UC Irvine
ICS Technical Reports

Title
A fast area-delay estimation technique for RTL component generators

Permalink
https://escholarship.org/uc/item/5b92c1n4

Authors
Jha, Pradip K.
Dutt, Nikil D.

Publication Date
1992-04-10

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Notice: This Material
may be protected
by Copyright Law
(Title 17 U.S.C.)

ARCHIVES
Z
699
C3
no. 92-33
c.2

A Fast Area-Delay Estimation Technique for RTL Component Generators

Pradip K. Jha and Nikil D. Dutt

Technical Report #92-33
April 10, 1992

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717
(714) 856-8059

Notice: This material
may be protected
by copyright law
(Title 17 U.S.C.)

Abstract

An important benefit of high-level synthesis is rapid design space exploration through examination of different design alternatives. However, such design space exploration is not feasible without fast and accurate area and delay estimates of the synthesized designs. These estimates must factor in physical design effects and technology-specific information in order to achieve accuracy. High-level synthesis tools often use abstract, parameterized component generators for describing the synthesized RT design, and thus need to be supported by fast and accurate estimators for these parameterized RT-components. Ideally, we would like to obtain the actual area and delay attributes of each component by constructing (or generating) the designs. However, such constructive methods require excessive run times, prohibiting on-line integration with the tasks of scheduling and allocation. In this paper, we describe a fast (on-line) method for estimating the area and delay of regular-structured generic RT components that are tuned to a particular technology library. The estimation models are generated using a least-square approximation on a set of sample data points from selected component implementations. We performed an extensive set of experiments to validate our estimation technique on combinational as well as sequential RT component generators. The results show a prediction of the area and delay to within 10% of the actual values. These models have also been integrated with a high-level synthesis system to permit on-line estimation of a component's area and delay.

Contents

List of Figures

List of Tables

1 Introduction

Behavioral or High-Level Synthesis (HLS) maps the behavior of a design to a RT structure and a controller that execute the input behavior under user specified design constraints. A major task in HLS is design space exploration in terms of selecting and allocating a proper set of RT components. This task of design space exploration is guided by metrics such as the area and delay of the components. Extensive design space exploration and trade-off analysis between different design alternatives can only be achieved if fast and accurate area and delay estimators exist for evaluating selected components; such estimators are crucial to the success and acceptance of high-level synthesis as a design methodology [GDWL92] [MiLD92].

High-Level Synthesis typically relies on a library of well-defined, parameterized RT component generators to simplify the mapping of behavioral variables and operators to physical components. These parameterized components are used as the building blocks for the tasks of allocation and binding. Each component is customized by parameterized attributes such as the required bit-width and functionality. Such a library of RT component generators provides a complete component set for HLS [Dutt88], and provides a path to physical design through logic and layout synthesis [Dutt91].

Since we need accurate estimates for effectively supporting the tasks of component selection and binding, we would ideally like to store the area-delay attributes for all the component instantiations so as to provide accurate figures for component selection and binding. However, the storage of all components is infeasible since there is virtually an infinite number of possible components that can be generated by varying the parameter attributes. For example, consider the class of ALU component generators that can perform any subset of 26 arithmetic, logic and comparison functions. There are 2^{26} possible combinations of the functions alone. The bit-widths of the two inputs to the ALU can also be parameterized. Furthermore, each ALU component (with a fixed set of parameters) can have multiple implementations (e.g., ripple-carry, carry-lookahead). If we assume that the ALU's width varies from 1 to 128, and that each ALU component has two possible implementations, we get $2^{26} * 128 * 2 = 1.72 * 10^{10}$ area-delay values for the class of ALU generators alone! Although it may be unrealistic to assume that all possible values of the parameters can be attained, this number is indicative of the vast design space populated by different component instantiations.

An obvious alternative to storing all the designs and their metrics is a constructive approach: we can generate the component implementations on demand, and return the actual area-delay metrics of the constructed design. For example, DTAS[Kipp91] is a system that constructs component designs by decomposing a generic RT component into primitive cells and performing a mapping to a given RTL technology library [DuKi91]. However, such constructive tools have to search a large space of candidate designs and exhibit relatively long run-times (on the order of few-to-several minutes), and hence cannot be directly integrated on-line with HLS tools for the task of rapid design space exploration.

We propose a practical solution to this problem by developing a set of estimation functions that generate these area-delay metrics on-line. These estimation functions accept generator parameters such as *input-width* and *list-of-functions* and return area-delay metrics for the component specified by the parameters.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 defines the problem of estimating the area and delay of generic component generators, given a component's

parameters and a few technology-specific design data points. Section 4 describes the estimation models we have developed. Section 5 describes the experiments performed to validate our models. We show that our models are simple, fast and provide fairly accurate results (within 10% of actual values). Furthermore, the estimators are integrated with an existing HLS system[LiGa88] and run on-line during the tasks of component selection and allocation. Section 6 concludes with a summary.

2 Related Work

The problem of area and delay estimation has been studied at several design levels and in several contexts. At the level of a complete datapath design, work has been done to predict the area-time tradeoffs for a datapath given the area and delay values for the primitive modules used to construct the structural design[JaMl88]. Component (module) selection techniques[Jain90] have been proposed to perform area-time analysis for a datapath, given a library of RT components with area and delay characteristics.

At the logic design level, work has been done to predict the area and delay of an RT component, given its structural implementation as a netlist of logic cells or blocks [JOLS92]. Early approaches such as PLEST[KuPa89] analytically estimated the area of a component represented as a netlist of cells, and returned the area as a sum of the component's functional and estimated wiring areas. LAST[ChKu91] and TELE[ChKu92] use a combination of analytical and constructive techniques to predict the area and delay for a netlist of cells, while accounting for wiring. Recent work in layout-based estimators[WCGa91] [CWGa91] also provide area and delay values, given the structure of the modules (components); these approaches take into account technology factors such as layout architecture and wiring.

Purely constructive approaches can be used to estimate the area-delay characteristics of RT components. One constructive method generates Boolean equations for the RT component and represents these in a canonical form to provide area-delay estimates of the component[BRSW87]. Another constructive method builds a tree of possible component implementations by decomposing the component into an interconnection of primitive logic blocks. This large design space can then be pruned and searched for candidate designs. Tyagi[Tyag90] uses an algebraic model for representing the design space and a set of area-delay-power functions to prune design search space. DTAS[Kipp91] also provides functional area and delay values for a component by pruning the design space with a performance filtering function. Although these constructive techniques yield good estimates, they are too slow for direct integration with high-level synthesis tools that require real-time (on-line) estimators.

[WaCh90] proposes a simple technology-independent model for predicting the delay of combinational control (random) logic, given the Boolean equations for the logic (it does not deal with area estimation). Their technique assumes a certain structuring of logic. Consequently, it cannot handle multiple design implementations – a common occurrence for RT datapath components. Although our approach is similar to [WaCh90], we provide both area and delay estimates at a level higher than logic equations.

[Brew88] presents a few simple estimation functions for datapath components, derived from a general structure of the component's implementation and technology-specific values from a data-book. However, these estimation models have not been tested against actual designs, and do not account for parameterized multi-function components, or for a range of component implementa-

tions.

In summary, previous approaches have dealt with estimation either at a higher level (i.e., a complete RT datapath design) or at lower level (i.e., logic equations or structural implementation of a particular component), but have not addressed the problem of rapid estimation of parameterized RT components. However, HLS tools can be effective only if they are supported by fast (on-line) estimates of area-delay metrics for RT-components to support component selection, scheduling and allocation decisions, as well as design tradeoffs between different component implementations. Furthermore, at the behavioral level, it is convenient to specify an RT-component using a set of parameters for the component's functionality, bit width and other attributes. This convenience comes at cost: estimation becomes difficult, since the possible space of design implementations is infinitely large. To alleviate this problem, we propose a technique to rapidly estimate the area-delay metrics for such parameterized RT-datapath components used in HLS.

3 Problem Definition

We define the area-delay estimation problem in terms of a set of generic component generators, their possible parameters and the estimation functions.

Let G be the set of component generators, P be the set of parameter names and D be the set of domains for the parameters.

- $G = \{G_i | G_i$ is a RT component generator.$\}$
- $P = \{P_i | P_i$ is a parameter.$\}$
- $D = \{D_i | D_i$ is a domain for parameter P_i.$\}$

An ordered set of parameters PG_i is associated with each generator G_i.

- $PG_i = \{P_{i1}, P_{i2}, ..., P_{in}\}$ such that P_{ik} is the k_{th} parameter of generator G_i and there are n parameters associated with G_i.

Each component generator G_i is a function that maps its parameter values PG_i to the instantiation of a specific component. That is, the set of components covered by a generator G_i is obtained by applying the functions G_i on the cross product of all the domains associated with the parameters belonging to G_i.

- $C_i = G_i(D_{i1} \times D_{i2} ... \times D_{in})$ and D_{ik} is the domain associated with k_{th} member of PG_i.

Consider the ALU generator, i.e., let G_i = ALU. This generator has three parameters: input-width, num-functions and function-list, i.e.,

- PG_i = (input-width, num-functions, function-list)

The set of components C_i covered by the ALU generator is given by the cross product of the domains of these parameters. For example, a 4-bit ALU component that can perform two functions: (ADD, SUB) is a member of C_i. This component is specified as ALU(4, 2, (ADD, SUB)).

Each component C_{ij} may have several alternative hardware implementations. Let S_{ij} be the set of implementations for the component C_{ij}.

- $S_{ij} = \{S_{ijk} | S_{ijk}$ is the k_{th} implementation of component $C_{ij}.\}$

Each implementation S_{ijk} has two metrics, area A_{ijk} and delay D_{ijk} associated with it. Thus we have a set of areas and delays corresponding to different implementations of a component.

We develop area-delay models for each component generator G_i based on the area and delay for a subset of components C_{ij} covered by G_i. Let A_{ij} be the set of areas and D_{ij} be the set of delays for the subset of components under consideration. Based on the members of A_{ij}, we propose the area model FA_{ij}. Similarly, based on the members of D_{ij}, we propose the delay model FD_{ij}. These area-delay models FA_{ij} and FD_{ij} predict metrics for the components C_{ik} that are not members of C_{ij} but that are derived from the generator G_i.

