October 2002

UILU-ENG-02-2225
DAC 95

ON THE TUNABILITY OF A HIGH-LEVEL AREA MODEL

Kavel M. Buyuksahin and Farid N. Najm

Coordinated Science Laboratory
1308 West Main Street, Urbana, IL 61801
University of Illinois at Urbana-Champaign

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-02-2225 (DAC 95)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION SRC
6c. ADDRESS (City, State, and ZIP Code) 1308 W Main St Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Brighton Hall, Suite 120 1101 Slater Rd Durham, NC 27703
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SRC 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Brighton Hall, Suite 120 1101 Slater Rd Durham, NC 27703	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) On the Tunability of a High-Level Area Model	
12. PERSONAL AUTHOR(S) Buyuksahin, Kavel M. and Najm, Farid N.	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) 02 Sep 09 15. PAGE COUNT 13
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Power estimation, high-level prediction, design exploration, characterization, tuning

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

One of the most important features of a high-level complexity model is its tunability, because it helps reduces the characterization overhead of the method substantially, and makes it much more practical to use. Tunability of the model ensures that for any given synthesis tool, we need to perform the computationally expensive characterization step only once (using a simple target library and a simple synthesis script), and use the resulting model for any combination of target libraries and synthesis scripts with only a minimal effort spent on tuning. In this report, we will demonstrate the tuning method and the results obtained by tuning. Our tuning methodology is very similar to the one used by Bogliolo for RTL power models of soft macros.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete.*

On the Tunability of a High-Level Area Model

Kavel M. Büyükşahin and Farid N. Najm

ECE Dept. & Coordinated Science Lab.
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

1. Introduction

One of the most important features of our high-level complexity model ([1]) is its tunability. That reduces the characterization overhead of the method substantially, and makes it much practical to use. Tunability of the model ensures that for any given synthesis tool, we need to perform the computationally expensive characterization step only once (using a simple target library and a simple synthesis script), and use the resulting model for any combination of target libraries and synthesis scripts with only a minimal effort spent on tuning. In this report, we will demonstrate the tuning method and the results obtained by tuning. Our tuning methodology is very similar to the one used in [2] for RTL power models of soft macros.

2. Overview of the Area Model

In [1], we have proposed a high-level area complexity measure, and a model that uses this complexity measure to get the gate count requirements of combinational VLSI circuits. The area complexity model was of the form

$$C(B) = n \cdot \overline{f_{in}} \cdot \overline{f_{out}} \tag{1}$$

where $C(B)$ is the area complexity measure extracted from Boolean network B, n is the number of nodes in B, $\overline{f_{in}}$ is the average fan-in (in-degree), and $\overline{f_{out}}$ is the average fan-out (out-degree) of the nodes of B.

In [1], we have also introduced a simple model that can be used to estimate the gate count of a circuit based on the complexity measure introduced by (1). The model is of the form

$$G = m \cdot C(B)^n \tag{2}$$

where G is the estimated gate count of the circuit, $C(B)$ is our area complexity measure, m and n are the model parameters that can be obtained by regression analysis. These parameters model the effect of the synthesis tool, the script, target library, and delay specifications on the gate count requirements. This method is very fast once the model parameters m and n are computed for a given *design environment* (to be defined in Section 3).

3. Definitions

- *Technology:* The target gate library that the circuit is mapped on after optimizations.
- *Synthesis script:* The settings of the synthesis tools used to optimize the circuit.
- *Delay point:* The target synthesis point on the delay/area trade-off curve.
- *Design environment:* A specific synthesis script, a specific, and a specific gate library.

- *DC:* Synopsys Design Compiler Suite
- *Area:* Gate count requirement of the optimized circuit

4. Tuning Methodology

The most computationally expensive step in the area estimation methodology introduced in [1], is the up-front characterization of the design environment. Although it is a task which is performed only once for a given design environment, it has to be repeated when a component of the design environment changes. It would be much more practical if it were possible to build the model only once, and somehow "tune" it when the design environment changes, instead of building it from scratch.

In [2, 3], authors show that the relative change in power consumption resulting from a change in technology and/or synthesis script is almost benchmark-independent. As a first step in our tuning approach, we have tried to verify this observation for the areas of the benchmark circuits. To do this, we have mapped a set of MCNC benchmark circuits [4] in different design environments. While changing the design environment, we have kept the synthesis tool unchanged (DC). Once we got the area for all the benchmark circuits in different design environments, we tried to find out if there is any correlation between them.

We began our trials by changing the technology. We have used 6 different target libraries to map all the benchmark circuits using the default synthesis script of DC at the minimum area point. The results are shown in the correlation plots Figs. 4.1–4.5. As you can see from these plots, the relative change in area caused by a change in technology is indeed almost benchmark independent. That means, we can use a constant scale factor to get the area in one technology based on the area in a different one. This is very promising, since it means we will not have to characterize the changes in design environment resulting from technology changes fully if we can approximate the area model with a linear one.

