point (see Fig. 12). This could be because simple gates, as opposed to complex gates like *aoi's* and *oai's*, may be preferred to implement a faster design.

8. Extensions to Area Model

So far we have looked at a prediction scheme to estimate the minimum area required to implement the function. However, there are many possible realizations of a Boolean function depending on the delay requirements, i.e., the area required to implement a Boolean function optimally, depends on the delay constraint on the Boolean function, as shown in Fig. 12. Hence in order to have meaningful power estimation one must estimate the area (and hence capacitance) as a function of the delay constraints on the function. In this section we will present extensions to the basic area estimation approach that allow one to estimate the area at any feasible delay point.

The different realizable delays of a Boolean function can be expressed in terms of a dimensionless parameter λ, such that $\lambda = 0$ corresponds to the minimum delay point (t_0), $\lambda = 100$ corresponds to the maximum delay point (t_{100}), and every intermediate value of λ, between 0 and 100, corresponds to a specific delay between the minimum and maximum delays. Specifically, if t_d is a feasible delay specification of a Boolean function, as shown in Fig. 12, whose minimum delay is equal to t_0 and maximum delay is equal to t_{100}, i.e., $t_0 \le t_d \le t_{100}$, then t_d can be expressed in terms of λ (in %) as follows:

$$t_d = \frac{\lambda}{100} t_{100} + (1 - \frac{\lambda}{100}) t_0 \tag{17}$$

Note that the RGBF curves which have been described as part of the basic area estimation approach correspond to the minimum area point, i.e., to $\lambda = 100$. The curves at $\lambda = 0$, i.e., the minimum delay

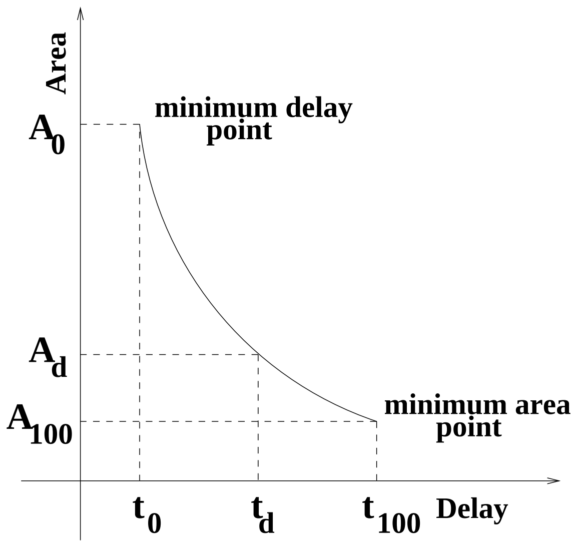

Figure 12. Feasible delay realizations of a Boolean function.

point, are built in a similar fashion, except that the RGBF functions are synthesized to have minimum delay instead of minimum area. Hence to obtain the curves for an intermediate value of λ, the RGBF functions are synthesized to the delay corresponding to the value of λ, as given by equation (17). As a result, the area complexity model, capable of predicting the area complexity at any feasible delay point, contains a family of curves parameterized in terms of delay parameter λ.

We derived the RGBF curves for three different values of λ, namely, 0%, 50% and 100%. In practice, this granularity may be enough, although this would really depend on the application. It is definitely possible to generate the curves for any value of λ. The curves for these three values of λ require about 36KB of memory. While this memory cost is quite reasonable, the total computational effort can be significant, due to the necessity to make so many synthesis runs. It took us a couple of days to generate the curves for each value of λ (using SIS), having spread the computational effort over a few workstations. However, this is a one-time cost associated with using a specific gate library.

In [21] we demonstrate that the parameter λ for a multi-output function is approximately the same as λ of its transformed single-output counterpart. Given a multi-output Boolean function and a specified value of λ, the estimation procedure is as follows. Construct the single-output Boolean function counterpart, using a multiplexor as usual, then compute its output entropy and complexity measure. Then, look up the appropriate (for the given λ) set of RGBF curves to get the area complexity of the transformed function, followed by area recovery to obtain the area complexity of the multi-output function. Recent results [22] show that it is also possible to carry out this procedure starting with a specification of t_d instead of λ.

