Linking Register-Transfer and Physical Levels of Design[1]

F. J. Kurdahi †
D. D. Gajski ‡
C. Ramachandran †
V. Chaiyakul ‡

May 27, 1993

Technical Report #93-22

†Electrical & Computer Engineering Department
‡Information & Computer Science Department
University of California, Irvine, CA 92717

[1]This work was partially supported by NSF grants #MIP-8909677 and #MIP-8922851, and CAL-MICRO grants #90-046 and #91-080. Preliminary versions of this work were presented at SASIMI-92 and ICCAD-92 Conferences.

Abstract

System and chip synthesis must evaluate candidate Register-Transfer (RT) architectures with respect to finished physical designs. Current RT level cost measures, however, are highly simplified and do not reflect the real physical design. Complete physical design, on the other hand, is quite costly, and infeasible to be iterated many times. In order to establish a more realistic assessment of layout effects, we proposed a new layout model which efficiently accounts for the effects of wiring and floorplanning on the area and performance of RT level designs, before the physical design process. Benchmarking has shown that our model is quite accurate.

Keywords: Layout, Area, Delay, Estimation, High-Level Design

1 Introduction

THE Design of a VLSI chip starts with an abstract behavioral or functional specification and ends with a detailed specification of the layout mask level geometries. In between these two stages of the design is a hierarchy of design steps (automatic or manual) involving functional partitioning, architectural and logic synthesis, floorplanning, placement and routing. The global decisions made during the initial phases of design have a pronounced impact on the final chip's performance and consequences of these decisions will not be apparent until very late in the design process. Consequently, during the early design stages, there is a need for accurate design quality metrics which can take into consideration the effects of the subsequent design steps and provide guidance for the global design decisions.

The various design synthesis tasks which take place prior to physical design usually comprise architectural (or behavioral) synthesis, register-transfer level synthesis, and logic synthesis. These tasks could be performed either manually or automatically. Each one of these synthesis tasks usually attempts to optimize a cost function representing the layout area, delay, or a combination of both. Since there is no physical layout information readily available at this point, typical area cost functions that actually get optimized are usually over-simplified models of layout design which are based solely on the contribution of active logic (i.e. gates or functional units) to both area and delay. Such abstract metrics do not accurately reflect the quality of the real physical design because they do not consider important layout effects such as wiring and floorplanning. For large designs, these physical design considerations tend to dominate layout area and delay. The lack of such realistic modeling of layout could result in "misleading" the synthesis tasks into generating inferior designs. This was indeed the conclusion in McFarland's work [1]. As technology provides smaller and smaller feature sizes and higher density, it is expected that physical effects will play an even more important role in the overall design area and performance. Therefore, such effects can no longer be ignored.

1.1 *Previous work*

Most of the previous work in developing predictive models of layout was done at the gate or transistor levels. The standard cells style, being the most popular design method for custom random logic applications was studied by researchers, and predictive models of standard cell layouts were developed. Most notable is the work by Pedram and Preas [2], [3] who developed accurate analytical models for area and wire length estimation, and Zimmerman [4] who developed a novel slicing technique for estimating the area and shape function of custom layouts. All these models were benchmarked and found to predict the area of standard cell layouts with errors around 5 to 10%. The work in [5, 6] describes a layout area and delay prediction approach using a hardware model which combines analytical and constructive predictive models of layout. By controlling the relative contribution of each of the two sets of models to the prediction results, it is possible to tradeoff accuracy of the estimate versus runtime.

These gate level predictive models are of great value in early chip planning and layout tasks (such as floorplanning). However, in order to address the specific problems in high level synthesis described earlier,

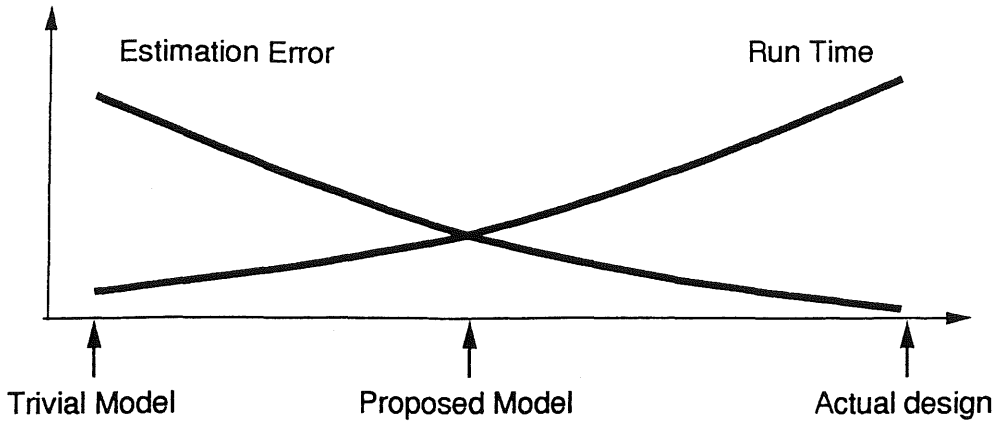

Figure 1: Error vs efficiency of predictive models

predictive models must operate at a higher level of design abstraction. The output of a typical high level synthesis phase consists of RT level components, control, and memories. Clearly, such a design would not be laid out as a large piece of random logic. Thus, any high level predictive model must be able to handle several variations in design styles and must reflect the effects of high level design tradeoffs on silicon accordingly.

In [7] and [8], abstracted layout area and timing models for high level synthesis were presented. These models were experimentally shown to accurately and efficiently reflect the effects of the data path design tradeoffs on the final layout. However, these models concentrated on modeling the datapath and controller separately and did not consider the impact of floorplanning which could generally be a significant factor in area and delay.