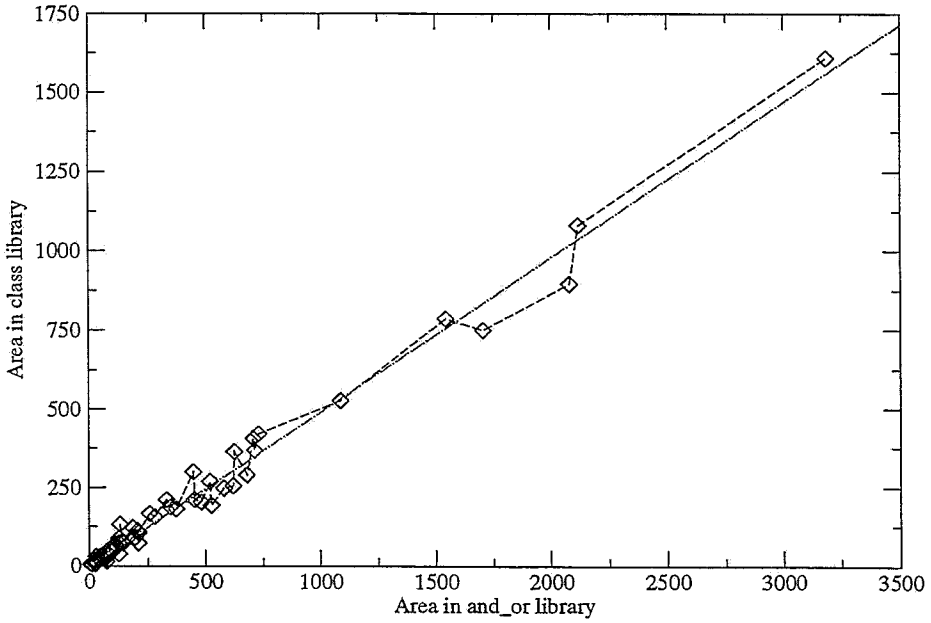

Figure 4.1 Gate Count in Class lib. vs Gate Count in and_or lib.

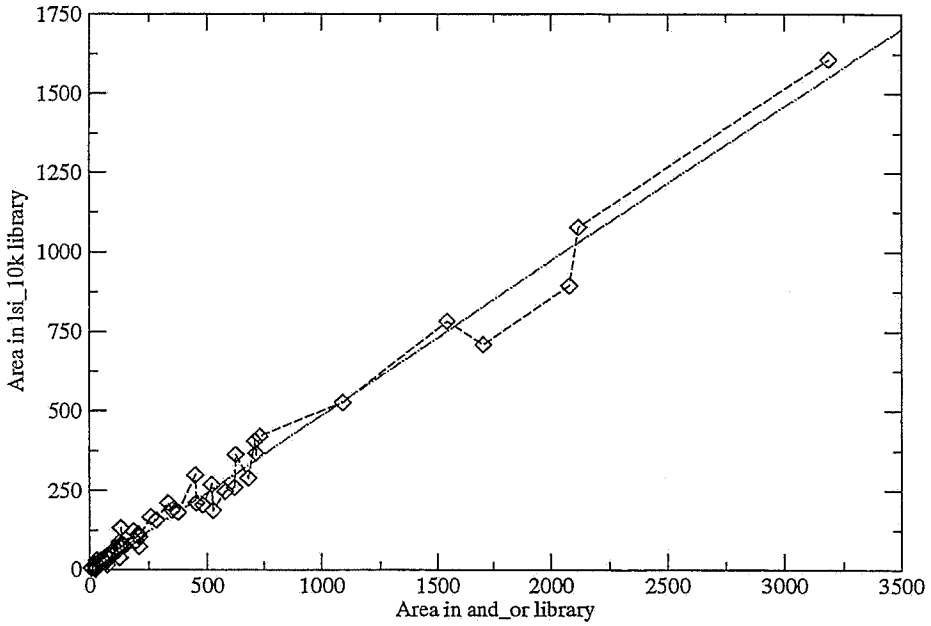

Figure 4.2 Gate Count in lsi_10k lib. vs Gate Count in and_or lib.

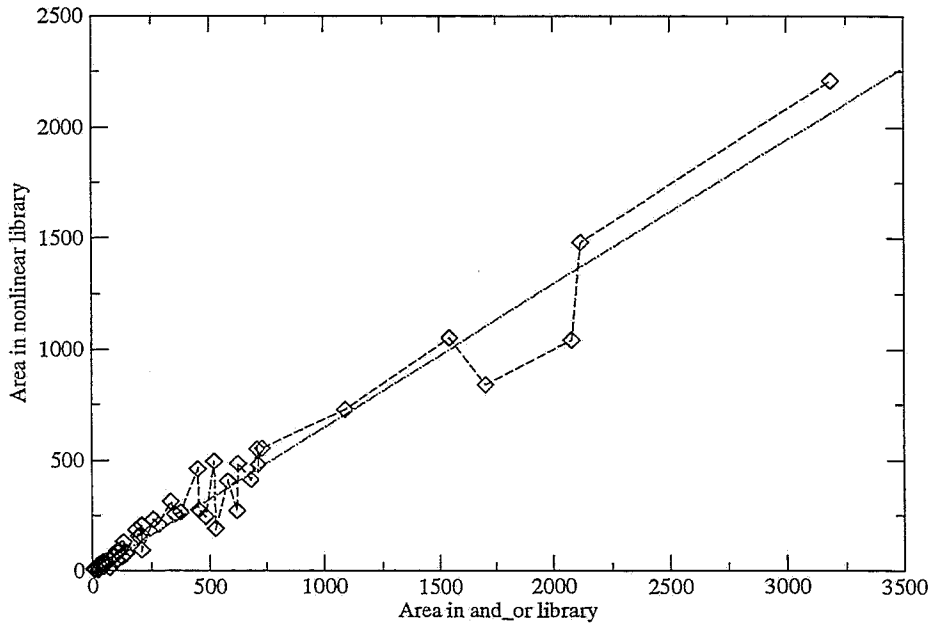

Figure 4.3 Gate Count in nonlinear lib. vs Gate Count in and_or lib.