Empirical results will now be reported at two specific delay points on the area-delay trade-off curve, corresponding to $\lambda = 0$ and $\lambda = 50\%$. As we have done before, the area of a multi-output function will be estimated by summing up the areas of its 16-output sub-functions. However, a more careful analysis is possible, based on the separate area-delay trade-off curves of the 16-output sub-functions, as described in [21].

In order to generate the minimum delay implementations, the functions were optimized in SIS using the script *script.rugged* followed by *script.delay* [13,14] for delay optimization, and mapped using the library *lib2.genlib*. The results comparing the actual area of the overall function with the predicted area, at the minimum delay point, are given in Fig. 13. The area estimation results have also been tabulated in Table 3. The average error in area estimation was 21.07%, which is close to the average error obtained at the minimum-area point. The results were reasonably accurate for all except *i8*. In the case of *i8*, there was an over-estimation by about 250 gates due to the conservative approach of simply adding the areas of the 16-output sub-functions. Also note that the two circuits, *s*35932 and *s*13207*, are missing from Table 3 because they

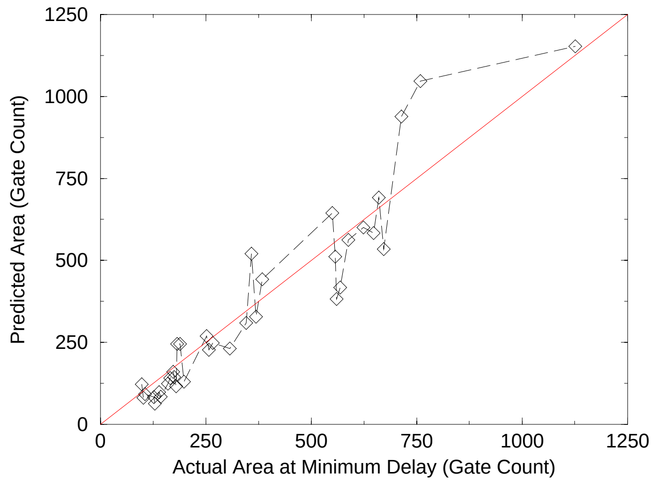

Figure 13. Comparison between actual and predicted areas at the minimum-delay point.

could not be synthesized in SIS at the minimum-delay point.

In order to generate the 50% delay implementations, the functions were first optimized in SIS using *script.rugged* for area optimization, and mapped using *lib2.genlib*. This was followed by speeding up the circuit to the 50% delay point using the command *speed_up* [14] in the SIS environment. In Fig. 14 we compare the actual and predicted area for the benchmarks at the mid-delay point and these area numbers have been tabulated in Table 4. It can be seen from this table that the average percentage error in area estimation at the 50% delay point is 22.18%. Also, note that seven circuits, namely, $alu4$, $i6$, $i7$, $i8$, $frg2$, $s35932$ and $s13207^*$, are missing from Table 4 because they could not be synthesized in SIS at the 50% delay point.

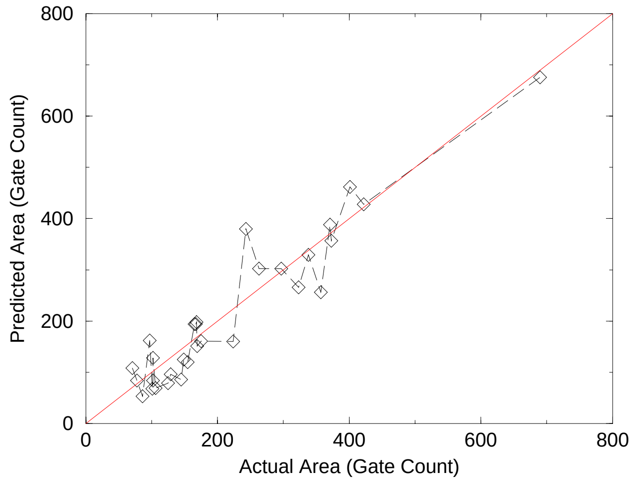

Figure 14. Comparison between actual and predicted areas at the 50% delay point.

These results indicate that the proposed area complexity model can be used to make area predictions at any feasible delay realization of the given Boolean function. While using traditional logic synthesis methods,

Table 3. Actual versus predicted areas
for benchmark circuits at minimum-delay point.

