HIGH-LEVEL POWER ESTIMATION AND THE AREA COMPLEXITY OF BOOLEAN FUNCTIONS*

Mahadevamurty Nemani and Farid N. Najm

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street, Urbana, IL 61801, USA
e-mail: najm@uiuc.edu

ABSTRACT

Estimation of the area complexity of a Boolean function from its functional description is an important step towards a power estimation capability at the register transfer level (RTL). This paper addresses the problem of computing the area complexity of single-output Boolean functions given only their functional description, where area complexity is measured in terms of the number of gates required for an *optimal* implementation of the function. We propose an area model to estimate the area based on a new complexity measure called the *average cube complexity*. This model has been implemented, and empirical results demonstrating its feasibility and utility are presented.

1. Introduction

Rapid increase in the design complexity and the need to reduce design time 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. Some of the first papers to report work on high level power estimation techniques include [1, 2, 3]. However, these papers only discuss the estimation of *average switching activity* in a circuit. In order to provide a high-level estimate of power, one should multiply the average activity value by a measure of the circuit *total capacitance.*

How can one predict total capacitance given only a high level functional view of a circuit? Among other things, circuit capacitance depends on the circuit function, the gate library and the delay constraint.

*This work was supported in part by Intel Corp. and by the Semiconductor Research Corp.

Obviously, a given Boolean function can be implemented in different ways, leading to different values of total capacitance. Thus, it seems at first glance that capacitance estimation from a high level view is impossible. Nevertheless, we have made some significant progress towards solving this problem, based on the following approach. We assume that the total capacitance is proportional to the product of two terms: 1) a technology-independent measure of circuit area, e.g., an estimate of the gate-count, and 2) a technology-dependent measure of average capacitance per gate. The second term depends on the gate library and on the delay specification, while the first term does not. In this paper, we report on work related to the estimation of the first term, hence the title "area complexity."

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 this 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 [5], 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. These include [6], [7] [8] and [9]. The model of [9], where area complexity was measured as literal count, was derived empirically for small n from *randomly generated* Boolean functions. 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, for a circuit with 32 inputs, this model predicts an area of $\approx$ 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 [5], 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-counts for optimal circuit implementations that were obtained using the SIS synthesis system.

Our estimation technique is based on the novel concept of *average cube complexity* of a Boolean function, to be introduced 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 [9]. 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. Our technique is, for now, limited to single-output Boolean functions and we are in the process of generalizing it to multiple-output functions.

Before leaving this section, it may be worthwhile, for completeness, to mention some previous work on layout area estimation. In [10], [11] and [12] layout area was estimated from gate or transistor level description of the circuit where a standard cell layout is assumed. In [13], a layout area model for datapath and control for two commonly used layout architectures was proposed. In [14], the above model was modified to account for effects of floorplanning. As the models of [13] and [14] require the SOP expression for generating a netlist, they are impractical for large circuits.

The paper is organized as follows: In section 2 we give a background discussion of the complexity of randomly generated Boolean functions. In section 3 we define the new complexity measure *average cube complexity*, used to estimate the area complexity. In section 4 we present an algorithm for computing the *average cube complexity*. We propose an area prediction model in section 5 and present empirical results in section 6 to demonstrate its utility and feasibility. We end the paper with some conclusions presented in section 7.

2. Randomly Generated Boolean Functions

We will introduce the notion of randomly generated Boolean functions, and discuss models that were previously proposed for estimating their area complexity. Throughout this paper, we will use terminology that is consistent with the definitions in [15].

It was pointed out by Shannon [4] that for large n, Boolean functions of n inputs have exponential complexity in n, based on a switch-count measure of complexity. A similar result was also shown by Pippenger [7]. While both these results are theoretical and for large n, an empirical study by Cheng et. al. [9] shows similar behavior for small n. Specifically, they observe an exponential complexity dependence on n 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 selected 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 $(1-\epsilon)\frac{2^n}{n}$ switch elements. 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 [9] applies this to every Boolean function, leading to the following area model

$$\mathcal{A} \propto 2^n \mathcal{H} \tag{1}$$

where n is the number of inputs, $\mathcal{H}$ is the entropy of the output of the Boolean function (with independent inputs, each with probability 0.5), and $\mathcal{A}$ is the area complexity measured as gate-count. The proportionality constant depends on the library being used.