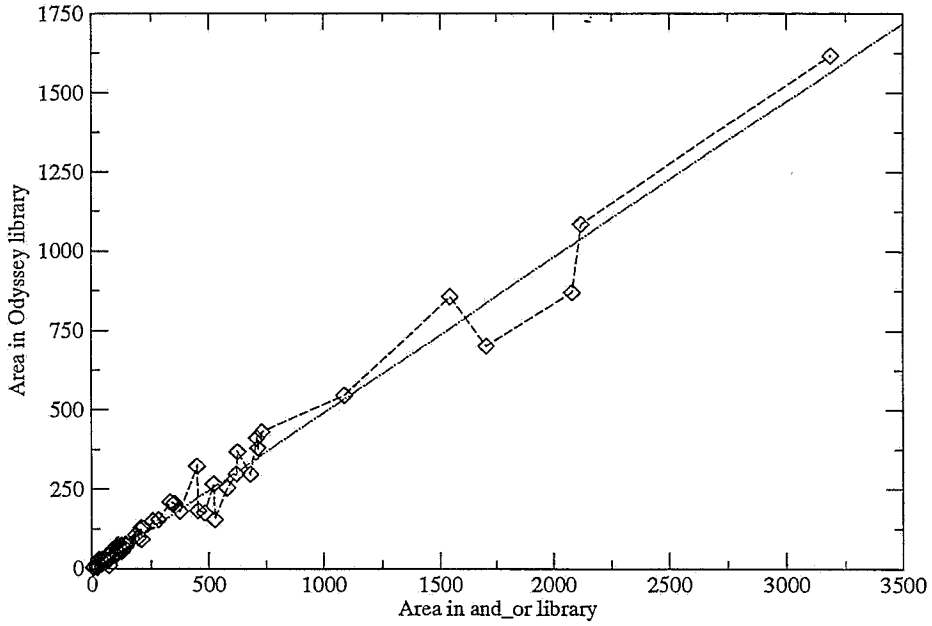

Figure 4.4 Gate Count in Odyssey lib. vs Gate Count in and_or lib.

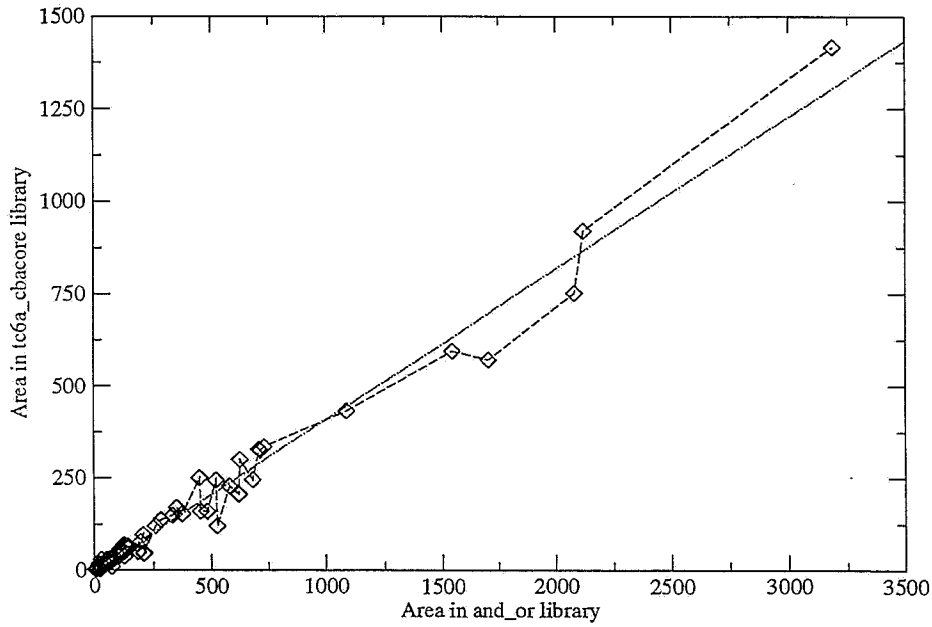

Figure 4.5 Gate Count in tc6a_cbacore lib. vs Gate Count in and_or lib.

After verifying that the change in area resulting from change in technology is almost benchmark independent, we went ahead and tried to see the case where the technology remains constant but the synthesis script changes. To do that, we have synthesized the benchmark circuits again, this time using more aggressive optimization settings (Boolean structuring, and high mapping effort in DC). Then, we have investigated the correlation

between the area obtained this way and the area obtained using default optimization settings (timing based structuring, medium mapping effort) for different technologies. The results obtained in this step can be seen in Figs. 4.6–4.11.

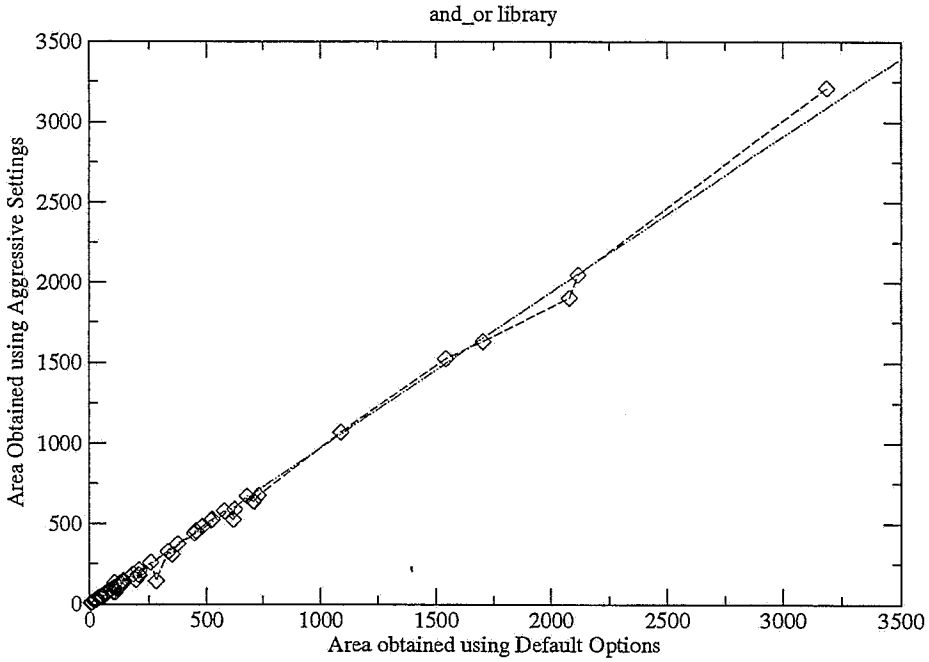

Figure 4.6 Gate Count in and_or lib. for different synthesis scripts.