CIRCUIT Name	Act. area Gate count	Pred. area Gate count	Abs. error %
c8	98	122	24.50
s298	102	81	20.6
b9	106	92	13.2
term1	127	83	34.6
s386	129	63	51.1
s400	138	98	29.5
s444	143	84	41.2
s510	160	124	22.5
s526	166	141	15.1
cht	173	160	7.50
s526n	175	144	17.7
ttt2	180	116	35.6
s641	189	245	29.6
s713	189	245	29.6
apex7	198	130	34.3
s832	252	269	6.70
x1	257	227	11.7
s820	267	248	7.11
alu2	307	231	24.8
i6	346	309	10.7
example2	358	520	45.2
x4	369	328	11.1
s953	384	442	15.1
s1196	550	645	17.2
s1238	557	511	8.20
s1494	560	382	31.2
s1488	569	417	26.7
i7	588	562	4.4
alu4	624	600	3.8
x3	648	583	10.0
vda	660	692	4.80
apex6	672	534	20.5
frg2	714	939	31.5
i8	759	1047	37.9
k2	1126	1153	2.40
Average			21.07

each area evaluation at a feasible delay point would require a separate run of SIS, using our model the area at all delay points of interest can be obtained in one shot. This we believe, is a major advantage of this high-level approach.

Using the proposed techniques for estimating C_{avg}, discussed in section 7, one can obtain an estimate of C_{avg} at every feasible delay point. Using these estimates of C_{avg} along with the area estimates, one can obtain capacitance estimates at any delay point, which can in turn be converted into high-level power estimates.

9. High-Level Power Estimation

In the previous sections we addressed the problem of estimating the area complexity of a multi-output Boolean function. This estimate can in turn be used to estimate the power dissipated by a Boolean function, by combining it with average activity estimates [1, 3] and the average node capacitance estimate. In this section we present results on high-level power estimates by comparing them with the power dissipated by a gate level optimal implementations of the Boolean function, at the minimum-area and minimum-delay points, under two different timing models, namely, a zero-delay model and a general-delay timing model. In the case of the *general-delay* timing model the delays were obtained from a gate library and an event

Table 4. Actual versus predicted areas for benchmark circuits at 50% delay point.

CIRCUIT Name	Act. 50% area Gate count	Pred. 50% area Gate count	Abs. error %
c8	71	108	52.11
s298	106	69	34.90
b9	78	84	7.69
term1	101	68	32.67
s386	86	53	38.4
s400	102	84	17.6
s444	125	79	36.8
s510	129	96	25.6
s526	149	125	16.1
cht	97	162	67.1
s526n	102	128	25.5
ttt2	145	86	40.69
s641	165	194	17.58
s713	167	194	16.17
apex7	155	120	22.58
s832	169	151	10.65
x1	168	198	17.8
s820	175	161	8.00
alu2	224	160	28.57
example2	243	380	56.4
x4	297	302	1.68
s953	263	302	14.82
s1196	373	357	4.29
s1238	338	329	2.67
s1494	323	266	17.65
s1488	357	256	28.29
x3	401	462	15.21
vda	371	388	4.58
apex6	422	428	1.42
k2	690	676	2.03
Average			22.18

driven simulation was performed. The estimated average activity was combined with the estimates of total capacitance to obtain an estimate of the power dissipated. In sub-section 9.1 we will discuss the power estimation results at the minimum-area point and, in sub-section 9.2 we will discuss the results obtained at the minimum-delay point.

9.1. Results at Minimum-Area Point

It must be noted that the activity prediction model (3) does not account for the increase in switching activity due to glitches, as is probably to be expected from a high-level model. Hence it is important to check the accuracy of the high-level power model against the zero-delay simulation results. The actual and the predicted zero-delay power values for the benchmark circuits of Table 1, at an input probability of 0.5, are tabulated in Table 5. The average percentage error obtained, at this input probability, is equal to 32.16%. Since the activity prediction model (3) depends on the input switching statistics of the circuit, we varied the signal probabilities at the circuit inputs from 0.1 to 0.9 and computed the actual and predicted zero-delay powers. This is shown in Fig. 15. Note that each benchmark circuit is represented by a number of data points in the figure. The average percentage error between the actual and the predicted zero-delay power over the range of input probabilities, from 0.1 to 0.9, was measured to be 32.9%.