Risking abuse of terminology, we will refer to a Boolean function for which the above model holds as an *average function.* Unfortunately, we have found that logic functions that are typically used in VLSI are far from being *average*, in the above sense, 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. [9]. Finding an area model for such functions has remained an open problem. This paper, to the best of the authors' knowledge, is the first to utilize the structure of the Boolean space, in addition to the entropy, to predict the area complexity. We have found that the area complexity of realistic VLSI circuits, rather than being exponential in n [9], is better modeled as being (approximately) exponential in $\mathcal{C}$, where $\mathcal{C}$ is the average cube complexity of the function, to be described below.

3. Average Cube Complexity

We start with a precise statement of the problem to be addressed. Consider an n-input single-output Boolean function $f(X)$. Given a library, we would like to estimate the *minimum* number of gates ($\mathcal{A}$) required to implement the function, given only its high level description (Boolean equations). It must be noted that we would like to compute $\mathcal{A}$ without performing any logic synthesis on the function $f(X)$. The rationale for this is that logic synthesis can be computationally expensive, and is therefore best avoided in a high-level analysis in order to maintain computational efficiency.

It seems reasonable to say that the area complexity of a Boolean function is related to the complexity of its on-set and off-set. There are many ways of quantifying the on-set/off-set complexity, the simplest being to consider the *sizes* of the on-set and off-set relative to the size of the Boolean space of the function (this is simply the probability of the function when its inputs are independent and set to a probability of 0.5, and a modification of this is the entropy of the function). However, we have found that this simple measure is too general, in the sense that many quite different functions can have the same on-set probability or entropy. Hence, a complexity measure based on more detailed analysis of the on-set/off-set of the function is required.

In order to measure the area complexity of a Boolean function, we propose to use (in addition to the entropy) the *average literal-count of the prime implicants* of the function, which we will call the *average cube complexity*. Thus the complexity of the cube $c = x_1\bar{x}_2x_4$ is 3, and we write $||c|| = 3$. Intuitively, if a Boolean function has a small *average cube complexity*, it means that it can be represented using cubes of small literal-count, which naturally require less area to implement. On the other hand, a large *average cube complexity* would imply the presence of cubes with large literal-counts, which would require a large number of logic gates to implement. In this paper we use the *average cube complexity* in conjunction with the entropy of the Boolean function to compute the area complexity of the function. In the following, we make the term *average cube complexity* more precise.

In principle, whatever measure of cube complexity is used, one should keep in mind that many other factors can influence the gate count, so that the result of using the cube complexity to predict area will involve some estimation error. For this reason, it does not make sense to use very expensive procedures for estimating the cube complexity. For instance, it would be prohibitively expensive to try and enumerate all the essential prime implicants of the function in order to compute their average complexity. Instead, we have chosen to estimate a certain weighted average of the complexity of the prime implicants, not of the essential primes. The weighted average can be efficiently computed by an iterative process of logic simulation and is guaranteed to lie between two quantities $A(f)$ and $B(f)$, either of which can itself serve as a measure of the average cube complexity and both of which are too expensive to compute directly, defined as follows.

Consider an n-input single-output Boolean function f. Let V_{ON} be the set of minterms (vertices of the Boolean space) corresponding to the on-set of the function, and let $m_i \in V_{ON}$ denote a minterm. Also, let a_i be the *largest* (i.e., contains the most minterms) *prime implicant* of f that contains the minterm m_i. Then, define $A(f)$ to be the following *average* of the $||a_i||$s over all the minterms in V_{ON}

$$A(f) = \frac{1}{|V_{ON}|} \sum_{m_i \in V_{ON}} ||a_i|| \qquad (2)$$

Similarly, define b_i to be the *smallest* (i.e., contains the least minterms) *prime implicant* of f that contains the minterm m_i. And define $B(f)$ to be the following *average* of the $||b_i||$s over all the minterms in V_{ON}

$$B(f) = \frac{1}{|V_{ON}|} \sum_{m_i \in V_{ON}} ||b_i|| \qquad (3)$$

It is clear from the definitions that $A(f) \leq B(f)$.

Each of these measures, A and B, may itself contain enough information about the population of prime implicants in the on-set of f to be useful for studying its complexity. However, even though they are defined in terms of prime implicants, and not essential prime implicants, A and B are still too expensive to compute. Instead, we will define a measure $\mathcal{C}_1$ which is very easy to compute and which lies between A and B.