Two distinct components C_{ij} and C_{ik} that are derived from the same generator G_i, vary in terms of their values of parameters PG_{ij} and PG_{ik}, respectively. The proposed models should be able to predict the area and delay across a range of values for these parameters. Furthermore, each component C_{ij} has a set of possible implementations S_{ij}, with multiple area (A_{ij}) and delay (D_{ij}) values. For example, an ADDER component has multiple implementations and multiple area-delay values. Hence, we need a set of area-delay models FA_i and FDi to handle multiple S_{ij} and multiple metrics (A_{ij} and D_{ij} for a component C_{ij}). It is obvious that the models FA_{ij} and FD_{ij} should be some function of the parameters PG_i specifying the component C_{ij}:

- $FA_{ij} = a_{ij0} + \sum_k (a_{ijk} \star fA_{ijk}(PG_{ijk}))$
- $FD_{ij} = d_{ij0} + \sum_k (d_{ijk} \star fD_{ijk}(PG_{ijk}))$

The problem thus translates into two steps: formulation of the functions FA_{ij} and FD_{ij}, and determination of the constants and coefficients (a's and d's). In this paper, we briefly describe these steps and provide experimental results to support their validity.

For the rest of the paper, we use the terms RT *component* and RT *module* interchangeably. We also use the term *metrics* to refer to the area and delay values for a component, and *estimates* to denote the estimated values for the area and delay metrics.

4 Our Approach

There are two ways by which area-delay metrics for RT components can be provided to a high-level synthesis (HLS) system. The first method precomputes these metrics for a set of component implementations and stores them in a component database. The second method uses estimation models to calculate the metrics online. While the first method provides very accurate metrics at the cost of long run-times and a huge component database (we have virtually an infinite number of possible components), the second method compromises accuracy for run-time.

We propose a combination of these two methods. For some components, we store area-delay metrics of actual design implementations in the component database. For other components, we use models to generate the metrics. In this section, we describe a method to model the area-delay

metrics and provide some insights on how to select components whose metrics are to be stored into component database.

4.1 Area-Delay Models

Our estimation technique uses a sample space of design points on which we perform a least-square regression fit of the formulated functions FA_{ij} and FD_{ij}. We believe that this is a useful approach since designers often store the attributes of commonly occurring designs (e.g., 4-, 8- and 16-bit adders). Since a regression analysis using least-square approximation may not capture the intricacies of certain component implementations, we have to pay attention to the appropriate selection of the sample data points. The following steps summarize our approach:

Step 1 *Generate some real design structures for a component and obtain sample data points for area and delay.*

Step 2 *Study the structure of the design generated, and the variations of the area-delay metrics with respect to the parameters that define the component.*

Step 3 *Formulate functions for estimating the area and delay of the generator.*

Step 4 *Run least square approximation to calculate the constants (a_{ij0} and d_{ij0}) and the coefficients of various terms (a_{ijk} and d_{ijk}) used for the functions modeling the metrics.*

Step 5 *Test the estimation model.*

We test the accuracy of the results against a user-specified error bound. If our models do not satisfy the user-specified error bounds, we go through an iterative experimentation phase, where we repeat some of the Steps 1-5 above. We note that the error bound is often satisfied by the simple addition of linear and log factors to the estimation functions, as suggested by the design's structure. In an extreme case when the error bound is not satisfied, we may have to generate more implementation data points and repeat all of the steps to obtain new coefficients. However, in our experiments, we have observed convergence within one or two iterations for an error bound of 10%.

4.2 Formulation of Estimation Models

Recall that the area-delay estimation models FA and FD for a generator G are functions of its parameters PG. One significant parameter for a generator is the component's bit-width, which clearly has a significant impact on the estimation models. Tyagi [Tyag90] has developed an information-theoretic model to understand the relationship between a module's parameters (e.g., bit-width) and its performance metrics. Based on the communication between n-bit slices of of a component, he classifies some combinational components into categories and provides a formulation of models for each category. These models describe the asymptotic behavior of area and delay with respect to major parameters of the generators. For example, both the area and delay of a ripple-carry adder vary linearly with respect to the bit_width of the inputs.

We use a similar formulation for the primary factors of the area-delay models that account for the major contribution towards the performance metrics of a component. In addition to these

primary factors, we add some secondary factors that are decided by some rules of thumb and by the structure of the components.

For example, consider the class of Multiplexer generators that has two parameters: num-inputs(ni) and input-width(iw). The area per bit of a multiplexer grows linearly with ni. This constitutes the primary factor in the area model of multiplexer. A constant and a logarithmic term with ni form the secondary factors. The delay of a multiplexer has logarithmic behavior with respect to ni (primary factor), and we add a constant and a linear term as secondary factors. Since we know that the area of multiplexer is directly proportional to iw and that its delay is independent of iw, we get the following area-delay models for a parameterized multiplexer:

$$Area = iw * (a_1 + a_2 \star log_2\lceil ni \rceil + a_3 \star ni)$$

$$Delay = d_1 + d_2 \star log_2\lceil ni \rceil + d_3 \star ni$$

It is interesting to note that at a first glance, certain components may not seem to have a variation with respect to the component's bit-width (e.g., bit-wise AND). However, when the components are laid out, we find that the routing effects begin to appear for larger bit-widths. We therefore need to add a function of the bit-width for these components also.

4.3 Estimation Model Generation

We use a least square approximation method on a set of sample design implementations, to determine the coefficients of the formulated estimation equations. We therefore need to address the following questions: (1) Which of the infinitely many possible components for a generator should be considered for calculating the coefficients? (2) For each selected component, how do we actually obtain the actual area and delay values?

The set of components that are to be used for calculating the coefficients should be fairly representative of all the possible components for a generator. We attempt to select the set of components so as to avoid a bias towards any particular parameter or any particular subset of values for a parameter. For example, several components that are implemented as tree-based logic structures show distinct behaviors for *num_inputs* that are powers of two. Consequently, we often use the following values for the parameter *num_inputs* : $(2, 3, 4, 5, 6, 7, 8, 12, 15, 16, 32, 48, 64)$.

The real area-delay values can be obtained by physically laying out each component and measuring the actual area and delay. Although this provides very accurate measures, it is a very time consuming process. Since our approach needs a substantial number of a component's area-delay values (on the order of 15), we use existing tools to provide these metrics.

DTAS [Kipp91] is one such tool that provides performance metrics. DTAS maps RT-level design components from the GENUS generic component library [Dutt88] to technology-specific library cells and macros. Given the specification of a component, DTAS generates a set of alternative designs corresponding to different design decompositions using the primitive building blocks. The area provided by the DTAS is the sum of the functional areas of the building blocks used in the design implementation. DTAS computes the delay values for all critical paths through a design implementation (pin-to-pin, rising and falling). This calculation takes into account the fan-out of the designs. Thus, DTAS provides a good source of sample design points for our approximations, using functional area and delay values.

To incorporate wiring effects, we use LAST[ChKu91] and TELE[ChKu92], which are estimators that provide more accurate metrics by considering not only functional blocks but also the wiring contributions. Since LAST and TELE have been benchmarked against actual layouts produced by commercial tools as well as against custom-designed layouts, they provide a good source of sample design points for area-delay values that include routing information.

4.4 Selection and management of components in the database

As mentioned before, it is infeasible to store attributes for all possible components, since there are virtually infinite number of possible component implementations. We need to select a subset of components that fits in our space-limited component database and yet enhances the overall performance by providing accurate metrics for some components. This database should include data for components that are frequently used and components for which estimation models do not perform well.

We propose the following guidelines to choose the subset of components to be stored in the database.

- Store metrics for components that are characterized by parameters with obvious values. For example, very often designs use components that have bit-width equal to some power of two (e.g. 8, 16, 64).
- Store metrics for components that are frequently used. The component database can record the tally of various component references in the past. Components with high tally should have its attributes stored.
- Prior design knowledge of a component's circuitry could provide hints regarding the components that have high probability of being referenced. For example, if the design in consideration is 16-bit microprocessor, chances are high that at least one 16-bit register will be required for the design. Such component attributes should be stored in the database.
- Previously referenced components can provide some insights on the nature of components that will be referenced in the future. If a synthesis tool has queried for a 16-bit ALU, chances are high that it will require a 16-bit register very soon.
- While generating and testing the estimation models, we may discover some "problematic" components whose area-delay values do not fit well with models. These are another set of components whose area-delay values should be stored in the database.
- Also, if the database is running out of space and the attributes of some components need to be deleted, then the least-recently-used components should be compromised. This is based on the memory management principle that the least recently referenced components have the least chance of being referenced in future.

4.5 An Example

We illustrate the derivation of metric models for the Logic unit (LU) generator. A generic LU component can perform any subset of the 16 primitive logic functions. The parameters associated

Function	*AB*	*AB'*	*A'B*	*A'B'*
ZERO	0	0	0	0
ONE	1	1	1	1
AND	1	0	0	0
NAND	0	1	1	1
OR	1	1	1	0
NOR	0	0	0	1
XOR	0	1	1	0
XNOR	1	0	0	1
LID	1	1	0	0
RID	1	0	1	0
LNOT	0	0	1	1
RNOT	0	1	0	1
LINHI	0	0	1	0
RINHI	0	1	0	0
LIMPL	1	1	0	1
RIMPL	1	0	1	1

Table 1: Function table for a bit-slice of a Logic Unit(LU)

with this generator are *input-width* and *set-of-functions*. Table 1 lists the 16 possible functions for a generic LU.

Let us assume that A and B are the two inputs to the LU. Also, let *bw* be the bit-width of the two inputs A and B, *nf* be the number of functions, *fl* be the set of functions and *num1* represent the sum of the number of 1's in Table 1 corresponding to the functions for LU instance under consideration.

We now walk through the method outlined in the previous section using this example.

Step 1 Using DTAS (or any other design generator) we compute the area and the delay for logic units with varying *bw*(bit-width) and *fl*(set of functions). As mentioned before, we choose designs with *bw* that are both powers of two and values that lie between these powers of two. Also, these data points include various combinations of functions, both in terms of *nf* and *fl*.

Step 2 We study the structure of some of the designs generated. Consider the structure of a 2-bit LU component that performs the following functions: ONE, NAND and XOR, (i.e., $bw = 2$, $nf = 3$ and $fl = (ONE, NAND, XOR)$). The top level structure of this LU is shown in Figure 1. This figure shows that the design is composed of two independent modules: FG_12 and FG_13 (one for each bit), with each module sharing the three control lines $S0, S1$ and $S2$. Each of these modules has the structure shown in Figure 2.

We analyze the structure shown in Figure 2 from the input to the output. First, we have five inverters, two for the inputs and three for the control lines. Next, we have nine 3-input AND gates. Each of these AND gates represents a '1' in Table 1 and has a number of inputs equal to *nf*. Since $num1 = 9$ (4 for ONE, 3 for NAND and 2 for XOR), we need nine AND gates. At the next stage, we have four 3-input AND gates, one for each column. The inputs to these AND gates arrive from A and B and some ORed function of result of the nine 3-input AND gates from the previous level. Finally, the output of these four AND gates is ORed to give the result.