1.2 *Contributions of our work*

In this paper, we propose an overall layout model which can be used to efficiently obtain accurate area and timing information of a given RT level design. Clearly, accuracy and runtime efficiency are mutually competing goals as shown in Figure 1. Simple hardware models (usually analytical) are quite fast to evaluate but are generally not accurate. More complex models (usually constructive) are generally more accurate but also more costly to evaluate. In the extreme case, one can get exact values of design quality measures by going through the full physical design steps. However, this would be too costly, especially if a large number of candidate RT level designs are to be evaluated when going through iterative design improvement steps. Our model offers a reasonable compromise between accuracy and efficiency and furthermore, allows the user to tradeoff these two criteria. Our additional contributions over previous work are the following:

- We model chip level designs (using the Finite State Machine with Datapath, or FSMD paradigm [9]), as opposed to gate level designs, thus achieving a higher level of abstraction compared to [4], [2], and [5] without sacrificing accuracy or efficiency,

- we incorporate floorplanning information into the prediction and have provably good wiring models of global interconnect. Thus, our prediction model accurately reflects the effects of layout and floorplanning on area, timing, and system clock cycle time. Therefore, our system can achieve chip level modeling whereas [7] and [8] concentrated on blocks level modeling and estimation, and

- we have validated our model with fully constructed and simulated layouts of standard high level synthesis benchmarks, and some real industry standard designs, using industrial layout tools.

In developing our models, we sought as much as possible to maintain simplicity and runtime efficiency without sacrificing accuracy. Both accuracy and efficiency are primary criteria at this level of abstraction, accuracy for obvious reasons, and efficiency because during early design and synthesis tasks, a large number of candidate designs (or design decisions) are typically generated and must be evaluated as fast as possible. An inefficient estimator can easily become a runtime bottleneck and thus render the encapsulating synthesis tool intractable in runtime. Since these two goals are competing, one must concentrate on developing estimators which can provide the most beneficial tradeoff between accuracy and efficiency.

In order to compare difference predictive models we introduce two measures: *predictor quality* and *predictor fidelity*. Given a design description D, and let $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ be a set of different implementations of that description. Given metric type M of design quality (say area or performance, for example), consider two predictive models, A and B for M. Given a finite amount of runtime, T, and that models A and B can predict the value of metric M of the same implementation D_i of D (call the predictions $\widehat{M}_A(D_i)$ and $\widehat{M}_B(D_i)$, respectively) with relative errors of $E_A(D_i)$ and $E_B(D_i)$, respectively, then the prediction quality of Model A is higher that the prediction quality of Model B if

$$\frac{1}{n}\sum_{i=1}^{n}|E_A(D_i)| < \frac{1}{n}\sum_{i=1}^{n}|E_B(D_i)| \tag{1}$$

Predictor fidelity is another important property defined as follows: Consider two designs $D_i, D_j, 1 \leq i, j \leq n$. Define μ_{ij} as follows:

$$\mu_{ij} = \begin{cases} 1 & if\ (M(D_i) - M(D_j))(\widehat{M}_A(D_i) - \widehat{M}_A(D_j)) > 0, \\ & i \neq j,\ 1 \leq i, j \leq n \\ 1 & if\ M(D_i) - M(D_j) = \widehat{M}_A(D_i) - \widehat{M}_A(D_j) = 0, \\ & i \neq j,\ 1 \leq i, j \leq n \\ 0 & \text{otherwise} \end{cases}$$

Then, the *fidelity* of Model A (denoted by $\mathcal{F}(A)$) can be defined as a percentage by the following equation:

$$\mathcal{F}(A) = 100 \times \frac{2}{n(n-1)} \sum_{\substack{i,j=1 \\ i<j}}^{n} \mu_{ij} \tag{2}$$

For example Model A has perfect (100%) fidelity if $\mathcal{F}(A) = 100$. The experimental results described in Section 7 indicate that our models are not only highly cost-effective, but that they achieve a high-fidelity when compared with other existing models.

Finally, while our layout model is clearly not the only possibility to go from RT level to silicon, we maintain that our approach at the chip level allows for the incorporation of other design styles. In that case, one must develop new estimation model for each new style used within the chip model.

The paper is organized as follows: Section 2 presents some preliminaries on modeling of digital VLSI systems. The datapath models are presented in Section 3, and the controller models are described in Section 4. The chip level models are described in Section 6. Section 7 shows some experimental results which demonstrate the accuracy of the model. Conclusions are drawn in Section 8.

2 Preliminaries

2.1 *General modeling of digital systems*

A digital system can generally be described using the FSMD (Finite State Machine with Datapath) model [9] shown in Figure 11. Ideally, one would like a predictive layout model to read in design specifications in which only high level constructs are described and partitions the FSMD specifications into four structurally distinct blocks, which are:

1. datapath,
2. controller,
3. macros (e.g. multipliers), and
4. memories.

Clearly, the automatic generation of such a partitioning is a problem of considerable difficulty which has only recently been addressed by researchers [10]. As a first step in abstracting the level of predictive models, we assume for now that the input to our model is a an FSMD [9] description of the system as a partitioned RT level design in which components have been assigned to structurally distinct blocks.

The chip layout of such a description is composed of blocks which can be either hard macros (e.g. datapath, multipliers, memory, PLA), or soft macros (e.g. controller, random logic). Thus, in order to model the chip level layout, we must estimate the dimensions of each of these blocks, and use these estimates to predict the dimensions of the complete chip. The flow of our modeling technique is as follows: we first compute estimates of the dimensions, as well as the timing of each RT level block[1]. These models are described in Sections 3, 4, and 5. Next, our chip level model uses the estimates of the RT level components to obtain an approximate topology of the complete chip as well as the wiring component connecting the

[1] Given the uncertainty in aspect ratio (which is not known until the whole chip is laid out), our models produce *shape functions* for each block. Shape functions are further treated in Section 6.

blocks together. From this topology, estimates of chip area and timing information (clock cycle) can be derived.

The subsequent sections will describe area and timing models for datapath, controller (random logic or PLA), and macrocells (including memories), as well as models for the overall chip.

2.2 *Electrical timing models*

All of our high level timing models are based on the well known lumped RC model, also called the Elmore delay model [11], which is widely used for delay calculation. In this model, the propagation delay along a path from the start point to the end point (t_p(start,end)) is computed as a product of lumping all of the resistances R_j and capacitances C_k along the path, that is,

$$t_p(start, end) = \sum_j R_j \times \sum_k C_k. \tag{3}$$

We can use Equation 3 to obtain the delay of a connecting wire between two components, or between two blocks of components. In CMOS technology we model a component as having input capacitance (C_{in}) and output resistance (R_{out}). The well-known π-model is used to model the connecting wire having capacitance C_w and resistance R_w.