In Table 6 a comparison is shown between the actual general-delay power and the predicted power, at an input probability of 0.5. The average estimation error was equal to 30.95%. We also compared the predicted power against the general-delay simulation results for input probabilities ranging from 0.1 to 0.9. This is shown in Fig. 16. The average estimation error in this case was 33.7%.

Table 5. Actual versus predicted zero-delay power for benchmarks at minimum-area point for input probability of 0.5.

CIRCUIT Name	Act. ZD power uW	Pred. ZD power uW	Abs. error %
s298	151.7	126.14	16.93
c8	119.40	207.10	73.45
s386	65.90	41.30	37.33
b9	169.8	152.14	10.40
s444	127.8	95.17	25.53
s400	131.7	125.2	4.93
cht	380.5	458.94	20.61
s510	161.6	81.2	49.75
term1	180.6	72.54	59.83
s526	201.1	237.1	17.90
s526n	180.7	191.56	6.00
ttt2	187.4	129.45	30.92
s641	87.60	127.1	45.10
s713	81.70	124.86	52.83
apex7	188.20	166.80	11.37
s832	217.7	142.83	34.40
s820	218.3	155.76	6.90
x1	310.77	308.70	0.64
alu2	108.8	108.11	0.63
i6	234.1	248.88	6.30
example2	261.0	432.88	65.66
x4	298.6	340.7	14.1
s953	196.3	230.24	17.29
s1196	289.0	318.12	10.07
s1238	240	215.3	10.29
s1494	463	303.63	34.4
i7	427.7	408.31	4.53
s1488	482.7	349.1	27.68
vda	219	583.35	166.37
alu4	182.1	196.24	7.76
frg2	330.9	510.48	54.27
x3	690.4	605.90	12.24
apex6	510.3	435.14	14.73
i8	586.36	700.84	19.47
k2	200	501.37	150.68
s13207*	1274.3	1060.1	16.8
s35932	1740.1	2266.24	30.24
Average			32.16

9.2 Results at Minimum-Delay Point

In Table 7, we compare the predicted power with the actual power dissipated by the gate-level optimum-delay implementation under zero-delay conditions at an input probability of 0.5. As seen in the table, the average estimation error was 30.21%. It must be noted that two circuits, namely, $s35932$ and $s13207$*, are missing from the table because they could not be synthesized in SIS at the minimum-delay point. In Fig. 17, we compare the actual and predicted power values for input probabilities ranging from 0.1 to 0.9. The average estimation error for this range of input probabilities was measured to be 30.1%.

In Table 8 we compare the error between the actual power under general-delay conditions with the predicted power for an input probability of 0.5, and in Fig. 18 we compare these quantities over an input activity range of 0.1 to 0.9. The estimation errors obtained were 31.91% and 31.2% respectively.

It must be noted that average estimation error at the minimum-area and minimum-delay points for the benchmark circuits is approximately the same. Also, for 80% of the circuits, the relative estimation error in zero-delay power between the minimum-area and minimum-delay implementations was within 25%. Hence it can be concluded that the proposed high-level power estimation approach is relatively accurate across different implementations.

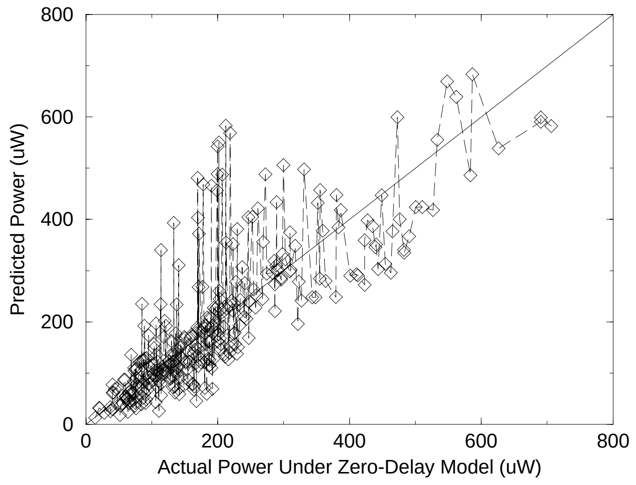

Figure 15. Comparison between actual zero-delay power and predicted power at minimum-area point.