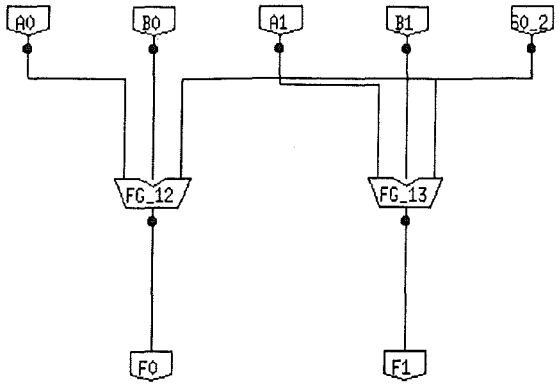

Figure 1: Top level structure of the Logic Unit

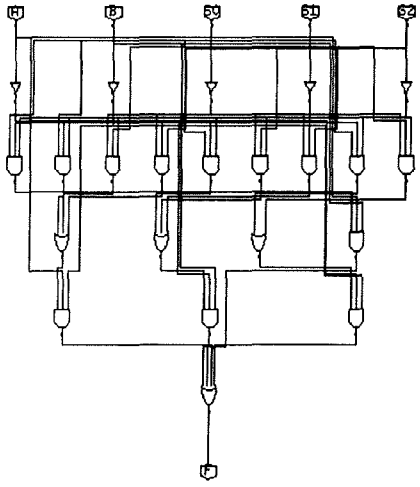

Figure 2: Structure per bit of Logic Unit

Also, we study the variation of area and delay as we change the parameters: *bit-width* and *set-of-functions*. We observe that the area of an n-bit LU is n times the area of a one-bit LU. We also note that the delay of an LU is independent of its bit-width.

Step 3 Combining the study of previous steps, we observe the following:

- Since the area is proportional to bw, we can make area predictions for an LU by multiplying the area for $(bw = 1)$ with bw.
- The AND gates at level 2 account for most of the LU area. This is proportional to *num1*. As observed before, each of these AND gates has number of inputs equal to *nf*. Thus, the area is proportional to the product of *nf* and *num1*.
- The number of inverters is *nf*.
- There are a few other gates that are independent of the LU parameters.
- The critical path goes through an inverter, an AND gate with number of inputs equal to the number of functions, an OR gate, a 3-input AND gate and finally a 4-input OR gate.

Based on these observations, we propose the following area and delay functions for the LU component generator:

$$Area(bw, nf, num1) = bw \star (a_1 + a_2 \star nf + a_3 \star nf \star num1)$$

$$Delay(bw, nf, num1) = d_1 + d_2 \star \ln(nf) + d_3 \star \ln(num1)$$

Step 4 We run least square approximation routines on the area-delay values of sample design points to obtain the coefficients a_{is} and d_{is}.

Step 5 We test our model against some real design points that are not stored in the component database. The initial model we derived from the functions mentioned above yields satisfactory results, and thus we use this model.

5 Models for other components

In this section, we discuss the estimation models developed for some other generic RT component generators using the method described in the previous section.

5.1 With respect to DTAS

5.1.1 Gates

Generic gate components include AND, OR, NAND, NOR, XOR and XNOR logic gates. Each gate generator has two parameters: num-inputs(ni) and input-width(iw). The area of an n-bit gate is n times the area of a 1-bit gate. The gate delay is independent of iw. We develop the following area-delay models:

$$Area = iw * (a_1 + a_2 \star log_2\lceil ni \rceil + a_3 \star ni)$$

$$Delay = d_1 + d_2 \star log_2\lceil ni \rceil + d_3 \star ni$$

5.1.2 Decoder

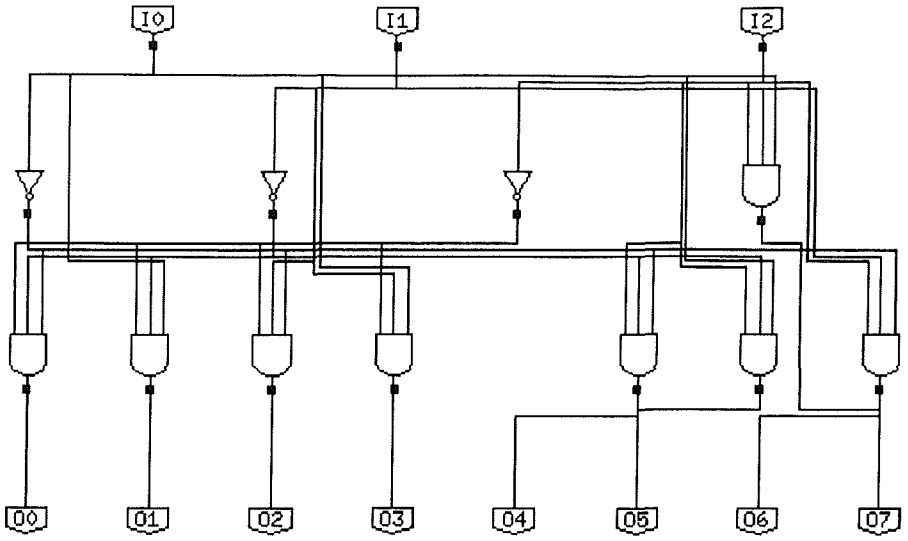

Figure 3: Design of decoder

A decoder is characterized by a single parameter, input-width(iw). The structure of a decoder with $iw = 3$ is shown in Figure 3. I0 through I2 are the inputs and O0 through O7 are the outputs. Each output is fed by an AND gate, and each of these AND gates has number of inputs equal to iw. Number of outputs is given by 2^{iw}. Also, the critical path consists of an inverter and one of the AND gates discussed above. Based on these observations, we develop the following models:

$$Area = a_1 + a_2 \star iw + a_3 \star 2^{iw} \star log_2\lceil iw \rceil$$

$$Delay = d_1 + d_2 \star iw + d_3 \star 2^{iw}$$

5.1.3 Multiplexer

A multiplexer has two parameters, num-inputs(ni) and input-width(iw). The structure of a multiplexer with four inputs($ni = 4$), each input being two-bit wide($iw = 2$), is shown in figure 4. I0 and I1 constitutes the first input, I2 and I3 the second input, I4 and I5 the third input and I6 and I7 the last input. S0 through S3 are the four control lines. Similar to the design of LU discussed in section 4.5, the structure of a multiplexer is composed of two independent modules : MUX_12 and

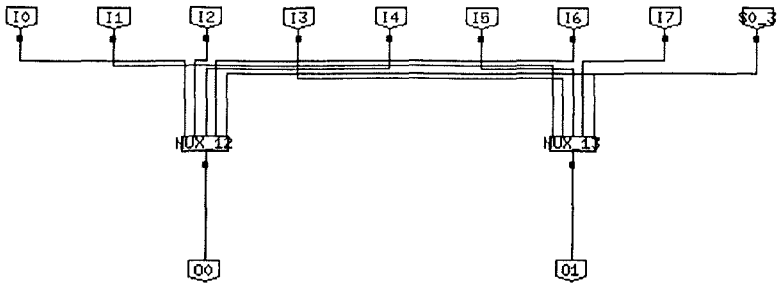

Figure 4: Top level structure of Multiplexer

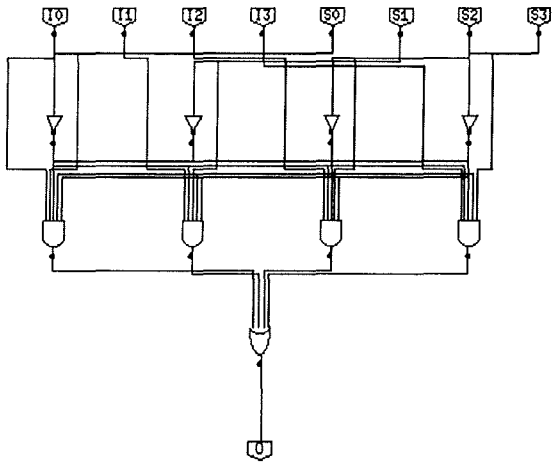

Figure 5: Structure per bit of Multiplexer

MUX_13(one for each bit), with each module sharing the four (S0 through S3) control lines. Each of these modules has the structure shown in Figure 5.

In Figure 5, we have four AND gates, one for each input. The number of inputs to each of these AND gates is one more than the number of control lines. Thus, area of these AND gates will be proportional to the ni, which is equal to the number of control lines. The total area contributed by the AND gates is proportional ni^2. Similarly, area contributed by the inverters and the OR gate before the input is proportional to ni. The total area of the multiplexer is given by the product of iw and area per bit-width of multiplexer.

$$Area = iw * (a_1 + a_2 \star ni + a_3 \star ni^2)$$

The critical path goes through an inverter, an AND gate and an OR gate, with number of inputs proportional to ni. The following delay model is developed:

$$Delay = d_1 + d_2 \star log_2\lceil ni \rceil + d_3 \star ni$$

5.1.4 Comparator

A comparator performs a subset of 6 functions: EQ, GT, LT, NEQ, LEQ and GEQ. Out of these 6 functions, the last three functions(NEQ, LEQ and GEQ) are inverse of the first three functions(EQ, GT and LT) respectively. Accordingly, the functions NEQ, LEQ and GEQ are implemented by adding an inverter to the output of the corresponding inverse functions.

[Kipp91] uses the design described in [Mano88] to implement the comparator. Based on the design of [Mano88], the comparator has been divided into three cases. For each of these three cases, separate area-delay models based on bit-width(iw) have been developed.

Case I Comparators that do not have GT or LT functions.

$$Area = a_0 + a_1 \star iw$$

$$Delay = d_0 + d_1 \star log_2\lfloor iw \rfloor + d_2 \star iw$$

Case II Comparator with only one of these two functions: GT and LT.

$$Area = a_0 + a_1 \star log_2\lceil iw \rceil + a_2 \star iw + a_3 \star iw \star log_2\lceil iw \rceil$$

$$Delay = d_0 + d_1 \star log_2\lceil iw \rceil + d_2 \star iw$$

Case III Comparator with both of the functions: GT and LT.

$$Area = a_0 + a_1 \star log_2\lceil iw \rceil + a_2 \star iw + a_3 \star iw \star log_2\lceil iw \rceil$$

$$Delay = d_0 + d_1 \star log_2\lceil iw \rceil + d_2 \star iw$$

The flowchart in Figure 6 illustrates steps to calculate area and delay for a component. In the flowchart, area(Case x) and delay(Case x) refers to the area and delay respectively generated using the above model for Case x.