The propagation delay, $t_p(n)$, through a wire n connecting the output of ($comp_i$) and driving load components ($comp_j$, $1 \leq j \leq n$) can be computed as

$$t_p(n) = (R_{out_{(comp_i)}} + R_w)(C_w + \sum_{j=1}^{n} C_{in_{(comp_j)}}). \tag{4}$$

Thus, the delay for signals to propagate from the input of $comp_i$, through a wire n, to one of the inputs of component $comp_j$, driven by $comp_i$, is

$$t_p(comp_i, n, comp_j) = t_p(comp_i) + t_p(n). \tag{5}$$

where $t_p(comp_i)$ is the internal delay of component $comp_i$.

This model can be generalized to account for the total delay through a path $P(I, O)$ in a circuit starting from input pin I to output pin O by adding all the component to component delays and the net delays along $P(I, O)$ as follows:

$$t_p(P(I,O)) = \sum_{\forall comp_k \in P(I,O)} t_p(comp_k) + \sum_{\forall n \in P(I,O)} t_p(n) \tag{6}$$

Clearly, this model is accurate when considering systems running at medium to high frequencies. For very high frequencies (200MHz or more) the accuracy of the lumped RC model decreases and other more elaborate models must be used instead. In addition, for very fine technologies (0.5 μ or less) other factors such

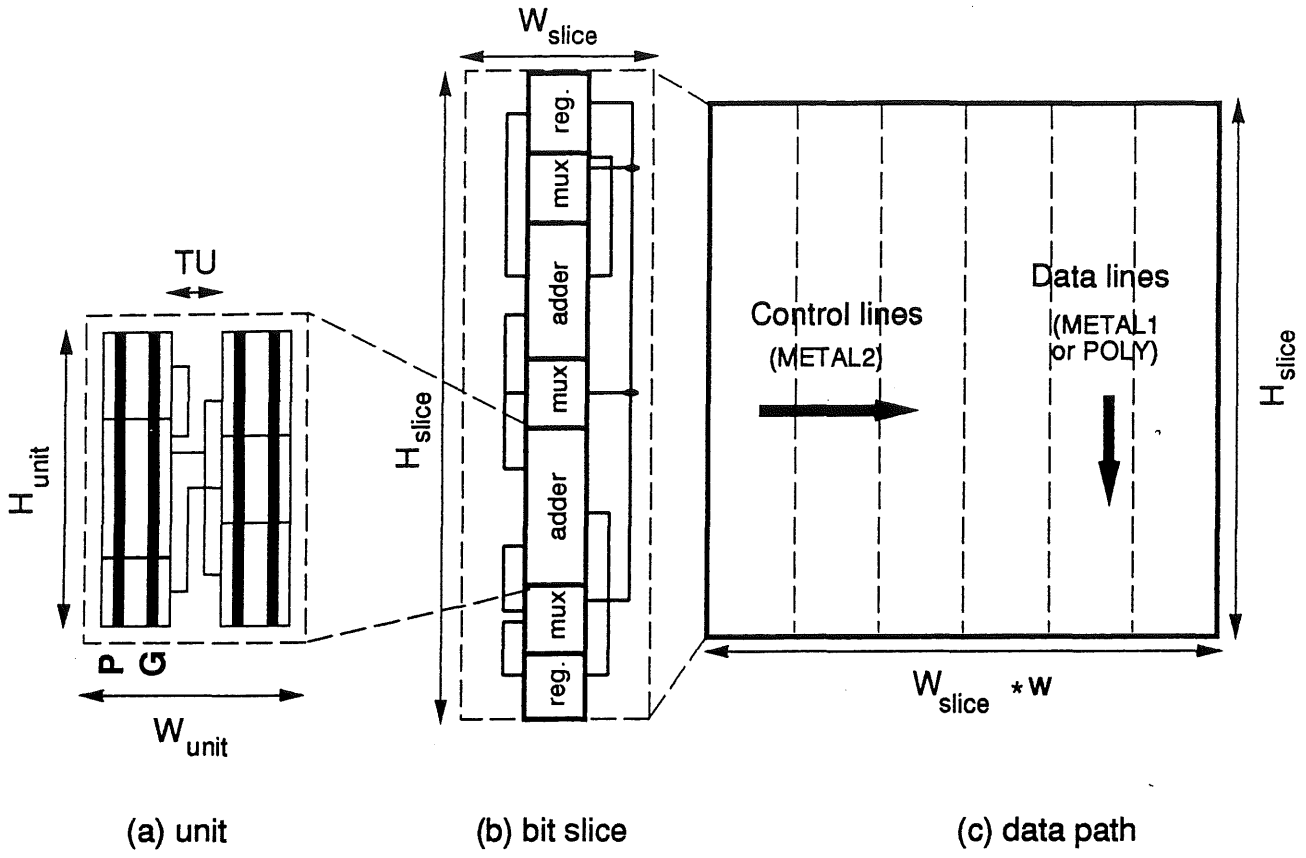

Figure 2: The datapath layout architecture

as the input waveform and layout pattern dependencies must be considered when estimating the propagation delay.

3 Datapath Layout Model

Figure 2 shows a popular layout architecture for datapaths known as bit-sliced standard cells, where each slice of a microarchitecture unit is realized with standard cells. This architecture is currently being used in industry (see [12], for example). In this architecture, standard cells of each bit slice are placed vertically in one or more rows and routing channels are used to connect different cells inside one complete bit slice. Control lines run horizontally in second metal and power and data lines run vertically in first metal or poly. In the following sections, we develop area and timing models for this architecture. Should other approaches be used to layout the data path, other area and timing models should be developed to accommodate such approaches.

3.1 *Datapath area model*

We conservatively assume that the standard cell technology used for the data path does not have over-the-cell routing tracks, hence a routing area is always needed. Thus, the total area of a bit slice is the sum of the unit areas and the routing area. The wiring area is proportional to the number of routing tracks used to connect units. We can first calculate the area of one bit slice, and obtain the total area by multiplying the area of one bit slice with the stack's bit width. The area of each bit slice consists of two parts: the *unit* area and the *wiring* area.