Finally, before leaving this section, we would like to discuss the *relative* accuracy of the proposed approach. In order to get a feel for relative accuracy, we compared the ratio of actual powers at the minimum-area and minimum-delay points with the ratio of the predicted powers at the same points. The results of this comparison are summarized in Table 9. It can be seen from this table that the average error of this comparison is 15.68%. Based on this, we believe that the proposed approach preserves relative accuracy reasonably well.

10. Conclusions

We have presented a new area estimation approach to predict the area complexity of multi-output Boolean functions. This was based on transforming the multi-output function to an equivalent single-output function. The advantage of this area model is that it can be easily characterized, and it also offers a natural framework to account for sharing occurring in a multi-output function. Moreover, the utility of the area model in predicting the area of a multi-output Boolean function at any feasible delay point has been demonstrated. We have also proposed a methodology for estimating $\mathcal{C}_{avg}$ needed to convert a gate count estimate of area complexity into an estimate of total capacitance. The predicted capacitance was then combined with average activity estimates to get high level power estimates.

References

[1] F. Najm, "Towards a High-Level Power Estimation Capability," *ACM/IEEE International Symposium on Low-Power Design*, pp. 87–92, 1995.

[2] D. Marculescu, R. Marculescu and M. Pedram, "Information theoretic measures for power analysis," *IEEE Transactions on Computer Aided Design of Integrated Circuits and System*, vol. 15, no. 6, pp. 599-610, 1996.

[3] M. Nemani and F. Najm, "Towards a high-level power estimation capability," *IEEE Transactions on Computer Aided Design of Integrated Circuits and System*, vol. 15, no. 6, pp. 588-589, June 1996.

[4] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell System Technical Journal*, val. 28, no. 1, pp. 59-98, 1949.

Table 6. Actual general-delay power versus predicted power for benchmarks at minimum-area point for input probability of 0.5.

CIRCUIT Name	Act. GD power uW	Pred. power uW	Abs. error %
s298	181.3	126.14	30.30
c8	160.46	207.10	29.10
s386	83.23	41.30	50.38
b9	195.7	152.17	22.24
s444	168.3	95.17	43.45
s400	162.3	125.2	22.86
cht	432.1	458.94	6.20
s510	222.8	81.2	63.55
term1	234.46	72.54	69.06
s526	237.70	237.1	0.25
s526n	213.0	191.56	10.06
ttt2	242.54	129.45	46.63
s641	117.24	127.1	8.40
s713	110.85	124.86	12.61
apex7	228.40	166.80	26.97
s832	287.65	142.83	50.34
s820	288.45	155.76	46.0
x1	353.84	308.70	12.75
alu2	192.24	108.11	43.76
i6	321.89	248.88	22.68
example2	351.38	432.88	23.19
x4	356.36	340.7	4.40
s953	249.24	230.24	7.63
s1196	374.22	318.12	15.00
s1238	318.56	215.3	32.41
s1494	622.8	303.63	51.20
i7	586.51	408.31	30.38
s1488	649.9	349.1	46.28
vda	337.12	583.35	73.04
alu4	310.24	196.24	36.74
frg2	425.45	510.48	20.00
x3	850.9	605.90	28.79
apex6	657.3	435.14	33.80
i8	850.74	700.84	17.62
k2	282.20	501.37	77.68
s13207*	1453.8	1060.1	27.1
s35932	2317.6	2266.24	2.21
Average			30.95

[5] N. Pippenger, "Information theory and the complexity of Boolean functions," *Mathematical Systems Theory*, vol. 10, New York: Springer-Verlag Inc., pp. 129–167, 1977.

[6] K-T Cheng and V. Agrawal, "An entropy measure for the complexity of multi-output Boolean functions," *27th ACM/IEEE Design Automation Conference*, pp. 302–305, 1990.

[7] A. C-H. Wu, V. Chaiyakul and D. D. Gajski, "Layout area models for high level synthesis," *International Conference on Computer Aided Design*, pp. 34-37, 1991.