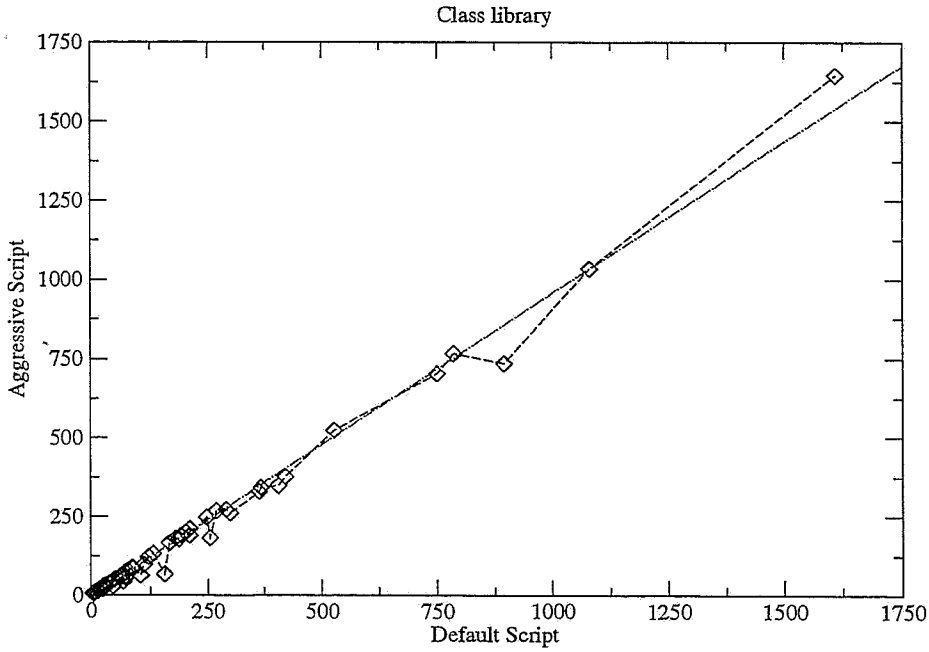

Figure 4.7 Gate Count in Class lib. for different synthesis scripts.

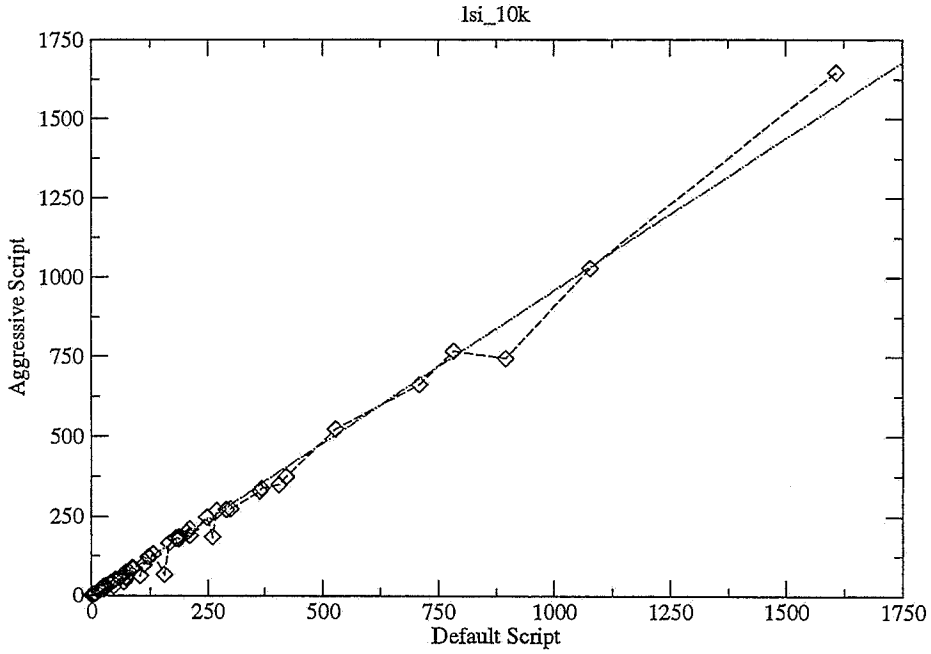

Figure 4.8 Gate Count in lsi_10k lib. for different synthesis scripts.

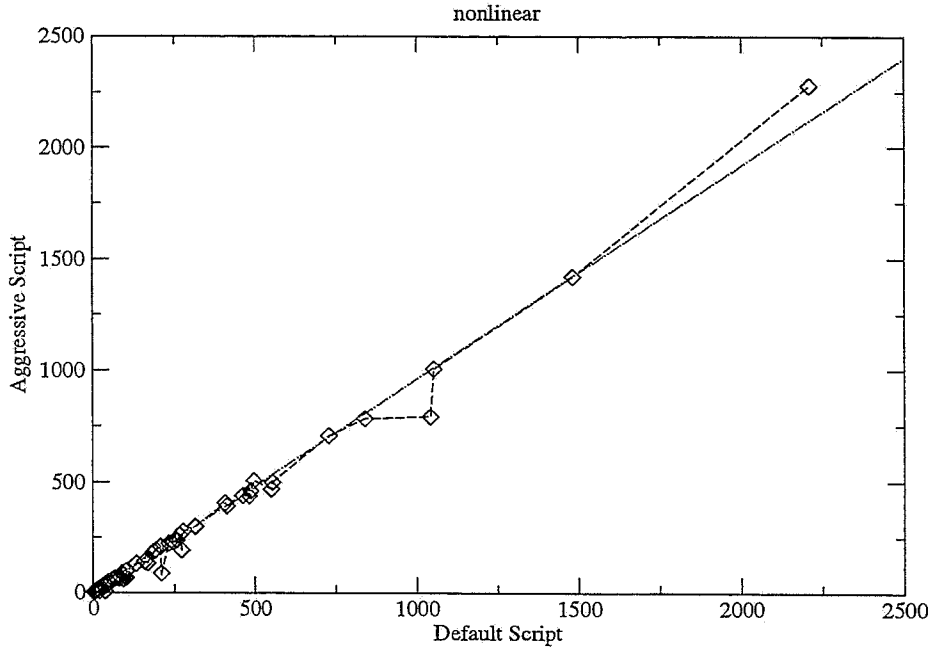

Figure 4.9 Gate Count in nonlinear lib. for different synthesis scripts.