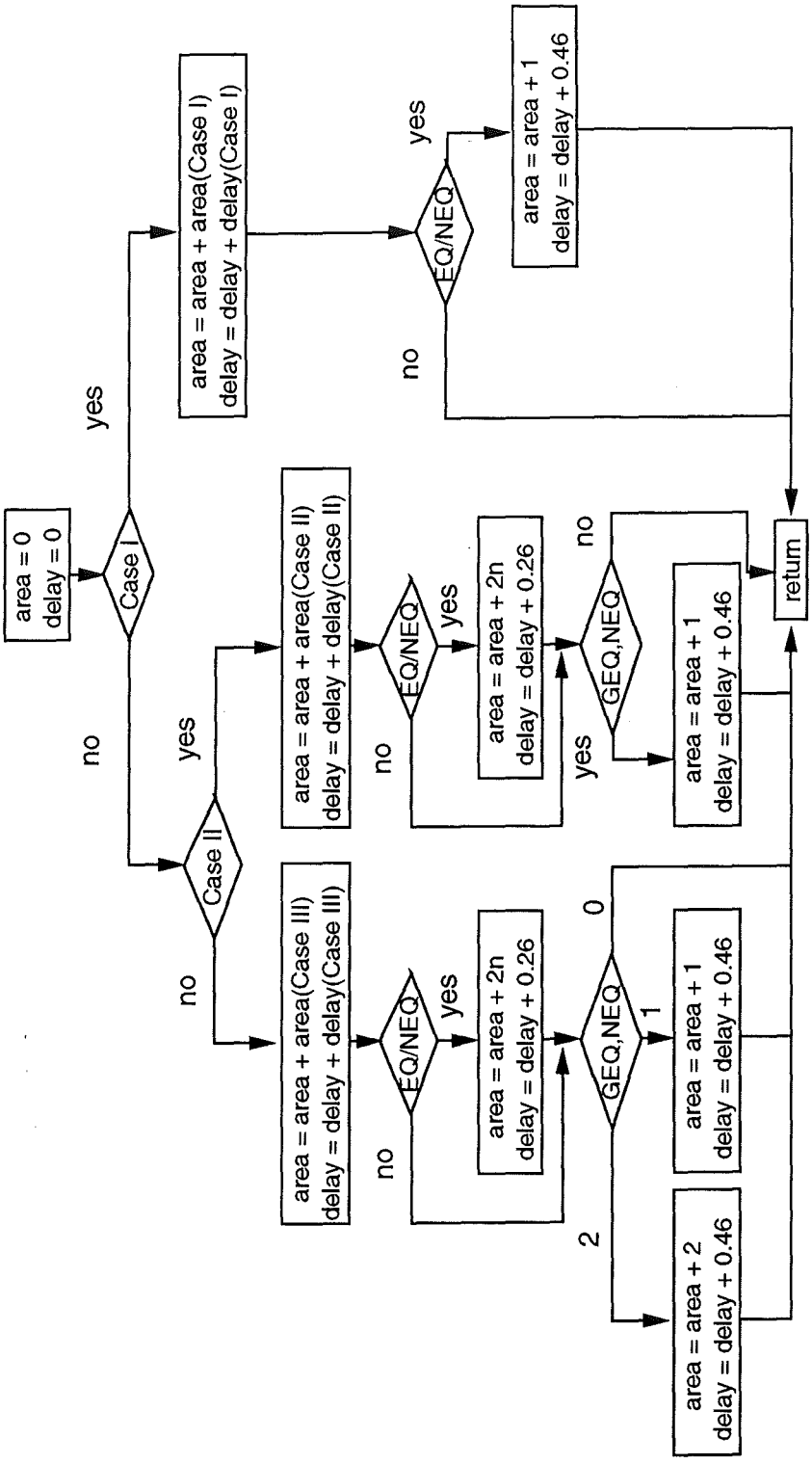

Figure 6: Flowchart to estimate area-delay for a comparator

5.1.5 Arithmetic and Logic Unit(ALU)

The generic ALU component can perform any combination of 4 arithmetic (*+, -, INC, DEC*), 6 comparisons (*GT, LT, EQ, GE, LE, NEQ*) and 16 logic functions. Any specific ALU component performs a subset of these functions for some given bit-width. An ALU is typically built using one of two styles [Mano88]:

Integrated style A basic adder with additional logic at its input and output.

Segregated style Separate arithmetic and logic blocks muxed at the output for the result.

Also, extra logic is added if the subset of functions required includes any comparison function. Based on this implementation, the ALU design has been categorized into four cases:

Case I: ALU with arithmetic functions only.

Case II: ALU with arithmetic and comparison functions only.

Case III: ALU with integrated style.

Case IV: ALU with segregated style.

We have developed area-delay models for Case I. We are working on models for the other three cases. These models are functions of input-width*(iw)*, number of functions*(nf)* and some of the terms(*CI, num1*) used to describe the functionality of ALU in [Kipp91].

$$Area = area(Adder) + a_0 + a_1 \star nf \star (CI + iw \star num1)$$

$$Delay = delay(Adder) + d_0 + d_1 \star log_2(\lceil nf \rceil) + d_2 \star log_2(\lceil CI \rceil) + d_3 \star num1$$

5.2 With respect to LAST/TELE

In this section we present the models developed for some of the component generators with respect to LAST/TELE. We followed the similar methodology described in the previous section, though some extra terms are added to the models to capture effects of wiring. Recall that LAST/TELE provides very accurate area/delay values by considering the effects of shape functions and wiring. Instead of going through the steps discussed in the previous section, we present the models directly.

5.2.1 Generic AND gate

An AND gate is characterized by the two parameters: num-inputs (*ni*) and input-width(*iw*).

$$Area = a_0 + a_1 \star log_2(\lceil ni \rceil) + a_2 \star ni + a_3 \star log_3(\lceil iw \rceil) + a_4 \star iw$$

$$Delay = d_0 + d_1 \star log_2(\lceil ni \rceil) + d_2 \star ni + d_3 \star log_3(\lceil iw \rceil) + d_4 \star iw$$

5.2.2 Mutiplexer

Multiplexer is characterized by two parameters: num-inputs(ni) and input-width(iw).

$$Area = a_0 + a_1 \star log_2(\lceil ni \rceil) + a_2 \star ni + a_3 \star iw$$

$$Delay = d_0 + d_1 \star log_2(\lceil ni \rceil) + d_2 \star ni + d_3 \star iw$$

5.2.3 Logic unit(LU)

The area-delay models for Logic unit generator is functions of input-width(iw), num-functions(nf) and number of 1's($num1$) in the table described in the example section.

$$Area = a_0 + a_1 \star nf + a_2 \star nf \star num1 + a_3 \star log_2(\lceil iw \rceil) + a_4 \star iw$$

$$Delay = d_0 + d_1 \star iw \star log_2(\lceil iw \rceil) \star num1 + d_2 \star iw^2$$

5.2.4 Encoder

The encoder generator is characterized by num-inputs(ni).

$$Area = a_0 + a_1 \star ni + a_2 \star ni \star log_2(\lceil ni \rceil)$$

$$delay = d_0 + d_1 \star ni$$

5.2.5 Shift Register

A shift-register is characterized by input-width(iw) and a set of functions from (LOAD, SHIFT-LEFT and SHIFT-RIGHT). A one-bit register is typically designed with a flip-flop associated with some logic. The one-bit structure is then replicated for iw times. The following area-delay model is used for our experiments:

$$Area = a_0 + a_1 \star iw + a_2 \star iw \star log_2(\lceil iw \rceil) + a_3 \star iw^2$$

$$Delay = d_0 + d_1 \star iw + d_2 \star iw \star log_2(\lceil iw \rceil) + d_3 \star iw^2$$

Different coefficients have been calculated for the three cases: registers with one function, registers with two functions and registers with all the three functions.

5.2.6 Adder

An adder is characterized by specifying the input-width(iw). We considered four design styles for the adder: Full ripple-carry adder, Full carry-lookahead adder, Carry-save adder and Medium adder with 4-bit CLA rippled through. For each of these we use the following models:

$$Area = a_0 + a_1 \star iw + a_2 \star iw \star log_2(\lceil iw \rceil) + a_3 \star iw^2$$

$$Delay = d_0 + d_1 \star iw + d_2 \star iw \star log_2(\lceil iw \rceil) + d_3 \star iw^2$$

Although we have same models for the four styles; value of coefficients vary based on the style.

Note that the area and delay models for a generator with respect to LAST/TELE are very similar. The area-delay metrics provided by DTAS is functional only and does not consider the effects of wiring. When wiring is taken into consideration, delay becomes dependent on the wire-length which in turn depends on the area of the component. For example, the delay model for multiplexer with respect to LAST/TELE includes terms with input-width, even though functional delay is independent of input-width. This is because area of multiplexer is proportional to input-width and with increasing area wire-length increases leading to higher delay value for the component.

6 Experiments and Results

In this section, we describe the experiments performed to test our models. We compared the estimates generated by our model against the metrics derived from the design structures generated by DTAS [Kipp91], LAST [ChKu91] and TELE [ChKu92]. The area values provided by DTAS counts the number of equivalent two-input NAND gates used to implement the component. For a component's delay (measured in nanoseconds), DTAS returns the worst-case delay for all paths through the design. LAST and TELE provide area and delay values respectively, based on GDT 3μ CMOS standard cell technology.

Our experiments attempted to cover a wide range of possible component implementations, including combinational and sequential components. We did this by generating parameter values randomly for each component generator. The number and set of functions (for multi-function components) were also chosen randomly. For each such randomly chosen component, we ran our models, and compared the results with an actual design generated by above mentioned tools.

We considered the following generators: AND, OR, NAND, NOR, XOR, XNOR, MUX, Comparator, LU, ADDER, ALU and SHIFT-REGISTER. Two parameters are needed for the Gates and MUX components: num-inputs and input-width, whereas ADDER requires only one parameter: input-width. The Comparator, Logic unit, Alu and Shift-register require not only the input-width, but also a set of functions. We first present the data in Section 5.1, and then provide an analysis in Section 5.2. In all the results presented, the percentage error is defined as :

$$Percentage\_error = \left| \frac{estimated - actual}{actual} \right| * 100$$

Besides the percentage error per test component, we present the *Coefficient of Correlation(CC)* between the metrics generated from our models and the actual metrics from DTAS and LAST/TELE. The coefficient of correlation between two variables X and Y is defined as follows :

$$Corrletion\_Coefficient = \frac{(E(X*Y) - E(X)*E(Y))^2}{(E(X^2) - E^2(X))(E(Y^2) - E^2(Y))}$$

A correlation_coefficient value of 0.0 signifies that the two variables under study are not correlated at all, whereas a value of 1.0 says that they are fully correlated. Thus, *CC* values close to 1.0 are desired for our experiments as it signifies that the metrics from our models are very close to the actual metrics.

6.1 Results

6.1.1 With respect to DTAS

Tables 2 through 12 summarize the results of our approach as compared to DTAS. For each data point, we also report the percentage error for area and delay and for each generator we report Correlation_Coeff.