For a minterm m_i, let $c_{i1}, c_{i2}, \ldots, c_{iN_i}$ be all the prime implicants that contain m_i. In the next section, we will describe a (logic simulation based) algorithm by which the different prime implicants c_{ij} can be obtained from m_i. Irrespective of how this is done, let p_{ij} denote the probability that, using our algorithm, c_{ij} is obtained from m_i. Based on this, we define the following

$$\mathcal{C}_1(f) = \frac{1}{|V_{ON}|} \sum_{m_i \in V_{ON}} \sum_{j=1}^{N_i} p_{ij} ||c_{ij}|| \qquad (4)$$

Since $\sum_{j=1}^{N_i} p_{ij} = 1$, $0 \leq p_{ij} \leq 1$, and $||a_i|| \leq ||c_{ij}|| \leq ||b_i||$, it follows that

$$A(f) \leq \mathcal{C}_1(f) \leq B(f) \qquad (5)$$

We refer to $\mathcal{C}_1(f)$ as the *average cube complexity of the on-set* of the Boolean function f.

Likewise, the *average cube complexity of the off-set*, denoted by $\mathcal{C}_0(f)$, can be defined in a similar fashion. If only $\mathcal{C}_1$ (or only $\mathcal{C}_0$) is used as a complexity measure, one runs into cases where the complexity of a function and its complement are predicted to be quite different, which is clearly unrealistic. Instead, we have found that the following composite measure works best

$$\mathcal{C}(f) = \mathcal{P}(f)\mathcal{C}_1(f) + \mathcal{P}(\bar{f})\mathcal{C}_0(f) \qquad (6)$$

where $\mathcal{P}(f)$ denotes the probability of the function f, which can be efficiently computed using Monte-Carlo techniques [16]. We refer to $\mathcal{C}(f)$ as the *average cube complexity* of the function f.

In this paper we will use $\mathcal{C}(f)$, along with the function entropy, to estimate the area complexity of a Boolean function. In order to compute $\mathcal{C}$ one needs to compute $\mathcal{C}_1$ and $\mathcal{C}_0$. It can be shown that $\mathcal{C}_1$ and $\mathcal{C}_0$ can be expressed as a mean, or weighted average, of the literal counts of all the prime-implicants of the on-set of the function. Hence, statistical techniques of *mean estimation*, i.e., Monte Carlo techniques, can be used for estimating the above quantities. In the next section we present the algorithm used for estimating $\mathcal{C}_1$. The computation of $\mathcal{C}_0$ is similar.

4. Algorithm for Computation of $\mathcal{C}_1$

The following are the steps involved in computing $\mathcal{C}_1$ for a given Boolean function $f(X)$. The algorithm takes as input an arbitrary technology independent Boolean network representation of the function. Firstly, a point in the Boolean space and in the on-set of the Boolean function $f(X)$, is chosen at random. The logic vector corresponding to this minterm is applied to the circuit inputs and a logic simulation is performed to determine the corresponding logic values at all gate outputs. Then, the process of generating a prime implicant from that minterm consists of two steps, which we call *backward* *scanning* and *forward simulation*, described below. The complexity of the generated prime implicant is used to modify the estimate of $\mathcal{C}_1$, and the process is repeated until convergence of $\mathcal{C}_1$ is obtained to a user specified accuracy and confidence. A stopping criterion used to stop the simulation can be derived using an approach similar to that in [17]. A detailed explanation of the prime-implicant generation algorithm is given below.

In the backward scanning step, all circuit nodes (including the inputs) are marked *don't-care* without actually altering their stored logic values, and the circuit is traversed from the output to the inputs, breadth first. During the traversal, all inputs to a gate whose logic values are controlling and which are marked don't-care are *unmarked.* For example, if the output of a two input OR gate is '1' with one input at logic '0' and the other at '1', the input at logic '1' is unmarked as a don't-care. If both the inputs are at logic '1', one of the inputs is randomly picked and unmarked. At the end of this step, the logic values of any primary inputs that are marked don't-care are changed to 'X' (to denote a logic *don't-care*). The resulting primary inputs assignment forms a cube (not necessarily a prime implicant) which contains the original minterm.