[8] F. J. Kurdahi, D. D. Gajski, C. Ramachandran and V. Chaiyakul, "Linking register transfer and physical levels of design,"*IEICE Transactions on Information and Systems*, September 1993.

[9] F. Najm, "Statistical Estimation of the Signal Probability in VLSI Circuits," Coordinated Science Laboratory Report #UILU-ENG-93-2211, April 1993.

[10] M. Xakellis and F. Najm, "Statistical Estimation of the Switching Activity in Digital Circuits," *31st ACM/IEEE Design Automation Conference*, pp. 728–733, 1994.

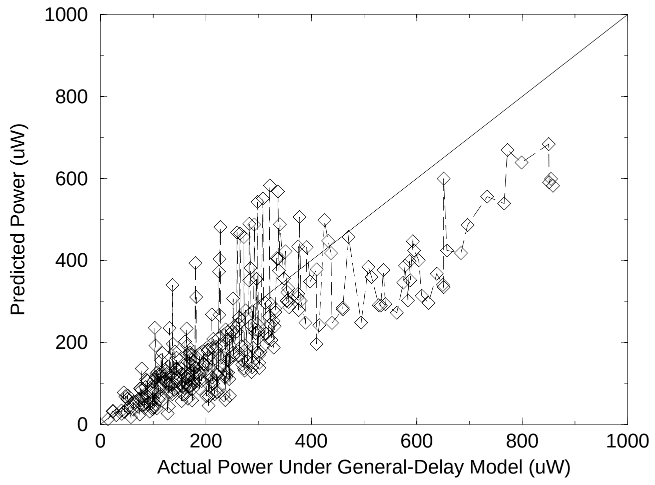

Figure 16. Comparison between actual general-delay power and predicted power at minimum-area point.

[11] R. K. Brayton, G. D. Hachtel, C. T. McMullen and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[12] M. Nemani and F. Najm, "High-Level Area Prediction for Power Estimation," *Custom Integrated Circuits Conference*, 1997.

[13] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, New York, NY: McGraw-Hill Inc., 1994.

[14] *SIS-1.2, Reference Manual*, University of California, Berkeley, 1992.

[15] F. Brglez, D. Bryan and K. Koźmiński, "Combinational profiles of sequential benchmark circuits," *IEEE International Symposium On Circuits and Systems*, pp. 1929-1934, 1989.

[16] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," *Rep. Microelectronics Center of North Carolina*, 1991.

[17] M. Nemani and F. Najm, "High Level Area and Power Estimation of VLSI Circuits," *IEEE/ACM International Conference on Computer Aided Design*, pp. 114-119, 1997.

[18] D. E. Muller, "Complexity in electronic switching circuits," *IRE Transactions on Electronic Computers*, vol. 5, pp. 15-19, 1956.

[19] E. Kellerman, "A Formula for Logical Network Cost," *IEEE Transactions on Computers*, vol. 17, no. 9, 881-884, 1968.

[20] R. W. Cook and M. J. Flynn, "Logical network cost and entropy," *IEEE Trans. on Computers*, vol. 22, no. 9, 823-826, 1973.

[21] M. Nemani, "High-level power estimation," Ph.D dissertation, Dep. of Electrical and Computer Engineering, Univ. of Illinois at Urbana-Champaign, May 1998.

[22] M. Nemani and F. Najm, "Delay Estimation of VLSI Circuits from a High-Level View," *IEEE/ACM Design Automation Conference*, pp. 591-594, 1998.

[23] S. Powell and P. Chau, "Estimating power dissipation of VLSI signal processing chips: The PFA technique," *VLSI Signal Processing (H. S. Moscovitz, ed.)*, pp. 250-259, New Jersey: IEEE Press, 1990.

[24] S. Powell and P. Chau, "A model for estimating power dissipation in a class of VLSI chips," *IEEE Transactions on Circuits and Systems*, pp. 646-650, 1991.

Table 7. Actual versus predicted zero-delay power values for benchmarks at minimum-delay point for input probability of 0.5.