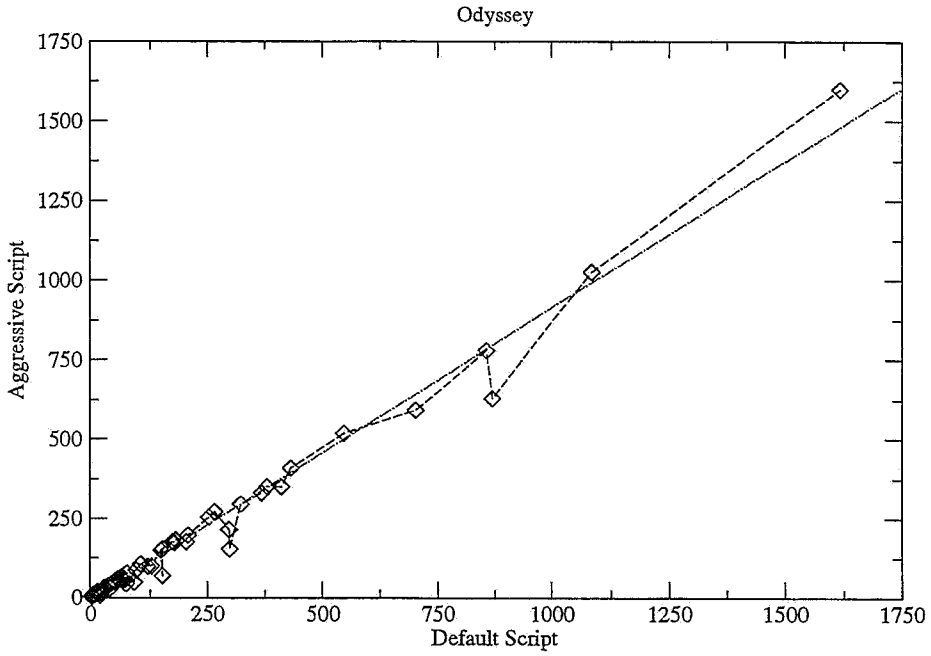

Figure 4.10 Gate Count in Odyssey lib. for different synthesis scripts.

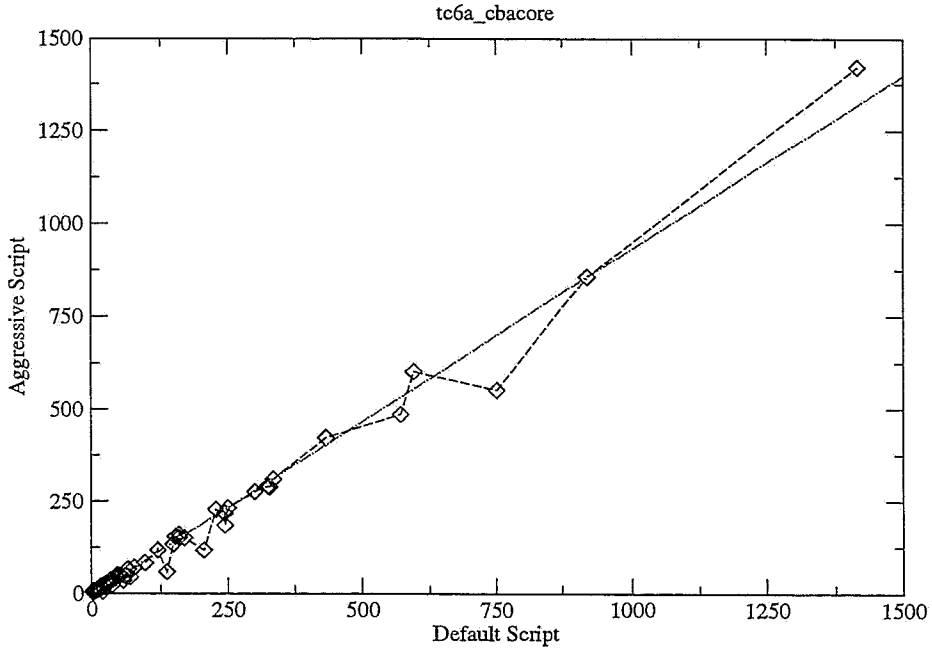

Figure 4.11 Gate Count in tc6a_cbacore lib. for different synthesis scripts.

Once again, it can be seen that the relative change in area resulting from a change in synthesis script is almost benchmark independent. This result, combined with the result obtained above about technology changes shows that we can handle and change in technology and/or synthesis script by just tuning if we can find a linear area complexity model.

Another thing that can change in the design environment is the delay specifications of the circuit. That can actually be treated simply as a change in synthesis script. Still, it will be nice if we can verify that relative change in area for different delay points is benchmark independent. To do this, we have fixed the technology and the optimization settings (except for the delay specifications), and synthesized the benchmark circuits at various delay points. We have labeled these delay points with numbers L=0–100 according to their relative position between the minimum delay and minimum area points. Hence, L=0 point is the minimum delay point, L=100 is the minimum area (maximum delay) point, L=50 is the halfway point between the minimum and maximum delay points, and so on. The results can be seen in Figs. 4.12 and 4.13. Here, again, the correlation is linear, suggesting that we can tune the model for changes in delay point too.