Unit area. Each unit consists of one or several rows of standard cells, where the height of each row is fixed and the width of each standard cell is proportional to number of transistors (see Figure 2(a)). Both dimensions are assumed to be known from technology and cell library information. We first describe the wiring area estimation model for units composed of one standard cell row each. Essentially, the model assumes a linear placement of cells where terminals are uniformly distributed such that probability of a wire emerging for a pin is $\frac{N}{w}$, where N is the number of two-point nets[2], and w is the total number of pins on the linear placement (which is also proportional to the total cell width. The length of a wire is assumed to be a random variable, L, with a probability density function $p_L(l) = pq^{l-1}$, where $1/p$ is the average wire length in pin pitches, estimated empirically using Feuer's formula [14], and $q = 1 - p$. Thus, the expected number of tracks at a point x on the linear placement, $E\{d(x)\}$ is estimated as the number of wires born before (or to the left of) x, and terminating after (or to the right of) x. given by:

$$E\{d(x)\} = \frac{N}{wpq}(1 - q^x)(1 - q^{w-x+1}). \tag{7}$$

Thus, the number of tracks needed for routing inside the unit, TU is estimated as $TU_{est} = \max_{1 \le x \le w} E\{d(x)\}$, which occurs at $x_{max} = \lceil \frac{w+1}{2} \rceil$. For large units, a single row layout may result in overly long slices which may be inefficient to layout. In this case, units are laid out using multiple rows as shown in Figure 2(a) where each unit is laid out using two standard cell rows. The above single row model is extended to handle multiple row cases. The multiple row model is described in [13] and is used in this case to compute TU_{est}.

Once TU_{est} is computed, we can estimate the width of the wiring channel as $\beta \times TU_{est}$ inside a unit, where β, is the wiring pitch (or the minimum spacing between two adjacent wires) which depends on the design rules of the layout technology. The total width of a unit, W_{unit}, can now be computed as:

$$W_{unit} = (\beta \times TU_{est}) + r \times w_{row}, \tag{8}$$

where r is the number of rows in the unit, and w_{row} is the width of each row. The unit height, or H_{unit}, is

[2]For modeling purposes only, multi-point nets are decomposed into two point segments. Thus, N is actually the number of *equivalent* two-point nets

estimated as:

$$H_{unit} = \frac{1}{r} \sum_{i=1}^{m} H_{cell}(i), \tag{9}$$

where $H_{cell}(i)$ is the span (or height) of cell i and m is the number of cells in the unit. Now, the total unit area, $A_{unit} = W_{unit} \times H_{unit}$.

Wiring area. Given a particular technology, the wiring area depends on the routing density which can only be determined after bit-sliced units have been physically placed. However, the placement procedure is often expensive in terms of computation runtime. Hence, we use an inexpensive, yet reasonably accurate placement model for wiring estimation. Given a netlist, we use a highly efficient min-cut partitioning heuristic to approximate the linear placement of the units along the bit slice (Shown in Figure 2(b)). This would result in a bitslice unit having a height of $H_{slice} = \sum_{j=1}^{n} H_{unit}(j)$, where $H_{unit}(j)$ is the height of $unit(j)$, $unit(j)$ is a functional unit, register, or interconnect unit; and n is the # of units in one bit slice.

Next, the linear placement is scanned and the track density is estimated at the beginning (or top edge) of each unit j, call it x_j. Let $d(x_j)$ be that estimate, the number of routing tracks in the bit slice is now estimated as TS_{est}, given by:

$$TS_{est} = \max\{\max_{1 \le j \le n} \{d(x_j)\}, d(H_{slice})\}, \tag{10}$$

where $d(H_{slice})$ is the estimated density at the edge of the bit slice. Thus, the width of the inter-unit wiring area, $W_{wire} = \beta \times TS_{est}$, and the width of the bit slice, W_{slice}, is

$$W_{slice} = \max_{1 \le j \le n} \{W_{unit}(j)\} + W_{wire}. \tag{11}$$

Finally, as shown in Figure 2(c), the overall area cost of the datapath, A_{dp} is

$$A_{dp} = (W_{slice} \times w) \times H_{slice}, \tag{12}$$

where w is the bit width of the datapath.

Note that our chip level model (which will be described in Section 6) allows the user to partition the datapath into several blocks where each block would be bit-sliced and laid out separately. Thus, this modeling technique is flexible enough to allow the designer to experiment with different ways of partitioning the datapath, ranging from a single datapath block, all the way to a multi-block strategy in which each block represents the layout of an individual RT level component.

3.2. *Datapath delay model*

Given the datapath architectures described in Section 3.1, computation of the propagation delay from one datapath component to another in the same datapath block requires two elements: internal delay of the

component and wiring delay. Typically, the internal delays of components are provided by the targeted component library.

In order to estimate the wiring delay we need to estimate the wiring length. The length of a net can be estimated based on the approximate topology provided by the area model. Given this information, the propagation delay between two datapath components ($comp_i$ and $comp_j$) via a net (n) is computed as follows: let (x_i, y_i) and (x_j, y_j) be to coordinates of $comp_i$ and $comp_j$ respectively. If both components are on the same bitslice, then $x_i = x_j$ and the length $L(n)$ of net n is $|y_i - y_j|$. If the components are in different bitslices k and l, respectively, then $L(n)$ can be estimated as follows:

$$L(n) = |y_i - y_j| + |k - l| \times W_{slice} \tag{13}$$

Where W_{slice} is the wifth of a single bit slice, computed as shown in Equation 11. Given $L(n)$, the wiring delay can now be estimated using Equations 4 and 5.

4 Controller Models

Given an RT level datapath design, the controller (usually specified as a state diagram, or state table) is usually implemented in two phases. First the state table is input to state encoding, logic synthesis, and technology mapping tools (e.g. MIS, NOVA), and hence a gate level netlist is obtained. Next, the netlist is placed and the interconnections routed (typically in standard cells design style).