CIRCUIT Name	Act. GD Power uW	Pred. Power uW	Abs. error %
c8	259.49	397.40	53.14
s298	275.66	244.12	11.44
b9	285.78	212.58	25.61
term1	286.21	139.35	51.3
s386	196.30	66.21	66.3
s400	285.44	217.28	23.9
s444	275.93	166.45	39.7
s510	296.50	209.64	29.3
s526	351.03	355.18	1.18
cht	594.98	602.13	1.20
s526n	393.86	396.62	0.70
ttt2	386.05	255.37	33.85
s641	251.15	320.39	27.56
s713	280.11	340.85	21.68
apex7	405.87	291.07	28.28
s832	444.30	290.29	34.66
x1	678.16	534.71	21.15
s820	475.14	304.48	35.92
alu2	343.69	265.75	22.67
i6	1302.64	1115.44	14.37
example2	656.57	803.44	22.37
x4	602.09	663.12	10.13
s953	495.51	531.76	7.30
s1196	843.60	810.52	3.90
s1238	800.38	672.07	16.03
s1494	1009.99	649.41	35.7
s1488	1098.87	712.99	35.11
i7	1950.48	1667.22	14.54
alu4	494.18	609.54	23.34
x3	1303.24	1052.80	19.21
vda	588.52	1345.49	128.6
apex6	1083.56	736.02	32.07
frg2	946.91	1377.47	45.47
i8	1196.23	1854.67	55.04
k2	652.09	1073.51	64.60
Average			30.21

[25] P. Landman and J. Rabaey, "Architectural power analysis: The dual bit model,"*IEEE Transactions on VLSI Systems*, vol. 3, no. 2, pp. 173-187, 1995.

[26] P. Landman and J. Rabaey, "Activity-sensitive architectural power analysis,"*IEEE Transactions on Computer Aided Design of Integrated Circuits and System*, vol. 15, no. 6, pp. 571-587, June 1996.

[27] A. Raghunathan, S. Dey, and N. K. Jha, "Register-transfer level techniques for switching activity and power consumption,"*IEEE/ACM International Conference on Computer Aided Design*, pp. 158-165, 1996.

[28] S. Gupta and F. Najm, "Power macro-modeling for high-level power estimation,"*Proc. IEEE/ACM Design Automation Conference*, pp. 365-370, 1997.

[29] D. Wallace and M. Chandrasekhar, "High-level delay estimation for technology independent logic equations,"*IEEE/ACM International Conference on Computer Aided Design*, pp. 188-191, 1990.

[30] H. Mehta, R. M. Owens, and M. J. Irwin, "Energy characterization based on clustering," *33rd Design Automation Conference*, pp. 702–707, June 3–7 1996.

[31] Q. Qiu, Q. Wu, M. Pedram, and C.-S. Ding, "Cycle-accurate macro-models for RT-level power analysis," *1997 International Symposium on Low Power Electronics and Design*, pp. 125–130, August 18–20 1997.

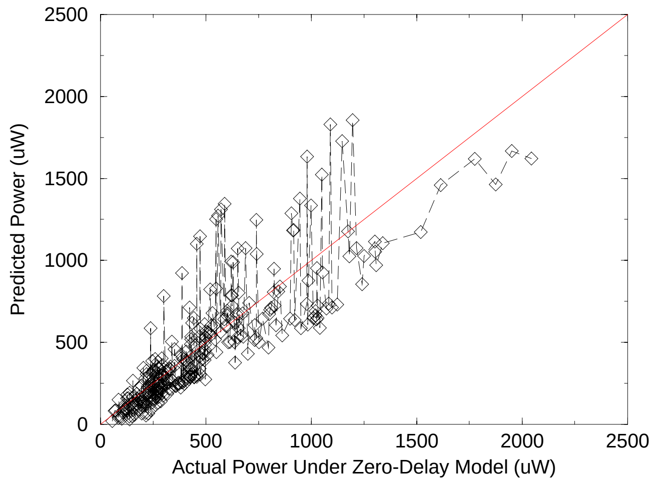

Figure 17. Comparison between actual zero-delay power and predicted power at minimum-delay point.

Table 8. Actual general-delay power versus predicted power for benchmarks at minimum-delay point for input probability of 0.5.