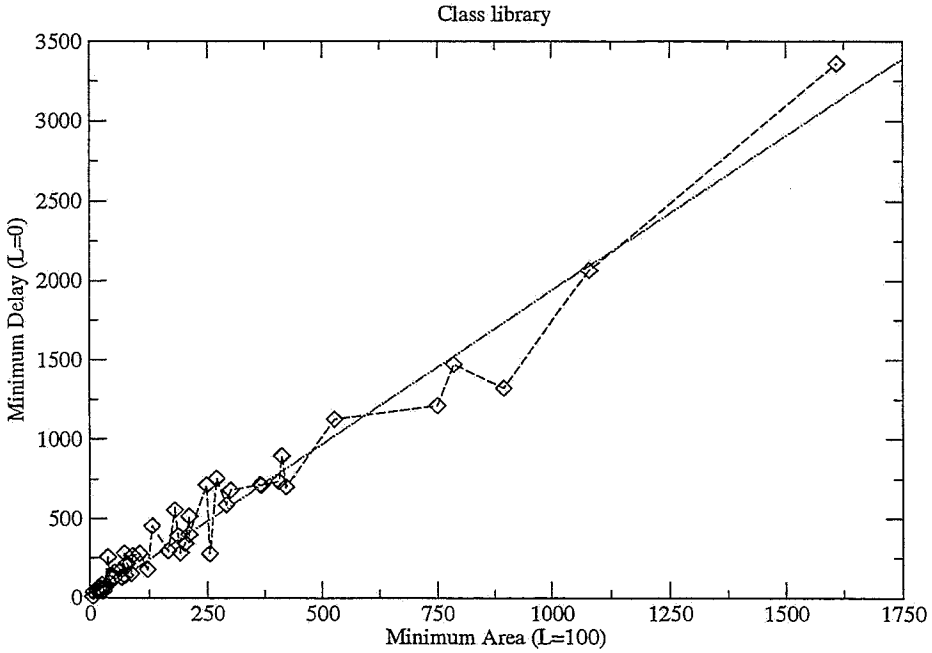

Figure 4.12 Gate Count in Class lib for different delay points.

Going through this process, we have shown that a linear scale factor can be used to get an idea about the area of a circuit implemented in a certain design environment based on its area in a different design environment. In our experiments with the model, we have also observed that the parameter n in (2) is very insensitive to the changes in the design environment. This is a very important observation, as it allows us to fix n for a given synthesis tool based on a full characterization step with some design environment, and use a linear model with a modified complexity measure for other design environments using the same synthesis tool. Suppose we have observed that the value of n for a synthesis tool is N. Then, we can rewrite (2) as

$$G = m \cdot C_N(B) \tag{3}$$

where $C_N(B) = C(B)^N$ is the "modified complexity measure." This model has the advantage of being linear. That means, we can tune this model for any change in design environment

by just computing the scale factor of the change and multiplying the parameter m with this factor.

This introduces a problem though. How do we compute the scale factor of a design environment change? Theoretically, since the changes are always linear, it is sufficient to choose one benchmark circuit, implement it in different environments, and use the ratio of areas obtained this way as the scale factor. In practice, however, this can introduce huge errors if we happen to choose a very "bad" benchmark to do this ("bad" meaning lying relatively far from the straight line). One simple solution to this problem is to use multiple benchmark circuits in this process. In our experiments, we were able to obtain good accuracy using only 3 benchmarks to compute the scale factor.

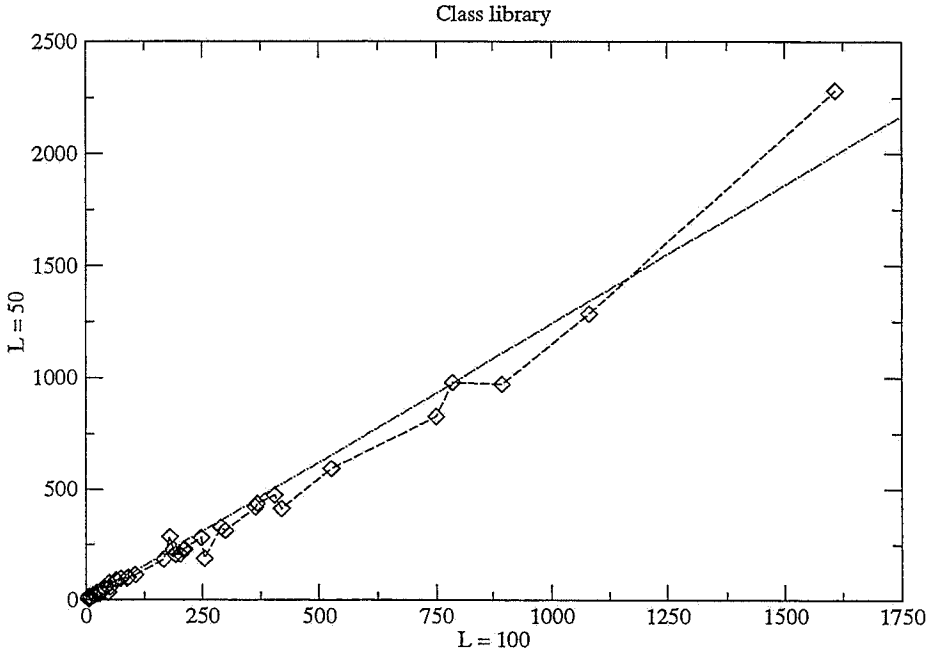

Figure 4.13 Gate Count in Class lib for different delay points.

5. Experimental Results

To test our tuning method, we have chosen a "base" design environment and fully characterized the model ((2)) for this environment. Then, we have used this model and the tuning method described in Section 4 to estimate the area of the benchmarks in different design environments.

As our "base" environment, we have chosen the *and_or* library, the default synthesis script of DC, and the minimum area point (*set_max_area = 0*). Fig. 5.1 shows the characterization of the model in this environment. The model parameters for this particular case is $m = 1.01246$ and $n = 0.809293$.