Tables 2 through 7 shows the results for the six logic gates. Table 8 lists the percentage errors for six gate generators. Both the average and the maximum percentage errors are shown. For other generators, the error per test point, average and maximum errors are reported. Figure 15 depicts the aggregate results for all the components.

Table 12 enumerates the percentage error for each of the designs for an ALU component. For each ALU component with fixed parameters, we generated two-to-three alternative design implementations. The average percentage error over all the designs for each of ALU component is also listed.

Figures 7 through 14 graphically shows the actual and calculated area and delay for each of the generators in discussion. In each of these plots, X-axis represents the different test points and Y-axis represents the area/delay values. Each point on the X-axis specify a particular value of the parameter associated with the generator. For example, in LU generator, a test point is specified by a 2-tuple (ni, nf), where ni represents the parameter num-inputs and nf represents the parameter num-functions for a LU component.

As mentioned before, corresponding to each point of ALU, we have multiple implementations, each having its area-delay values. Figure 13 and 14 plots values for only two of the implementations per test points.

6.1.2 With respect to LAST/TELE

Tables 13 through 17 describe the results as compared to LAST and TELE. Table 13 lists the percentage error per test point for generic And gates, Table 14 for the Multiplexer generator, Table 15 for Logic unit, Table 16 for the Shift-register and Table 17 for the ADDER generator.

Similar to ALU, an ADDER component has four designs, each based on a style. These styles are : ripple-carry(RC), full carry lookahead(CLA), carry save(CSA) and Medium(MED) with 4-bit CLA blocks rippled. We report percentage errors for each implementation style.

Figures 16 through 21 illustrates the actual and calculated area and delay for Shift-register and ADDER generators. For ADDER generator, we plot graph for each implementation.

6.2 Analysis

6.2.1 With respect to DTAS

Tables 2 through 12 show that the maximum average error for all the generators in our study is 8.20% for area and 6.58% for delay. Thus on the average, the data generated by our model is within $\pm 10\%$ of the values generated by DTAS using the LSI library[LsiC87]. The maximum percentage

error over all generators is 19.87% for area and 22.71% for delay. However, very few data points reach these extreme values: our low averages for the errors reflect this fact.

The above tables also list the correlation coefficients for area and delay models corresponding to each generator. We observe that except for one case(the delay model for NOR gate), CC values are very close to 1.0, thus validating our approach.

Figure 15 shows the aggregate error profiles for area and delay. For area roughly one-third and for delay roughly half the data points exhibit an error of less than two percent. After this huge concentration in 0-2 percent range, the frequency of error tapers off rapidly as the error increases. For area, *77% percent of the test points have error less than 10 percent* and *95% test points have errors less than 16 percent.* For delay, figures are 87% and 94% respectively. These results validate our hypothesis about the goodness of our estimators.

Table 12 reports the errors associated with each implementation for each ALU test point. This table verifies our claim that our approach can handle multiple implementations of a given component.

Figures 7 through 14 illustrate the fidelity of our estimates for different generators with respect to DTAS. We see that the estimated area and delay follow the actual values very closely. We observe that the absolute error values are larger for the data points with bigger metric values. This phenomenon results in a relatively uniform percentage error over different test points.

6.2.2 With respect to LAST/TELE

Tables 13 through 17 show the estimation results with respect to LAST/TELE, which includes the functional and routing components. We observe that the average error for area is between 6-15%. However, the corresponding figure for delay is between 9-19%. This can be attributed to the effects of wiring delay which are sensitive to physical design characteristics such as placement and routing, and that are difficult to capture in general. The maximum percentage error over all generators is 24.09% for area and 31.42% for delay.

The correlation coefficients between the metrics from our models and the actual metrics from LAST/TELE are, in most cases, very close to 1.0, the lowest being 0.7803 corresponding to the delay model for generic AND gate.

Figure 22 shows the aggregate error profiles for area and delay. For area roughly two-fifth and for delay roughly one-third the data points, the error is less than two percent. For area, *81% percent of the test points have error less than 10 percent* and *92% test points have errors less than 16 percent.* For delay, figures are 67% and 83% respectively. Once again, these results validate our hypothesis about the goodness of our estimators.

Figures 16 through 21 depict the fidelity of our estimates for Shift-register and ADDER, with respect to LAST/TELE. We see that the estimated area and delay follow the actual values very closely for different classes of generators (combinational and sequential), as well as with respect to two different backend tools.

We conclude our analysis with two important observations. First, our test points were generated randomly (i.e., we randomly selected the component generator parameter values). Note that in a real design situation, components with certain design properties (i.e., parameters) will be invoked more often, and can be stored with precomputed metrics in the component database. This will

lower our average error. Second, our estimation models are integrated on-line with HLS tools. This is possible because of the simple estimation functions chosen, which use only a few additions, multiplications and logarithmic operations. We thus tradeoff accuracy of the metrics (i.e., $\pm 10\%$) for real-time evaluation of the estimates.

7 Summary

In this paper, we presented an accurate on-line method for estimating the area and delay of RT component generators used in high-level synthesis. Our approach has reduced the estimation problem from complex circuit analysis to a virtual table-lookup and thus has brought down the estimation time from minutes to the order of microseconds. Our approach can handle the area/delay contributed by functional blocks as well as the total area/delay including the wiring. Furthermore, we have demonstrated the estimation technique on both combinational and sequential RT components. The experiments show very good results, with aggregate errors in the range of $\pm 10\%$. Our area-delay models are simple, fast and fairly accurate, and have been integrated with an existing high-level synthesis system [LiGa88] [RaGa91]. Although our experiments were based on area/delay values generated by previously benchmarked tools and not by actual layouts, we believe that the estimation approach we presented is general and that it can produce even better results if provided with more accurate sample design data points.

8 Acknowledgements

This research was supported in part by NSF grant #MIP9009239 and in part by SRC contract #91-DJ-146. We thank Prof. Fadi Kurdahi and Champaka Ramachandran for providing us with access to the LAST and TELE estimators and Dr. James Kipps for providing us with DTAS. We thank Michael P. Donohoe for help in developing logic equation generators for each GENUS component generator. We also thank Prof. Daniel Gajski for his helpful comments throughout the development of this work.

References

[Brew88] Forest D. Brewer, "Constraint Driven Behavioral Synthesis," *PhD Dissertation, University of Illinois, Urbana-Champaign,* May, 1988.

[BRSW87] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. R. Wang, "MIS : A multiple-level logic optimization system," *IEEE Transaction on Computer-aided Design,* pp1062-1081, November 1987.

[ChKu91] F. J. Kurdahi and C. Ramachandran, "LAST: A Layout Area and Shape function esTimator for High Level Applications," *Proc. of The European Conf. on Design Automation'91* pp351-355, February 1991.

[ChKu92] C. Ramachandran and F. J. Kurdahi, "TELE: A Timing Evaluator using Layout Estimation for High Level Applications," *Proc. of The European Conf. on Design Automation'92* March 1992.

[CWGa91] V. Chaiyakul, A. C-H. Wu and D. D. Gajski, "Timing Models for High-level Synthesis," *Technical Report 91-70, University of California at Irvine,* October 1991.

[DuKi91] N. D. Dutt and J. R. Kipps, "Bridging High-Level Synthesis to RTL Technology Libraries," *Proc. 28th Design Automation Conference,* June 1991.

[Dutt88] N. D. Dutt, "GENUS:A Generic Component Library for High Level Synthesis," *Technical Report 88-92, University of California at Irvine,* 1988.

[Dutt91] N. D. Dutt, "Generic Component Library Characterization for High Level Synthesis," *VLSI Design '91: The Fourth CSI/IEEE International Symposium on VLS I Design,* New Delhi, 1991.

[GDWL92] D. Gajski, N. Dutt, A. Wu and S. Lin, "High-Level Synthesis: Introduction to Chip and System Design," *Kluwer Academic Publishers* 1992.

[Jain90] R. Jain, "MOSP: Module Selection for Pipelined Designs with Multi-cycled Operations," *Proc. IEEE Int. Conf. on Computer-aided Design'90,* pp212-215, November 1990.

[JaMl88] R. Jain, M. J. Mlinar and A. C. Parker, "Area-Time Model for Synthesis of Non-Pipelined Designs," *Proc. IEEE Int. Conf. on Computer-aided Design'88,* pp48-51, November 1988.

[JOLS92] Q. Ji, Y. S. Oh, M. R. Lighter and F. Somenzi, "Technology Independent Estimation of Area in Logic Synthesis," *Proc. Synthesis and Simulation Meeting and Inter. Interchange (SASIMI92),* pp171-180, April 1992.

[Kipp91] J. R. Kipps, "An Approach to Component Generation and Technology Adaption," *Ph D Dissertation, University of California at Irvine,* December 1991.

[KuPa89] Fadi J. Kurdahi and A. C. Parker, " Techniques for Area Estimation of VLSI Layouts," *IEEE Transactions on Computer-aided Design,* pp81-92, January 1989.

[LiGa88] J. S. Lis and D. D. Gajski, "Synthesis from VHDL," *Proc. IEEE Int. Conf. on Computer Design'88,* pp.378-381, 1988.

[LsiC87] "LSI, CMOS Macrocell Manual," *LSI Logic, Inc., Milipitas, CA,* 1987.

[Mano88] M. M. Mano, "Computer Engineering Hardware Design," *Prentice-Hall, Inc. New Jersey,* 1988.

[MiLD92] P. Michel, U. Lauther and P. Duzy (Editors) "The Synthesis Approach to Digital System Design," *Kluwer Academic Publishers* 1992.

[RaGa91] L. Ramachandran and D. D. Gajski, "An Algorithm for Component Selection in Performance Optimized Scheduli ng," *IEEE International Conference on Computer-Aided Design,* pp92-95, November 1991.

[Tyag90] Akhilesh Tyagi, "An Algebraic Model for Design Space with Applications to Function Module Generation," *Proc. of The European Conf. on Design Automation'90,* pp114-118, March 1990.

[WaCh90] D. E. Wallace and M. S. Chandrasekhar, "High-level Delay Estimation for Technology-Independent Logic Equations," *Proc. of IEEE Int. Conf. on Computer-aided Design'90* pp118-191, November 1990.

[WCGa91] A. C-H. Wu, V. Chaiyakul and D. D. Gajski, "Layout-Area Models for High-Level Synthesis," *Proc. of IEEE Int. Conf. on Computer-aided Design'91* pp34-37, November 1991.

[Wolf89] Wayne H. Wolf, "How to Build a Hardware Description and Measurement System on an Object-Oriented Programming Language," *IEEE Transactions on Computer-Aided Design'89* pp288-301, March 1989.