A complete model of the controller from state diagram necessitates a modeling of the logic as well as the physical design phases. Modeling the impact of logic synthesis is an extremely complex task which has received very little attention in the past. Given the difficulty of such a modeling, and the efficiency requirements of the overall prediction modelling, our models of logic synthesis are simple but efficient in nature.

4.1 *Controller area modeling*

A control unit can be described by a control state-table that specifies next-state and control signals as a function of present states and conditional signals. We assume that the present states and the next states are encoded as binary values $p_k...p_1p_0$ and $n_k...n_1n_0$, where $k = \lceil \log_2 S \rceil - 1$ and S is the number of states, respectively. Thus, the total number of inputs to the control unit I equals $\lceil \log_2 S \rceil + C$, where C is the total number of conditional signals.

We consider the impact of optimization procedures by deriving a simplified model that is geared toward estimation of the lower bound delay of the control logic. In our model, each next-state and control signal is represented as sum of products of the present-state and conditional/status signals. The product term is implemented with AND gates and the sum with OR gates. Since, the target component library will

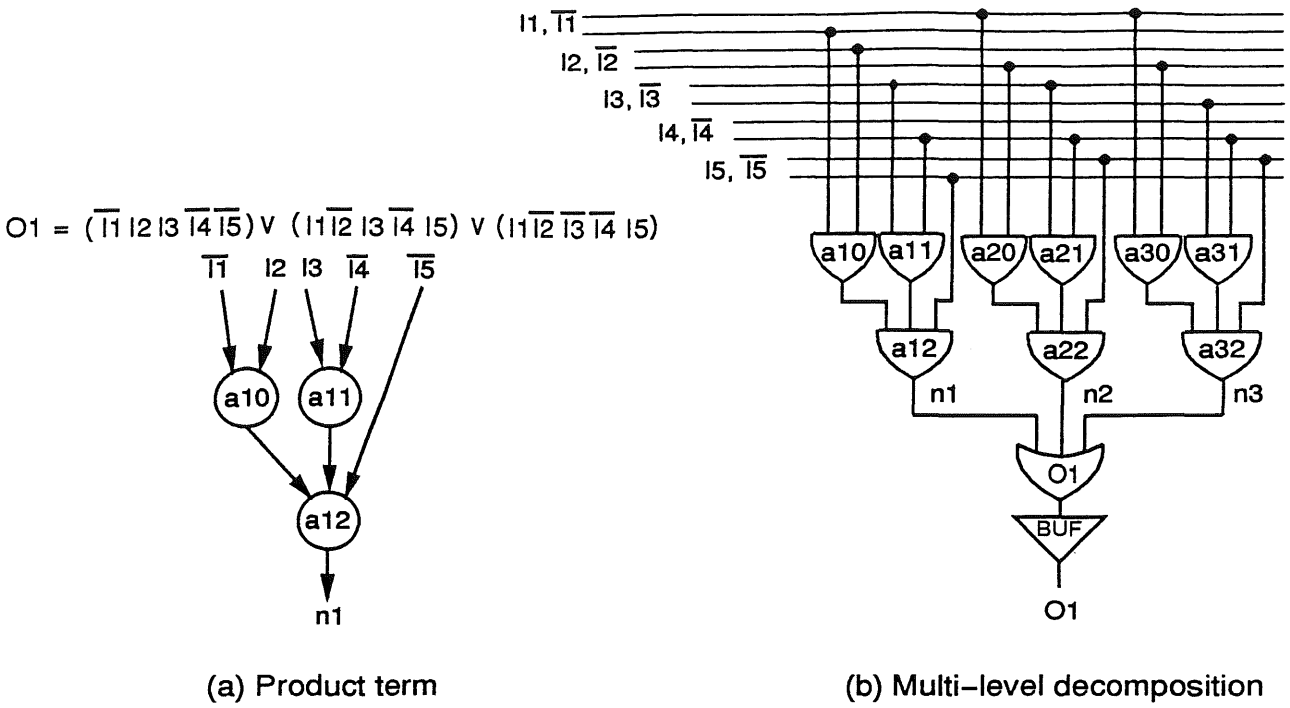

Figure 3: Random logic layout model:(a) decomposition of a product term, and (b) a multi-level implementation.

usually provide AND and OR gates with a limited number of inputs, the sum and product terms need to be decomposed into a multi-level implementation when the large AND or OR gates are not available in the target library.

Figure 3 shows an example of a multi-level implementation of a sum-of-products expression with maximum three inputs AND gates. The multi-level implementation is represented by a trinary tree shown in Figure 3(a) and its equivalent gate implementation is shown in Figure 3(b). The netlist generated as such partially models the effects of logic optimization.

The multi-level decomposition aims to produce an implementation with the minimal number of levels. This is guided by the fact that a multi-level implementation of a product term with I number of literals using AND gates with a maximum of n inputs is in the form of an n-ary tree; where each internal node in the tree denotes an AND gate, each leaf denotes a literal, and each edge denotes a net in the gate-level implementation. The height of the tree equals $\lceil \log_n I \rceil$, and the critical path of the product terms is denoted by the path that defines the height of the tree.

Given the netlist generated as shown above, we use a constructive/analytical model to estimate the shape function of the controller block. This enables the chip level layout model (described in Section 6) to estimate the aspect ratio which results in the most optimal packing of the overall chip floorplan. Essentially, the constructive/analytical model starts with a circuit netlist as input. The netlist is then recursively partitioned to generate a slicing tree, but instead of slicing all the way down to the cell level (as in fully constructive approaches, such as [4]), the depth of the slicing tree is restricted to a small number of levels for reasons

of runtime efficiency. We use the analytical predictor described in Section 3.1 at the leaf level to generate estimates of the shape functions for the leaf circuits. The slicing tree is traversed bottom up and the shape functions of the sibling nodes are used to form the shape function of the parent node. This procedure is carried out all the way up to the root node, at which point, a prediction of the chip shape function is obtained (see Figure 4). Since only a prediction of the controller dimensions is desired, we need not decide on the orientation of the slice line when traversing the slicing tree bottom up. Instead, we only keep track of the slicings (horizontal or vertical) which results in a smaller area configurations at each intermediate node in the slicing tree[3]. It can be proven [4] that this procedure does not result in the loss of any optimal points on the shape function. The main advantage of this approach is that it offers a reasonable compromize between accuracy and runtime efficiency.