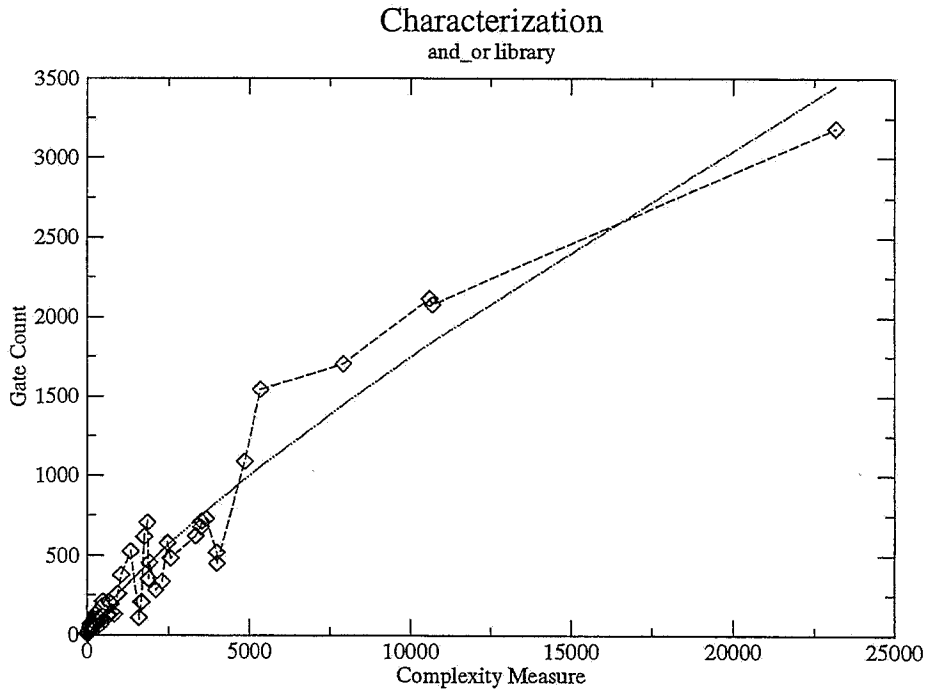

Figure 5.1 Characterization of the model.

After the full characterization step, we have fixed the value of n to be 0.81, and tuned the model for different environments using scale factors computed in the way described above. The estimation results for different environments can be seen in Figs. 5.2–5.5.

As can be seen from the figures, the tuning method gives quite acceptable results without going through the expensive characterization phase again and again for ever change in the design environment.

6. Conclusions

We have presented a single scale tuning method for the area complexity model we have presented previously.

References

[1] K. M. Buyuksahin and F. N. Najm, "High-level area estimation." Submitted to ISLPED 2002, Feb. 2002.

[2] A. Bogliolo, R. Corgnati, E. Macii, and M. Poncino, "Parametrized RTL power models for soft macros," *IEEE Transactions on VLSI Systems*, vol. 9, pp. 880–887, Dec. 2001.

[3] A. Bogliolo and L. Benini, "Node sampling: a robust RTL power modeling approach," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, pp. 461–467, 1998.

[4] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide - Version 3.0.* Microelectronics Center of North Carolina (MCNC), P. O. Box 12889, Reseach Triangle Park, NC 27709, Jan. 1995.

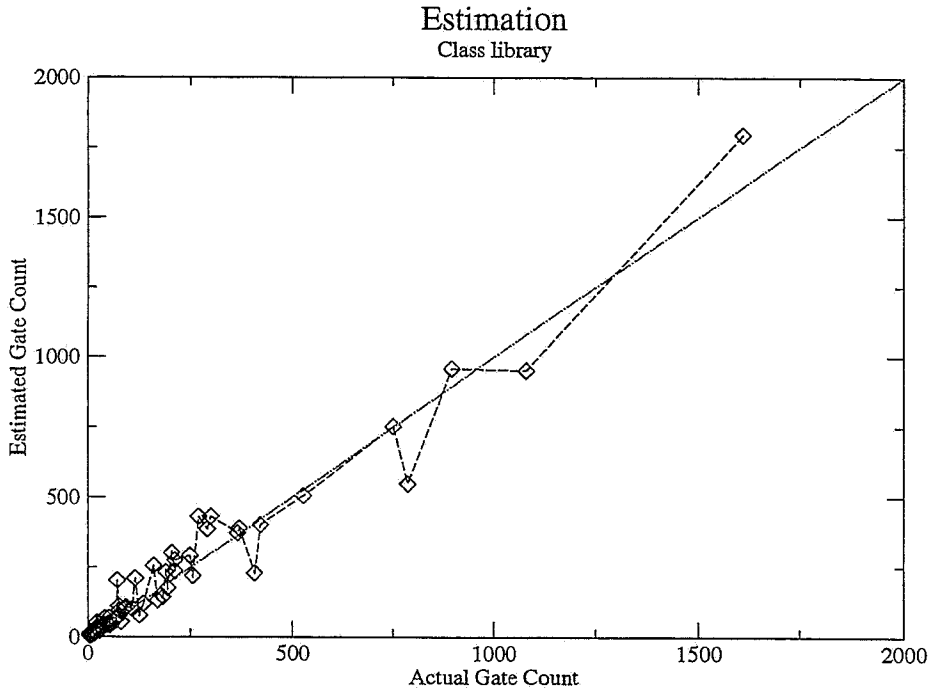

Figure 5.2 Estimated vs. Actual Gate Count for Class library with default script at minimum area point.