Parameters		*Percentage error*	
Num-inputs	*Bit-width*	*Area*	*Delay*
7	24	5.06	0.27
9	44	9.45	3.12
5	40	19.35	4.25
9	16	9.45	3.12
7	12	5.06	0.27
10	11	19.87	2.65
6	47	1.61	4.96
14	21	1.56	4.56
Average error		8.92	2.32
Maximum error		19.87	4.96
Correlation Coefficient		0.9186	0.9943

Table 2: Results for generic AND gate as compared to DTAS

Parameters		*Percentage error*	
Num-inputs	*Bit-width*	*Area*	*Delay*
14	5	5.94	3.22
10	17	3.22	0.50
10	1	3.22	0.50
11	38	5.54	0.24
9	28	13.92	0.77
7	2	6.99	1.34
5	24	3.05	3.38
Average error		5.98	1.42
Maximum error		13.92	3.38
Correlation Coefficient		0.9535	0.9899

Table 3: Results for generic OR gate as compared to DTAS

Parameters		*Percentage error*	
Num-inputs	*Bit-width*	*Area*	*Delay*
6	29	3.11	21.88
12	43	2.99	1.33
14	53	9.93	9.43
10	1	2.77	0.66
6	13	3.11	21.88
12	27	2.99	1.33
Average error		4.15	9.42
Maximum error		9.93	21.88
Correlation Coefficient		0.9720	0.9735

Table 4: Results for generic NAND gate as compared to DTAS
labelnand

Parameters		*Percentage error*	
Num-inputs	*Bit-width*	*Area*	*Delay*
7	8	6.32	14.86
14	37	4.86	10.48
10	49	13.13	13.32
6	61	10.75	10.36
12	11	8.31	8.14
Average error		8.68	11.43
Maximum error		13.13	14.86
Correlation Coefficient		0.9922	0.5214

Table 5: Results for generic NOR gate as compared to DTAS

Parameters		*Percentage error*	
Num-inputs	*Bit-width*	*Area*	*Delay*
3	30	0.0	0.0
20	1	0.0	0.0
15	58	0.0	0.0
13	16	0.0	0.0
11	22	0.0	0.0
7	2	0.0	0.0
5	24	0.0	0.0
3	30	0.0	0.0
2	57	0.0	0.0
16	63	0.0	0.0
Average error		0.0	0.0
Maximum error		0.0	0.0
Correlation Coefficient		1.0000	1.0000

Table 6: Results for generic XOR gate as compared to DTAS

Parameters		*Percentage error*	
Num-inputs	*Bit-width*	*Area*	*Delay*
14	21	0.0	0.0
4	16	0.0	0.0
10	33	0.0	0.0
16	15	0.0	0.0
6	45	0.0	0.0
12	59	0.0	0.0
5	40	0.0	0.0
3	46	0.0	0.0
36	1	0.0	0.0
13	32	0.0	0.0
Average error		0.0	0.0
Maximum error		0.0	0.0
Correlation Coefficeint		1.0000	1.0000

Table 7: Results for generic XNOR gate as compared to DTAS

Gates	*Percentage error*			
	Area		*Delay*	
	Avg	*Max*	*Avg*	*Max*
AND	7.14	19.35	2.32	4.96
OR	4.19	13.92	0.99	3.38
NAND	2.49	9.93	5.65	21.88
NOR	4.34	13.13	5.72	14.86
XOR	0.00	0.00	0.00	0.00
XNOR	0.00	0.00	0.00	0.00

Table 8: Results for Logic gates as compared to DTAS

Parameters		*Percentage error*	
Num-inputs	*Bit-width*	*Area*	*Delay*
7	2	9.65	2.96
21	8	10.11	1.70
19	30	17.97	0.03
18	25	13.70	0.82
23	25	2.12	3.32
24	25	1.22	2.97
20	5	14.11	0.87
14	5	5.13	0.51
3	5	8.46	22.71
12	27	1.51	0.12
9	27	15.83	1.77
2	27	4.22	4.80
8	7	5.18	9.21
21	7	10.11	1.70
30	7	3.74	6.17
Average error		8.20	3.98
Maximum error		17.97	22.71
Correlation Coefficient		0.9844	0.9878

Table 9: Results for Multiplexer as compared to DTAS

Parameters		*Percentage error*	
Bit-width	*#Functions*	*Area*	*Delay*
7	6	7.3	0.6
11	4	3.5	0.9
10	1	2.3	5.1
11	2	4.8	2.4
15	3	6.2	0.1
14	3	4.6	8.5
10	5	3.6	0.5
5	4	10.2	4.9
6	3	7.9	3.6
Average error		5.6	2.96
Maximum error		10.2	8.5
Correlation Coeff		0.9974	0.9890

Table 10: Results for Comparator as compared to DTAS

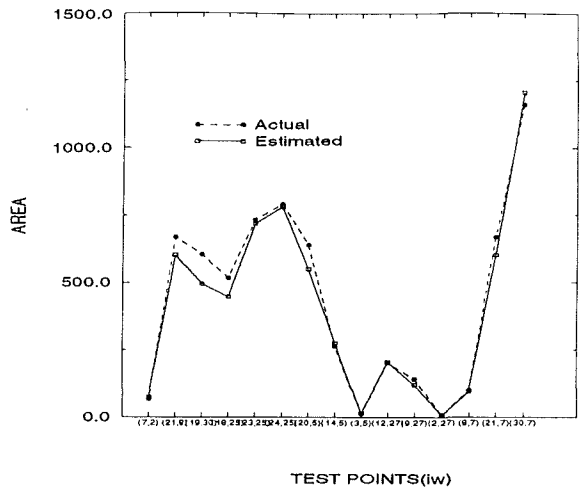

Figure 7: Comparison of actual and calculated area for Multiplexer wrt DTAS

Parameters		*Percentage error*	
Bit-width	*#Functions*	*Area*	*Delay*
6	4	0.01	2.96
2	8	0.65	1.99
20	2	17.31	13.99
6	12	7.48	5.14
30	13	6.93	4.39
26	7	1.48	4.90
19	12	7.73	3.64
12	14	4.69	3.70
16	16	1.64	3.03
Average error		5.33	4.86
Maximum error		17.31	13.99
Correlation Coefficient		0.9935	0.9870