When the slicing tree is traversed bottom up, and the shape functions of parent blocks are formed, the predicted dimensions of the parent block must include an estimate of the wiring area needed to route the nets connecting the two blocks. Given any two sibling blocks A and B, and a vertical or horizontal slice orientation the wiring area is estimated by first computing the track utilization factor α_A as $\alpha_A = \frac{T_A w_A}{N_A}$, where T_A is the number of tracks in block A, w_A is the width of block A and N_A is the number of nets in block A. In order to accommodate the extra routing needed for the interblock nets, we need to add some extra tracks in blocks A and B. We assume that this addition will not appreciably change the track utilization factor α_A in T_A. Let N_{AB} be the number of nets crossing between blocks A and B. We can estimate T_{Aw}, the number of tracks needed in block A after routing the inter-block nets as

$$T_{Aw} = \frac{T_A w_A (N_A + N_B + N_{AB})}{N_A (w_A + w_B)}, \tag{14}$$

and a similar estimate is derived for T_{Bw}. Given T_{Aw}, T_{Bw} and the dimensions of A and B, the total area of the parent block AB can now be estimated. This procedure is carried out all the way to the root of the slicing tree to obtain an estimate of the controller shape function.

4.2 *Controller timing model*

The controller timing model assumes that the controller netlist is described as blocks of random logic (next state, and output logic) bounded by state or status registers as depicted in Figure 11. Given this netlist, the delay in the combinational logic bounded by registers R_i and R_j can be expressed as the sum of three delay components on the worst case input to output path, call it $Path$, between R_i and R_j. The first two components are the contributions of the intrinsic cell delay and the fanout delay in $Path$ and can be estimated from the cell library and the netlist. In order to estimate the third component (i. e. the wiring delay), we use the Elmore delay model of Section 2.2 which requires the knowledge of the wire length, $L(n)$, of each net n in $Path$.

[3] This composition technique is further explained in Section 6.1

In order to estimate $L(n)$, we use the approximate layout topology generated by the area model described in Section 4.1. The area model essentially generates an approximate placement of the combinational block in which the relative locations of the leaf clusters are known. The size of the leaf cluster depends on the depth of the slicing tree as described in Section 4.1. The location of a particular cell inside the leaf cluster has yet to be determined in order to get an exact placement. Given this partial placement information, the length of each wire is estimated as the sum of two components: the *intercluster* component, and the *intracluster* component, as shown in Figure 4. This approach results in accurate and efficient estimates of wire lengths. Let $L(n)$ be the total length of net n which spans $N(n)$ clusters: $cluster_1, cluster_2, \cdots, cluster_{N(n)}$.

Intracluster components. The length of an intracluster net component inside any cluster i can be estimated as the length of the spanning tree connecting the cells inside the cluster. We use an analytical estimator developed by Feuer [14] to estimate $Awl(i)$, the average length of a two-point net segment in leaf cluster i. This average wire length value is used to weigh the edges of the spanning tree. Thus, the total contribution of the intracluster components, or $L_a(n)$ can be estimated as

$$L_a(n) = \sum_{i=1}^{N(n)} (nl(i,n) - 1) * Awl(i), \tag{15}$$

where, $nl(i,n)$ is the number of cells belonging to leaf cluster i and connected to net n.

Intercluster component. Once all the intracluster components of a net have been determined, the length of the net between leaf clusters (or intercluster component) can be estimated as the length of the spanning tree that connects the leaf clusters. The distance between the centers of the leaf clusters (as estimated from the partial placement) is used to weigh the edges of the spanning tree. Thus, the length of the intercluster component, or $L_e(n)$ can be estimated as: $L_e(n) = MST(cluster_1, cluster_2, \cdots, cluster_{N(n)})$, Where $MST(cluster_1, cluster_2, \cdots, cluster_{N(n)})$ is the length of the minimum spanning tree connecting clusters 1 through $N(n)$ by their center points. Given $L_a(n)$ and $L_e(n)$, the total length of net n, $L(n) = L_a(n) + L_e(n)$, and the wire resistance of net n, $R_w(n) = \frac{L(n)}{W_w} R_s$, and capacitance, $C_w(n) = L(n) W_w C_s$, where W_w is the width of the routing wire and R_s and C_s and the sheet resistance and capacitance per unit wire length, respectively.

Once the net delays are estimated, we can estimate the circuit delay through any input-output path by applying the Elmore delay model described in Section 2.2. We need to determine the delay between input signal I_i and output signal O_j, call it $t_p(I_i, O_j)$. Let $Path(I_i, O_j)$ be the longest path connecting the input pin I_i and output pin O_j, which can be determined using topological sorting of the circuit netlist (See [6] for more details). Thus, the worst case delay between pins I_i and O_j is $t_p(Path(I_i, O_j))$ computed using Equation 6.

$Path(I_i, O_j)$ provides a *topological* estimate of the worst case delay. However, in a combinational block described at the gate level, the topological delay estimation scheme may be overly pessimistic because of the possible *false paths* [15] in the circuit. In this case, the user may as an option, invoke a *functionality-based*

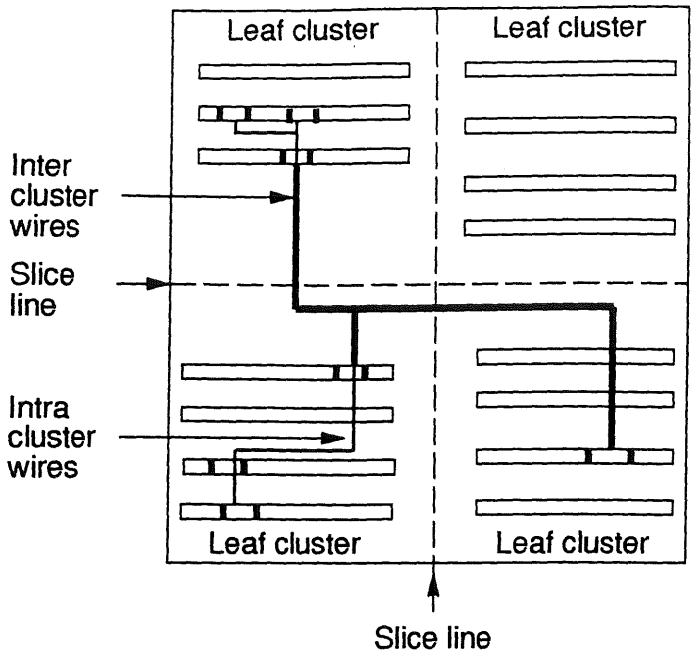

Figure 4: Wire length estimation for random logic

delay estimation heuristic model which attempts to find the true worst case path at the expense of increased runtime. More details on the functionality based technique can be found in [16].