In order to generate a prime implicant from the above cube, we must make sure that all the inputs not marked as don't-care at the end of the backward scanning step are indeed required to evaluate the function correctly. This is done in the forward simulation step. Here, the input nodes not marked as don't-care, are selected at random and set to a don't-care and a logic simulation is performed to determine if the function evaluates to '1'. If so, then the logic value at that input is set to a *don't-care*, otherwise it is reset to its original value. This procedure is continued until all unmarked inputs have been tested, so that the cube at the end of this step is a prime implicant.

5. The Area Model

In this section we present the area model to compute the area complexity $\mathcal{A}(f)$ of Boolean functions. The area model is based on the concept of *average cube complexity* $\mathcal{C}(f)$ introduced earlier.

Empirical data based on straightforward application of the algorithm outlined above for computation of $\mathcal{C}_1(f)$ and $\mathcal{C}_0(f)$ showed us that our complexity measure works well, but it somewhat under-estimates the area complexity in many cases. We have been able to fix this bias by devising a slight modification of the algorithm, which is described below, and which may be thought of as a *tuning* of the model.

The values of $\mathcal{C}_1(f)$ and $\mathcal{C}_0(f)$ depend on the probability of occurrence of various cubes (which are determined by the algorithm). Since cubes with smaller cube complexities have a higher probability of occurrence than those with larger cube complexities, the straightforward application of the above algorithm can lead to situations where the estimates of $\mathcal{C}_1(f)$ and $\mathcal{C}_0(f)$ are severely biased towards cubes with small complexities. We have come up with a strategy of overcoming this bias in the estimate, in a way that does not hurt other good data points (benchmarks on which the model works well) of the model and maintains computational efficiency.

We have done this by placing a ceiling on the frequency (i.e., probability) with which any specific prime implicant c_i is allowed to be sampled by the algorithm. This ceiling value is the same for all prime implicants. This procedure ensures that the probability distribution is somewhat flattened on the side of cubes with smaller cube complexities and somewhat raised on the side of cubes with larger cube complexities. This raises the estimates of $\mathcal{C}_1(f)$ and $\mathcal{C}_0(f)$ and corrects the bias in the original algorithm. The above modification only alters the distribution with which certain cubes are obtained by our algorithm, but does not alter the definition and properties of the measures $\mathcal{C}_1$ and $\mathcal{C}_0$ given previously. Likewise, the convergence criterion does not change. We have found that a ceiling of 0.15 seems to produce the best results. With this modification, we have used the $\mathcal{C}(f)$ complexity measure to predict the area complexity, as follows.

We first discuss how the data corresponding to the average functions shown in Fig. 1 was generated. For a given value of n, we computed $\mathcal{C}(f)$ and obtained an estimate of the gate-count, $\mathcal{A}(f)$, from an optimized implementation for a number of randomly generated Boolean functions whose output entropy is $\mathcal{H}(f) = 1$, based on all inputs being independent and with 0.5 probability. These points, for each n, were very closely clustered. This means that the distribution of $\mathcal{A}(f)$ of randomly generated Boolean functions (given n and $\mathcal{H}$) is *tight.* It also implies that the distribution of $\mathcal{C}(f)$ is tight. The curve referring to average functions in Fig. 1 corresponds to the average values of each cluster and is close, but not exactly equal, to an exponential.

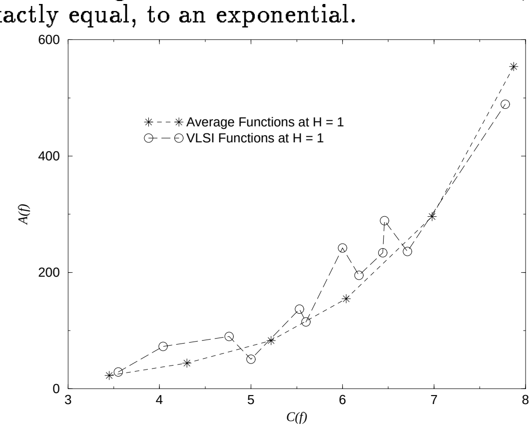

Figure 1: Typical VLSI functions fall close to the curve

This almost-exponential $\mathcal{A}$ versus $\mathcal{C}$ 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. 1. The data points shown in Fig. 1 correspond to the subset of the test cases to be presented in section 6 for which the output entropy is equal to 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 $\mathcal{C}(f)$. In fact, we use a family of such curves, corresponding to different entropy values, as shown in Fig. 2. 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.