Table 11: Results for Logic unit as compared to DTAS

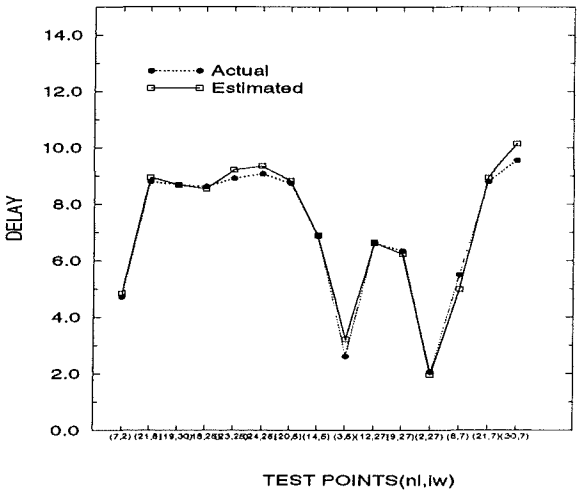

Figure 8: Comparison of actual and calculated delay for Multiplexer wrt DTAS

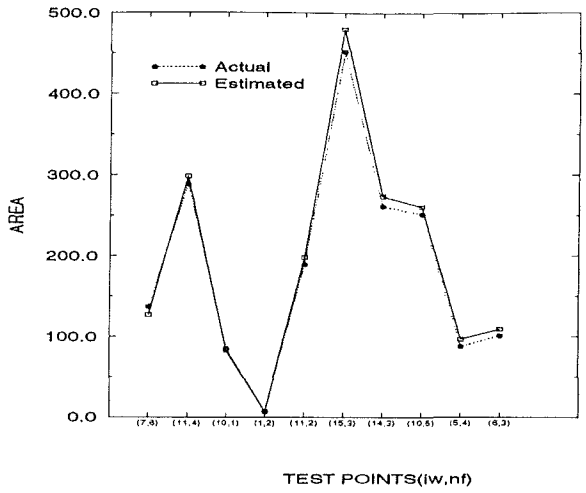

Figure 9: Comparison of actual and calculated area for Comparator wrt DTAS

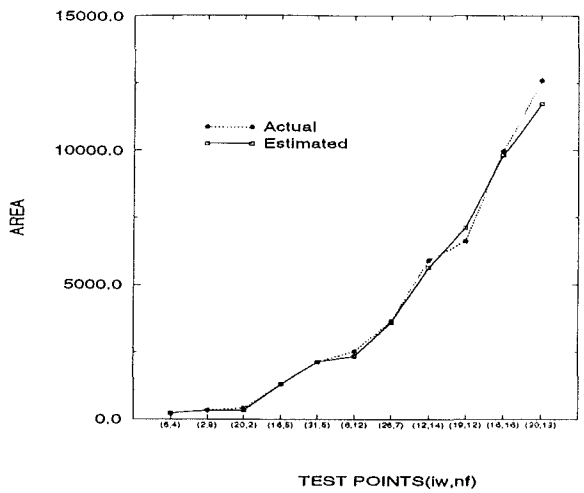

Figure 11: Comparison of actual and calculated area for LU wrt DTAS

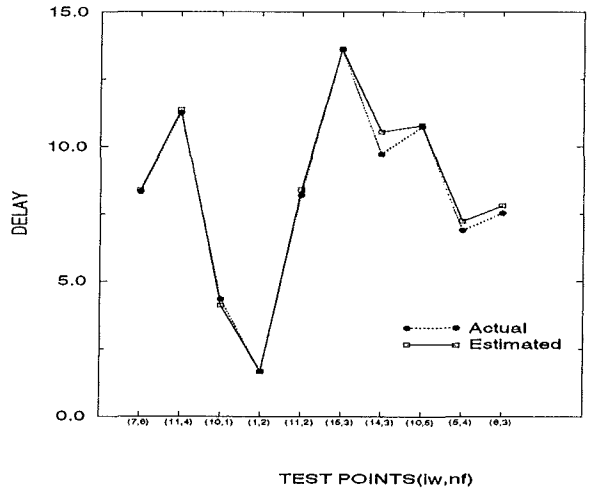

Figure 10: Comparison of actual and calculated delay for Comparator wrt DTAS

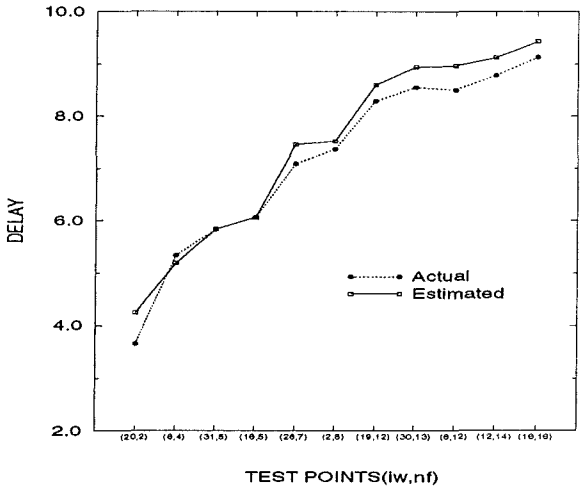

Figure 12: Comparison of actual and calculated delay for LU wrt DTAS

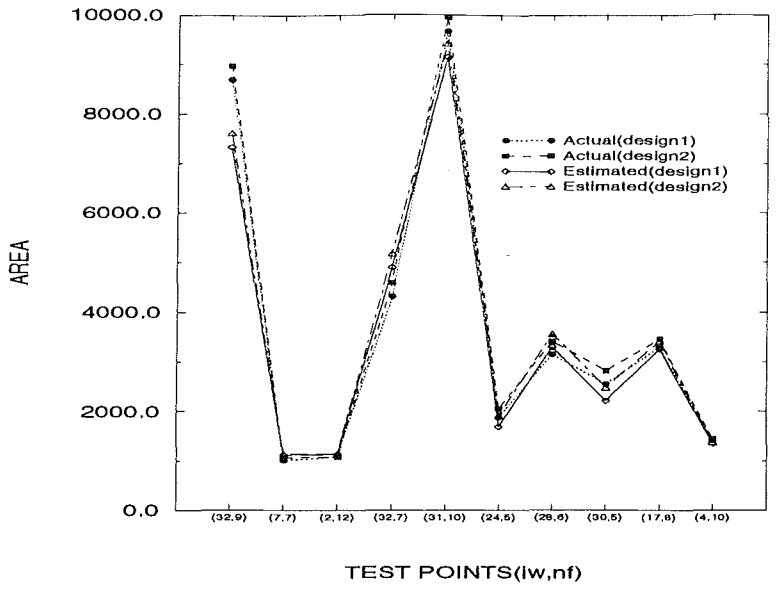

Figure 13: Comparison of actual and calculated area for ALU wrt DTAS

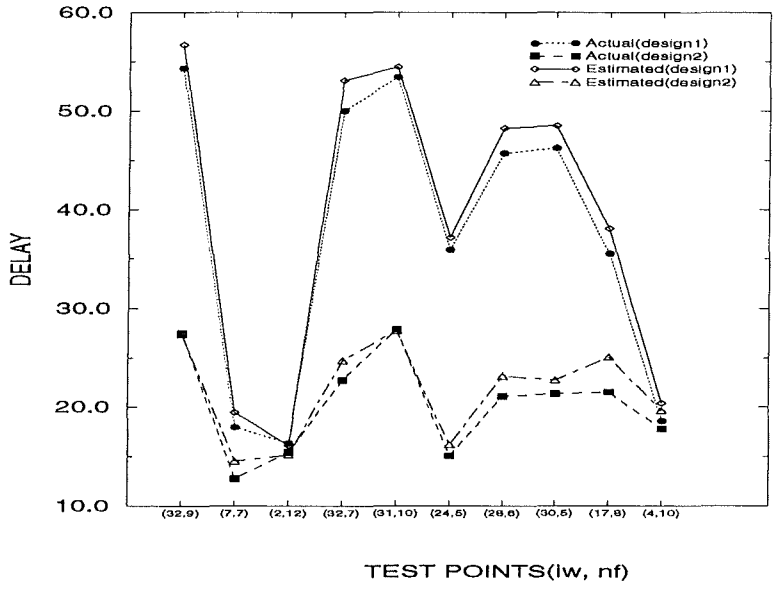

Figure 14: Comparison of actual and calculated delay for ALU wrt DTAS

Parameters		*Percentage error*							
Bit-width	*#Functions*	*Design1*		*Design2*		*Design3*		*Average*	
		Area	*Delay*	*Area*	*Delay*	*Area*	*Delay*	*Area*	*Delay*
32	9	14.65	4.34	14.18	0.54	14.16	1.17	14.48	2.02
7	7	10.93	8.34	6.38	13.73	6.30	16.00	7.48	12.65
2	12	5.45	1.52	5.43	1.74			5.44	1.63
32	7	13.64	6.14	12.79	8.70	12.75	8.27	13.06	7.70
31	10	5.40	1.90	5.25	0.46	5.24	1.53	5.30	1.29
24	5	7.71	3.45	6.87	7.41			7.30	5.44
28	6	4.69	5.55	4.33	9.66			4.46	7.00
30	5	13.22	4.91	11.96	6.41	13.10	8.06	12.32	6.13
17	8	1.24	7.12	1.18	16.39			1.21	11.76
4	10	4.79	9.66	4.71	10.44	4.69	10.49	4.73	10.20
Average error								7.58	6.58
Maximum error								14.65	16.39
Correlation Coefficient								0.9824	0.9938

Table 12: Results for ALU as compared to DTAS

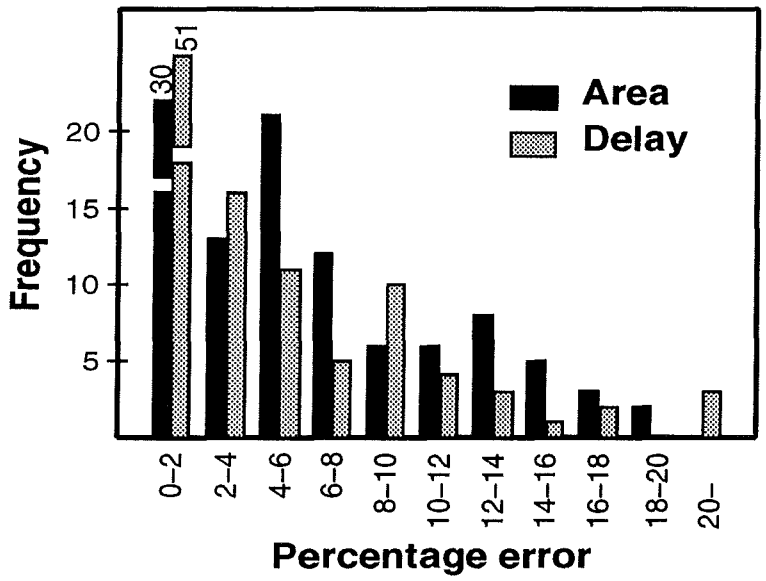

Figure 15: Aggregate Error profile wrt DTAS

Parameters		*Percentage error*	
Num-inputs	*Bit-width*	*Area*	*Delay*
10	11	5.88	31.42
7	12	4.86	26.12
9	16	5.37	26.14
4	1	11.19	10.17
14	21	23.17	6.47
7	24	1.70	20.12
4	35	0.07	11.23
9	44	0.11	13.38
6	47	8.77	15.59
Average error		8.31	18.68
Maximum error		23.17	31.42
Correlation Coefficient		0.9620	0.7803

Table 13: Results for generic AND gate as compared to LAST/TELE

Parameters		*Percentage error*	
Num-inputs	*Bit-width*	*Area*	*Delay*
25	2	1.03	7.36
2	7	7.20	2.18
3	5	15.98	17.43
5	14	2.47	17.33
5	20	0.84	8.15
7	21	1.69	6.49
7	30	16.88	2.91
7	8	2.16	9.65
Average error		6.03	8.94
Maximum error		16.88	17.43
Correlation Coeffcient		0.9835	0.8085

Table 14: Results for Multiplexer as compared to LAST/TELE

Parameters		*Percentage error*	
Bit-width	*#Functions*	*Area*	*Delay*
2	8	12.13	25.50
20	2	19.65	17.23
6	12	2.67	11.57
30	13	24.09	2.06
31	5	22.59	12.83
26	7	20.71	20.20
19	12	17.25	12.53
16	5	5.17	1.13
12	14	4.34	4.44
Average error		14.29	11.94
Maximum error		24.09	25.50
Correlation Coefficient		0.9879	0.9555

Table 15: Results for Logic unit compared to LAST/TELE

Parameters		*Percentage error*	
Bit-width	*#Functions*	*Area*	*Delay*
39	1	1.92	7.98
60	2	0.00	2.64
30	1	1.31	4.80
48	2	0.21	8.62
50	1	3.72	3.42
47	1	0.07	4.36
36	3	3.60	14.43
24	2	0.21	12.70
20	2	1.33	13.82
3	2	13.70	16.30
Average error		2.61	8.91
Maximum error		13.70	16.30
Correlation Coeffcient		0.9987	0.9941