5 Macrocells and memories

Some system components can be laid out as Macrocells. Macrocells can be either pre-designed, or parameterized. In the first case, the dimensions of the block and also the pin locations, are known in advance. Parameterized components are usually designed by replicating a bit slice of a bit cell in one or two dimensions by using generators such as GDT [17]. In this case the overall block dimensions can be easily calculated by using simple equations derived from the generating function. Other types of Macrocells include memory and PLA blocks.

A typical memory *macro* can be described using a number of *blocks* such as row decoder, column decoder, sense amp, cell array and I/O ports. These *blocks* are arrays of smaller cells called *modules*. For example, a cell array can be built using memory cells placed in an array structure. Similarly, the row decoder is made of r number of row decoder cells, where r is the number of rows in the memory. A typical static RAM memory configuration is shown in Figure 5. Let r and c be the number of rows and columns in the memory block, respectively. Let h_{row} and w_{row} be the height and width of the memory cells, h_{cd} be the height of the column decoder, h_{sa} the height of the sense amplifier, and w_{rd} the width of the row decoder. Thus, the area of the memory block can be estimated as:

$$A_{MEM} = (r \times h_{row} + h_{cd} + h_{sa}) \times (c \times w_{row} + w_{rd}) \tag{16}$$

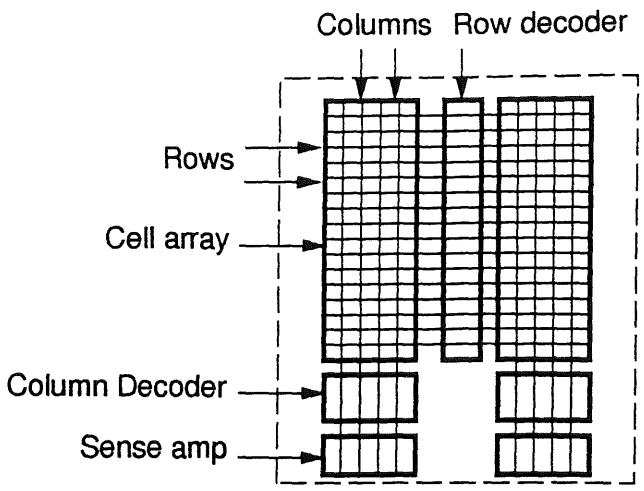

Figure 5: Macrocell layout for a static RAM

The layout of a PLA is modelled as a rectangular block which contains latch-buffer circuitries and an AND-OR plane, as shown in the Figure 6.

The width of the block is the sum of the width of the input latch-buffer circuitry, the width of a min-term buffer, Mb_w, and the width of the output latch-buffer circuitry. The width of the input and output latch-buffer is the product of maximum width of the latch (l_w) or the buffer (b_w) with the number of inputs (In) and outputs (Out), respectively. Whereas, the height of the block is computed as the sum of the latch height (l_h), the buffer height (b_h), and the height of the AND-OR plane. The height of the AND-OR plane is determined by the product of the number of min-terms ($Mint$) and the vertical space between transistor rows in the plane (v_{sp}). Thus, the area is calculated as follows:

$$\begin{aligned} A_{PLA} = \; & [(In + Out) \times MAX(l_w, b_w) + Mb_w] \\ & \times [l_h + b_h + Mint \times v_{sp}] \end{aligned} \tag{17}$$

6 Chip-Level Models

Once the dimensions of each block are determined, the chip level model uses these estimates to find an approximate topology of the complete chip, using the technique described in Section 6.1 below. Given this approximate topology, information about the the chip area and timing (i.e. clock cycle, delay) can be estimated.

6.1 *Chip level area model*

Our chip level area model uses a slicing tree technique derived from [4] for evaluating the area of designs implemented using macro blocks and standard cell clusters. This technique is described next.

The standard cells grouped into clusters called standard cell blocks and are treated as a single entity