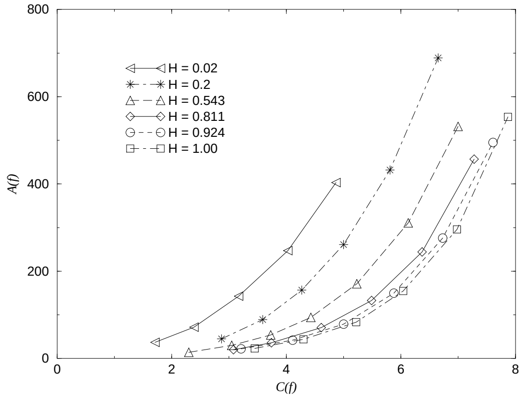

Figure 2: $\mathcal{A}(\cdot)$ versus $\mathcal{C}(\cdot)$ for different values of entropy.

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 $n \leq 8$ as in Fig. 2 is sufficient. Firstly, we have found that for typical VLSI functions, the value of $\mathcal{C}(f)$ 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 8 to 67) had $\mathcal{C}(f) \leq 8$, so that the curves in Fig. 2 were sufficient. This fact is key because it illustrates why the traditional (exponential in n) model breaks down while our (almost-exponential in $\mathcal{C}$) 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. For larger values of n, one can simply compute the area complexity as

$$\mathcal{A}(f) = 2^{\mathcal{C}(f)} k(\mathcal{H}) \tag{7}$$

where $k(\mathcal{H})$ is a proportionality constant that depends on the entropy $\mathcal{H}$.

6. Empirical Results

Before presenting the actual data, a word on how the benchmarks were generated is in order. Since most of the ISCAS-85, ISCAS-89 and MCNC benchmarks are multiple-output Boolean functions, these circuits were used to generate single-output Boolean functions by deleting all but one output. The resulting single output function was optimized using SIS, for minimum area. The script used for optimization was *rugged.script* of the SIS optimizer. Mapping was done using a library consisting of a *nand2*, *nor2* and an *inverter*. The gate count of the SIS-optimized circuit was used as the reference value for area. A number of single output circuits were generated using this method.

The area complexity values (gate-count predictions) of these benchmarks, using our model, were computed as follows. Firstly, the probability and the entropy of the Boolean function were estimated to a prescribed error tolerance and confidence statistically using a Monte-Carlo approach [16]. The probability was estimated to an accuracy of 95% with a confidence of 95%. The entropy of the output was computed using the estimated probability. Parameters $\mathcal{C}_1$ and $\mathcal{C}_0$ were estimated to an accuracy of 90% with a confidence of 90%. Equation (6) was used to estimate $\mathcal{C}(f)$ and the parameters $\mathcal{C}(f)$ and $\mathcal{H}(f)$ were then used to compute the area using the approach discussed in the previous section.

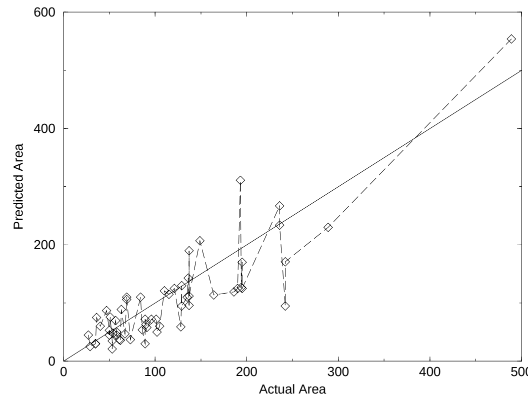

Figure 3: Actual versus Predicted Area.

The comparison between the predicted and SIS optimized gate count values is given in Fig. 3. While the agreement is not perfect, it is nevertheless reasonable, especially considering that the predicted gate counts were obtained *only* from a knowledge of the function, and no structural information or synthesis procedures were required to make those predictions. This represents a unique contribution and indicates that high-level analysis for power prediction can be realized. It must be mentioned that our current approach does not work well on circuits which can be realized as xor trees. We are currently working on this problem.

Finally, some words on run time are in order. The run times corresponding to 90% confidence with 10% error tolerance, on a SUN sparc-5 workstation, for a majority of the benchmarks was under 3 minutes. The worst case run time was for the circuit C3540_o10. For this circuit the run time was about 7 minutes. The number of iterations for $\mathcal{C}(f)$ algorithm were typically around 50 or 60 for most circuits, and only rarely did the iteration count go over 100.