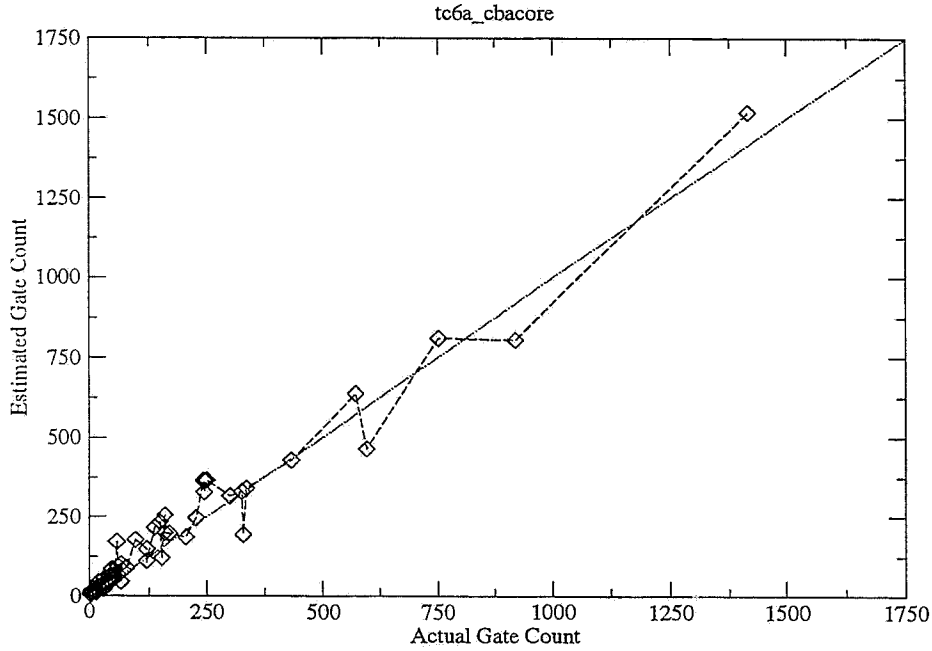

Figure 5.3 Estimated vs. Actual Gate Count for tc6a_cbacore library with default script at minimum area point.

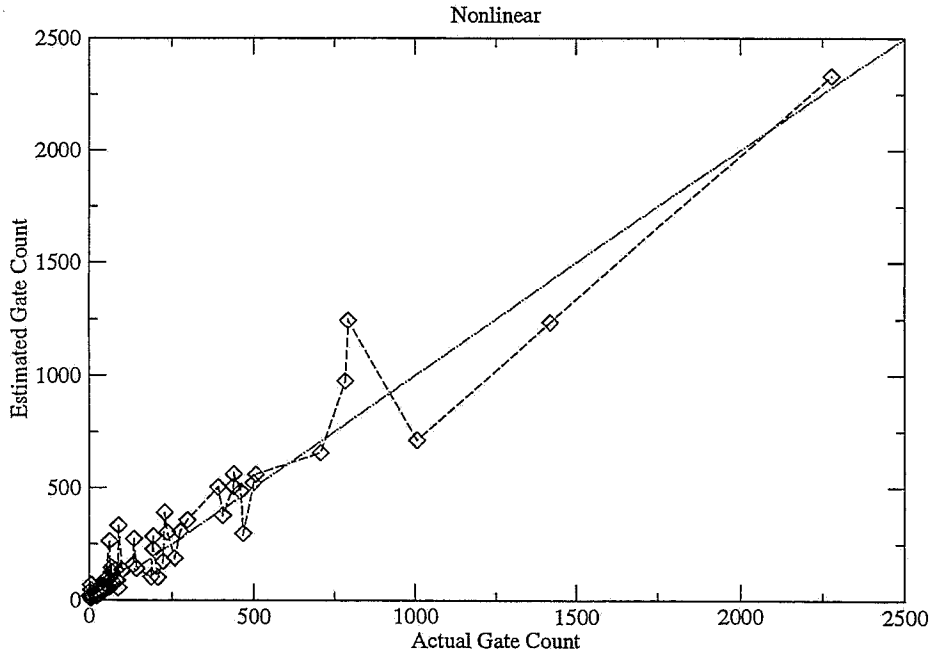

Figure 5.4 Estimated vs. Actual Gate Count for nonlinear library with the aggressive script at minimum area point.

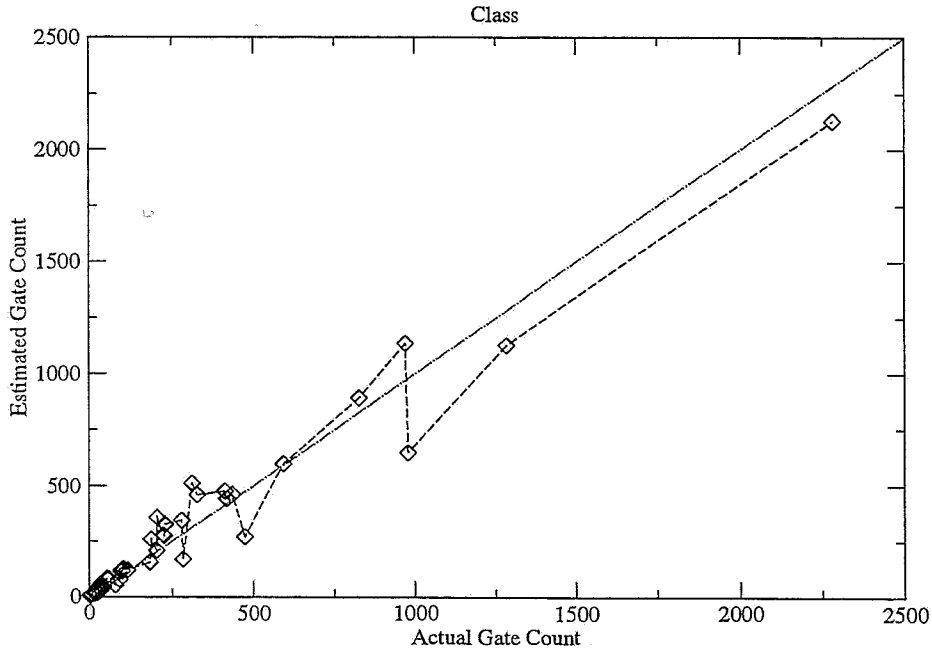

Figure 5.5 Estimated vs. Actual Gate Count for class library with the default at L=50 point.