CIRCUIT Name	Act. GD Power uW	Pred. Power uW	Abs. error %
c8	318.60	397.40	24.73
s298	308.61	244.12	20.89
b9	321.13	212.58	33.80
term1	342.09	139.35	59.3
s386	226.53	66.21	70.8
s400	350.05	217.28	37.9
s444	340.19	166.45	51.1
s510	366.54	209.64	42.8
s526	412.40	355.18	13.8
cht	695.89	602.13	13.5
s526n	443.12	396.62	10.49
ttt2	464.80	255.37	45.06
s641	292.40	320.39	9.57
s713	354.62	340.85	3.88
apex7	493.50	291.07	41.02
s832	541.39	290.29	46.38
x1	744.34	534.71	28.16
s820	554.59	304.48	45.09
alu2	481.43	265.75	44.8
i6	1604.35	1115.44	30.47
example2	866.32	803.44	7.26
x4	749.71	663.12	11.55
s953	618.52	531.76	14.03
s1196	1060.04	810.52	23.5
s1238	996.75	672.07	32.58
s1494	1202.54	649.41	46.0
s1488	1328.34	712.99	46.32
i7	2414.36	1667.22	30.94
alu4	724.42	609.54	15.86
x3	1626.27	1052.80	35.26
vda	737.90	1345.49	82.3
apex6	1407.94	736.02	47.72
frg2	1180.9	1377.47	16.64
i8	1716.17	1854.67	8.07
k2	857.20	1073.51	25.23
Average			31.91

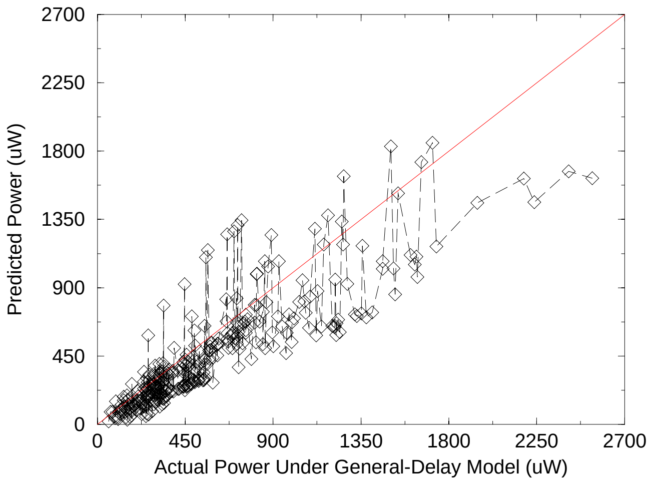

Figure 18. Comparison between actual general-delay power and predicted power at minimum delay point.

Table 9. Relative accuracy comparisons; RAP is Ratio of Actual Powers, RPP is Ratio of Predicted Powers, and ARE is Absolute value of the Relative Error between RPP and RAP.

Circuit	RAP	RPP	ARE (%)
s298	0.587	0.517	11.90
c8	0.504	0.521	3.37
s386	0.367	0.624	70.03
b9	0.609	0.716	17.57
s444	0.495	0.572	15.56
s400	0.464	0.576	24.14
cht	0.621	0.762	22.70
s510	0.608	0.387	36.30
term1	0.685	0.521	23.94
s526	0.576	0.668	15.97
s526n	0.481	0.483	0.62
ttt2	0.522	0.507	2.87
s641	0.401	0.397	1.00
s713	0.313	0.366	17.25
apex7	0.463	0.573	23.76
s832	0.531	0.492	7.34
s820	0.520	0.512	1.54
x1	0.475	0.577	21.47
alu2	0.399	0.407	2.00
i6	0.201	0.223	10.94
example2	0.406	0.539	32.76
x4	0.475	0.514	8.21
s953	0.403	0.433	7.44
s1196	0.353	0.392	11.05
s1238	0.319	0.320	0.31
s1494	0.518	0.468	9.65
i7	0.243	0.245	0.82
s1488	0.489	0.426	12.88
vda	0.457	0.434	5.03
alu4	0.428	0.322	24.77
frg2	0.360	0.371	3.06
x3	0.523	0.576	10.13
apex6	0.467	0.591	26.55
i8	0.496	0.378	23.79
k2	0.329	0.467	41.94
Average			15.68