7. Conclusions

Motivated by the need for high-level power estimation techniques, we have proposed a new model for predicting the area of a single-output Boolean function, given *only* its functional specification and *no* structural information. This was achieved by reformulating the area complexity problem in terms of the *average cube complexity*, which was introduced in this paper and for which an efficient algorithm was presented. The relationship between area complexity and the average cube complexity was found empirically to be almost-exponential, including a dependence on the function output entropy. Unlike other existing area models which fail on realistic VLSI circuits, this model is reasonably accurate, compared to SIS-optimized circuit implementations.

The significance of this work is that it relates a structural attribute (area) to a functional attribute (average cube complexity), which is a definite requirement for high-level power estimation. Future work includes extension of the area model in order to handle multiple-output functions.

8. References

[1] F. N. Najm, "Towards a high-level power estimation capability," *ISLPD*, pp. 87–92, 1995.

[2] D. Marculescu, R. Marculescu, and M. Pedram, "Information theoretic measures of energy consumption at register transfer level," *ISLPD*, pp. 81–86, 1995.

[3] M. Nemani and F. N. Najm, "Towards a high-level power estimation capability," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, to appear*, June, 1996.

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

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

[6] E. Kellerman, "A formula for logical network cost," *IEEE Transactions on Computers*, vol. 17, no. 9, pp. 881–884, 1968.

[7] N. Pippenger, "Information theory and the complexity of boolean functions," *Mathematical Systems Theory*, vol. 10, pp. 129–167, 1977.

[8] R. W. Cook and M. J. Flynn, "Logical network cost and entropy," *IEEE Transactions on Computers*, vol. 22, no. 9, pp. 823–826, 1973.

[9] K.-T. Cheng and V. Agrawal, "An entropy measure for the complexity of multi-output boolean functions," *DAC*, pp. 302–305, 1990.

[10] M. Pedram and B. Preas, "Interconnection length estimation for optimized standard cell layouts," *ICCAD*, pp. 390–393, 1989.

[11] G. Zimmerman, "A new area and shape function estimation technique for vlsi layouts," *DAC*, pp. 60–65, 1988.

[12] C. Ramachandran and F. J. Kurdahi, "Tele: A timing evaluator using layout estimation for high level applications," *EDAC*, 1992.

[13] A. C.-H. Wu, V. Chaiyakul, and D. D. Gajski, "Layout-area models for high-level synthesis," *ICCAD*, pp. 34–37, 1991.

[14] 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*, Sept., 1993.

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

[16] F. N. Najm, "Statistical estimation of the signal probability in vlsi circuits," *CSL Report #UILU-ENG-93-2211*, April 1995.

[17] M. Xakellis and F. Najm, "Statistical estimation of the switching activity in digital circuits," *DAC*, pp. 728–733, 1994.

Table 1: Analysis of the area Model

Circuit name	Input #	Opt. area # Gates	Model Pred.
s1494_o17	11	50	53
c2670_o1	16	51	76
s1196_o5	17	54	48
frg1_d0	25	58	68
s1238_o5	17	58	50
s1488_o1	14	61	37
s1494_o1	14	62	36
cm150a	21	63	89
s1488_o19	14	67	47
c880_o18	29	69	110
c5315_o7	25	69	106
s1488_o24	10	73	37
c880_o21	32	84	110
s1488_o12	14	86	54
s1238_o19	19	89	73
c5315_o25	28	90	63
s1494_012	14	91	58
s510_o6	20	96	72
c5315_o85	30	101	72
s1494_o18	14	105	60
c880_o19	36	110	121
rot_p4	46	102	50
alu2_k	8	115	115
s1238_o22	21	121	125
myadder_x0	33	128	59
c5315_o75	33	129	122
dalu_O15	42	129	95
c2670_o39	35	129	525
s1196_o17	21	135	110
s1196_o22	21	136	143
alu4_o	8	137	112
dalu_O0	29	137	190
rot_q5	55	137	96
dalu_O5	39	149	207
dalu_O10	41	164	114
c880_o24	45	186	119
c5315_o34	55	190	125
s1196_o20	21	193	311
alu2_o	10	194	127
s1238_o23	21	195	170
c1908_o1	32	236	267
c1908_o8	32	236	235
c1908_o6	32	242	171
c5315_o37	67	242	95
alu4_p	10	289	230
alu4_r	14	489	554