Table 16: Results for REGISTER as compared to LAST/TELE

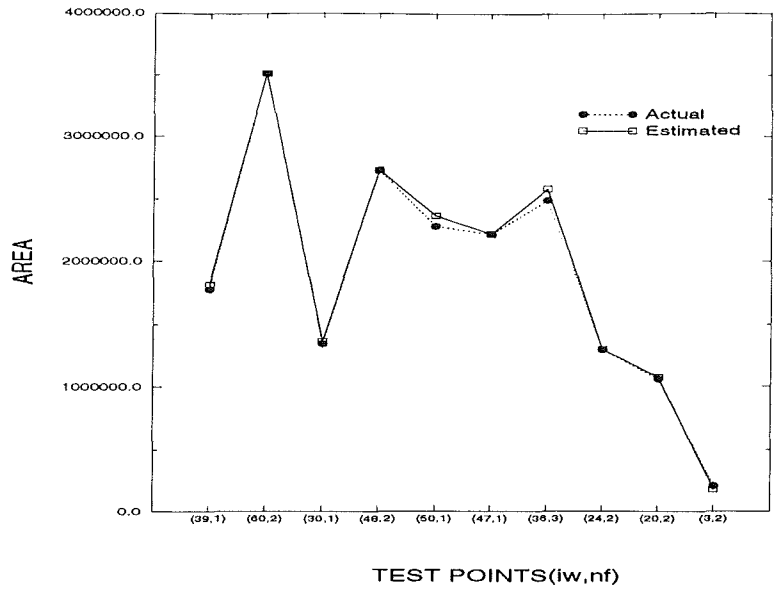

Figure 16: Comparison of actual and calculated area for REGISTER wrt LAST/TELE

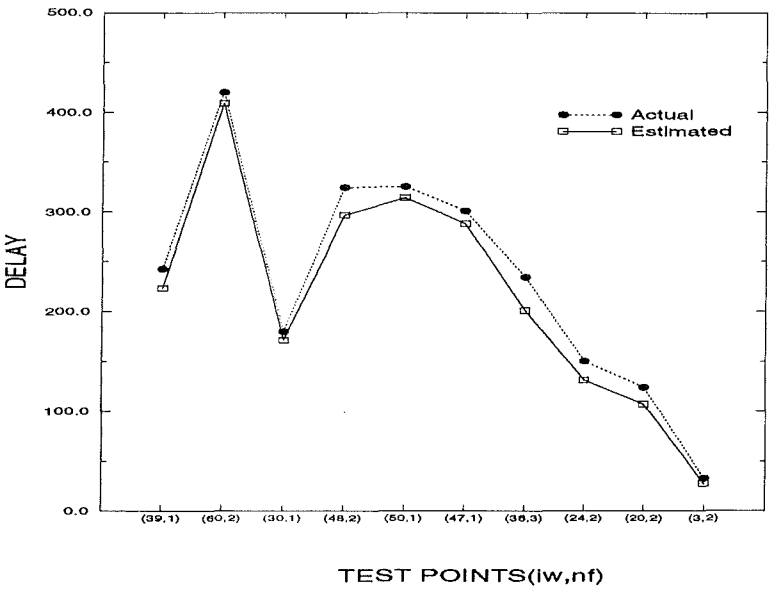

Figure 17: Comparison of actual and calculated delay for REGISTER wrt LAST/TELE

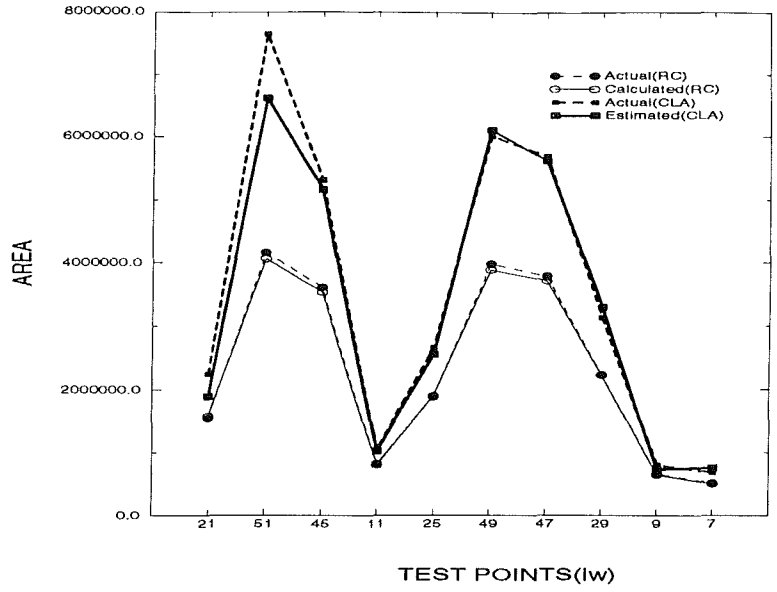

Figure 18: Comparison of actual and calculated area for ADDER wrt LAST/TELE

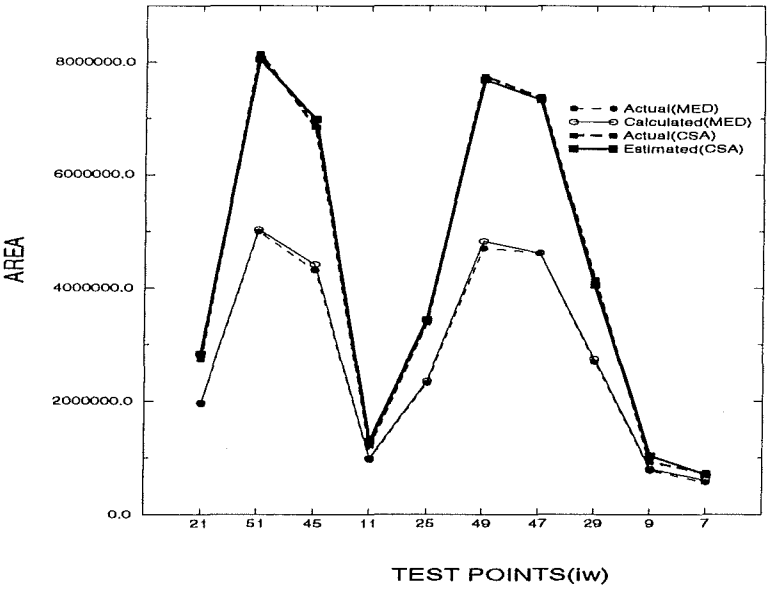

Figure 19: Comparison of actual and calculated area for ADDER wrt LAST/TELE

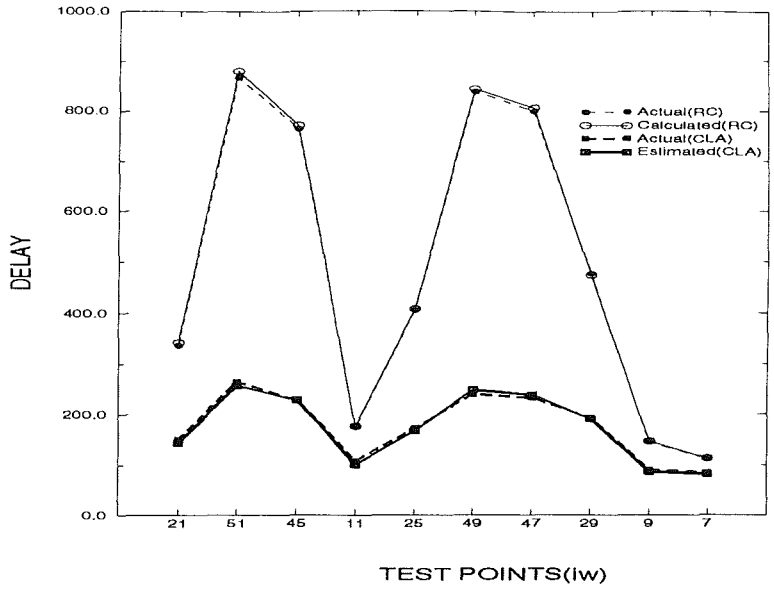

Figure 20: Comparison of actual and calculated delay for ADDER wrt LAST/TELE

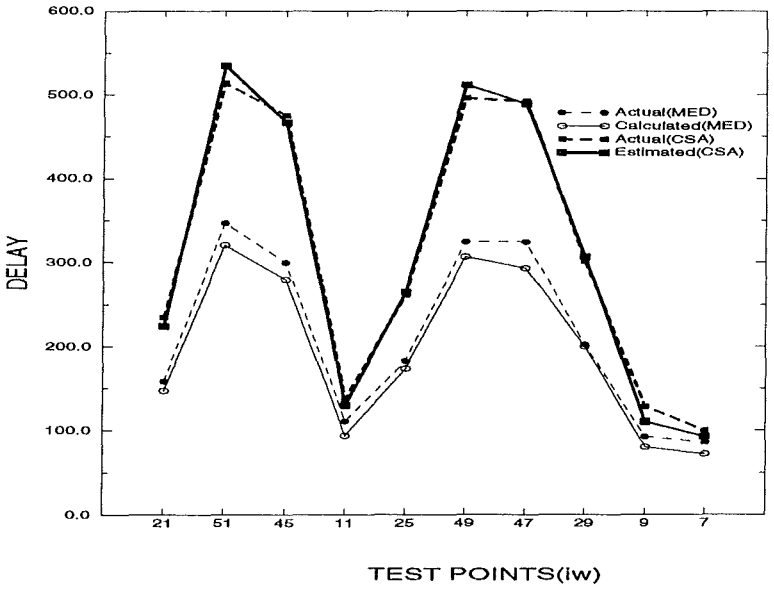

Figure 21: Comparison of actual and calculated delay for ADDER wrt LAST/TELE

Parameters	*Percentage error*									
Bit-width	*Ripple Carry*		*Carry Lookahead*		*Medium*		*Carry Save*		*Average*	
	Area	*Delay*	*Area*	*Delay*	*Area*	*Delay*	*Area*	*Delay*	*Area*	*Delay*
21	2.11	1.66	15.67	4.62	1.16	7.16	3.62	4.57	5.64	4.50
51	2.48	1.47	13.47	2.82	0.78	7.52	1.21	4.09	4.48	3.97
45	2.05	0.87	2.80	0.23	2.43	6.67	2.13	1.79	2.35	2.39
11	2.00	0.65	2.03	6.87	2.32	15.61	6.26	5.85	3.15	7.24
25	0.75	0.47	3.69	2.93	1.52	4.87	1.29	1.27	1.81	2.38
49	2.37	0.54	1.58	3.62	2.70	5.54	0.72	3.24	1.84	3.23
47	1.99	0.86	1.30	2.55	0.12	9.67	0.57	0.68	0.99	3.44
29	0.36	0.50	5.27	1.07	1.87	1.07	2.37	1.63	2.47	1.07
9	1.12	0.94	7.50	4.03	4.05	13.15	11.17	14.23	5.96	8.09
7	3.00	0.97	10.19	4.16	7.49	16.12	2.48	7.17	5.79	7.10
Average error									3.45	4.34
Maximum error									15.67	16.12
Correlation Coefficient									0.998	0.9937

Table 17: Results for adder compared to LAST/TELE

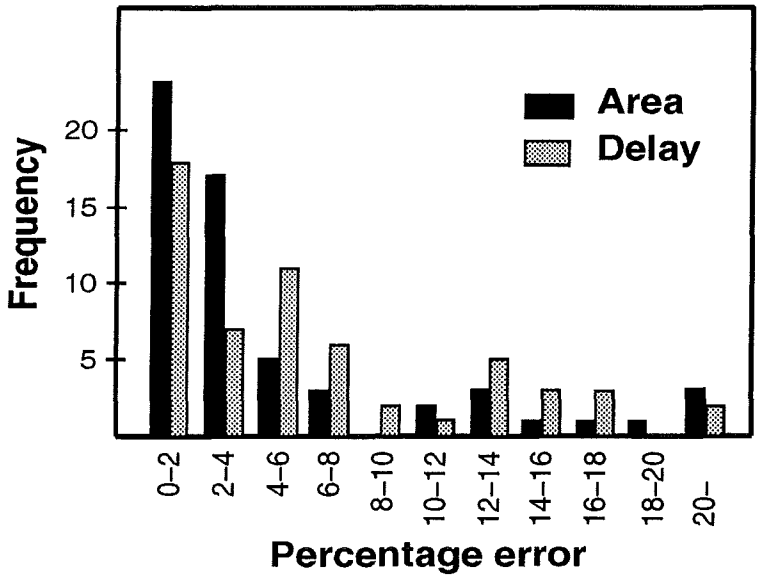

Figure 22: Aggregate Error profile wrt LAST/TELE