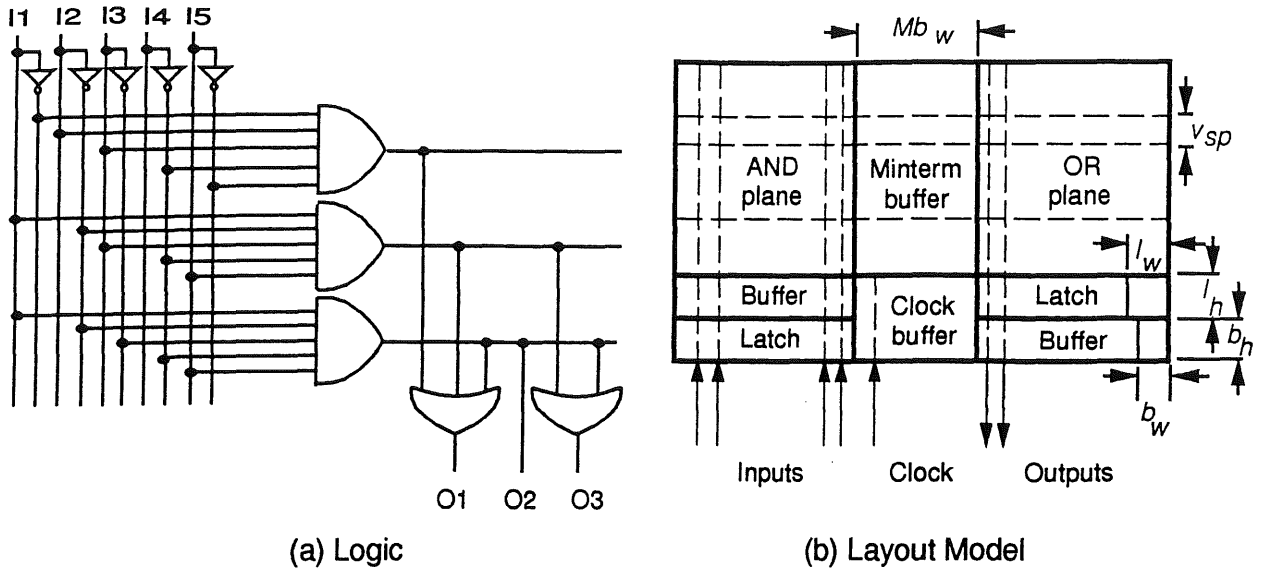

Figure 6: PLA layout model

while building the slicing tree and computing the shape function of the design. The shape function of the standard cell block can be evaluated using the model described in Section 4 with varying number of standard cell rows.

The chip level slicing tree technique involves slicing down to the leaf blocks which consists of either macros or standard cell clusters. This constructive approach does not consume excessive runtimes since the number of leaf blocks are limited to a relatively small number. This technique is illustrated in Figure 7.

The previous subsections discussed the techniques to evaluate the shape function of leaf blocks which are either macro blocks or standard cell blocks. The shape function of the entire design is computed by constructively adding the shape function of these leaf blocks. In addition to the area of the leaf blocks, the wiring area used by the nets connecting these blocks also needs to be accounted for.

As shown in Figure 7, the design could be implemented using hard macros (multiplier, memory) or soft macros (controller). The layout of hard macros is fixed and hence the pin locations are known. In the case of soft macros, the pin locations have to be approximately determined to the extent of the side location.

The pin side location can be determined by evaluating the shape function of the design prior to wiring area calculation. This process determines the approximate location of the blocks in the design. From the connectivity information, the blocks connected to any pin can be determined. By evaluating the mean location of these blocks, a preferred side location of each pin can be determined and is designated as pseudo-pin for each slicing tree window shown in Figure 8.

The area of the design is built up by looking at a window of the slicing tree as shown in Figure 8. This window consists of slice level i and slice level $i+1$. The area of the slices can be obtained by estimating the wiring channels at the slices. The shape function of the pseudo block AB is determined by the height and width of channels $H1$, $H2$, $H3$ and $V1$, $V2$, $V3$ and $V4$. Pseudo block AB is a composite block and could be composed of parallel slices or orthogonal slices. Let $x_m y_n$ be the number of nets connecting pins in side

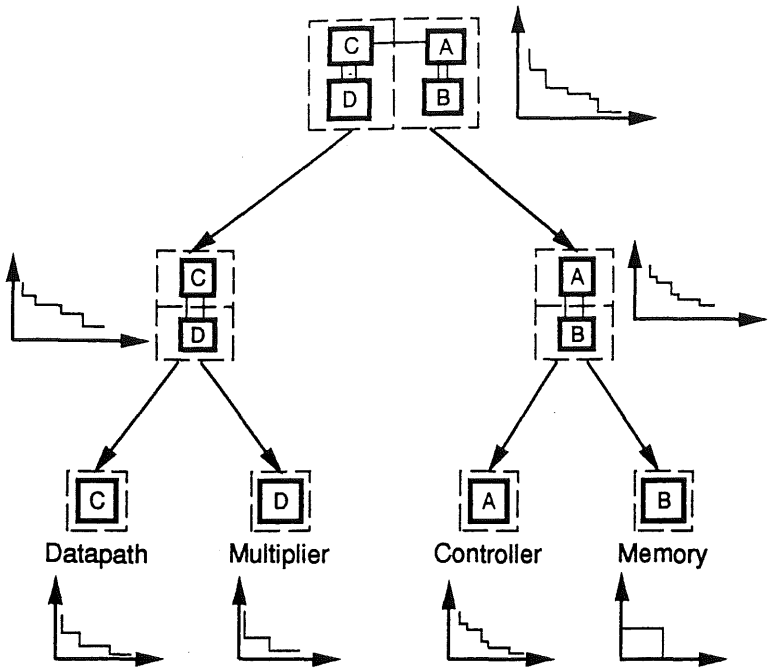

Figure 7: Constructive/analytical area estimation technique

x of block m to pins in side y of block n, where $x, y \in \{t, b, l, r\}$ as shown in Figure 8. Assuming a parallel slice, the widths of the channels around block A are estimated using the following formulas:

$$H1 = (t_1t_2 + t_1r_2 + t_1l_2 + t_1b_2)/2 \quad (18)$$

$$H2 = (2b_1t_2 + b_1r_2 + b_1b_2 + l_1r_2)/2 \quad (19)$$

$$V1 = (t_1t_2 + 2t_1l_2 + t_1b_2 + l_1t_2 + l_1l_2 + l_1r_2 + l_1b_2)/2 \quad (20)$$

$$V2 = (t_1t_2 + 2t_1r_2 + t_1b_2 + r_1r_2 + r_1l_2 + r_1b_2)/2 \quad (21)$$

Similarly, given an orthogonal slice, the widths of the channels around block A are estimated using the following formulas:

$$H1 = (t_1l_2 + t_1r_2 + t_1t_2 + l_1r_2/3 + l_1t_2 + l_1l_2/4)/2 \quad (22)$$

$$H2 = (b_1l_2 + b_1r_2 + b_1l_2 + b_1b_2 + l_1l_2 + l_1b_2 + l_1r_2/3)/2 \quad (23)$$

$$V1 = (t_1l_2/4 + t_1b_2 + b_1t_2)/2 \quad (24)$$

$$V2 = (3t_1l_2/4 + t_1b_2 + b_1l_2/4 + b_1t_2 + r_1r_2 + r_1b_2 + r_1l_2/4 + r_1t_2 + l_1l_2/4)/2 \quad (25)$$

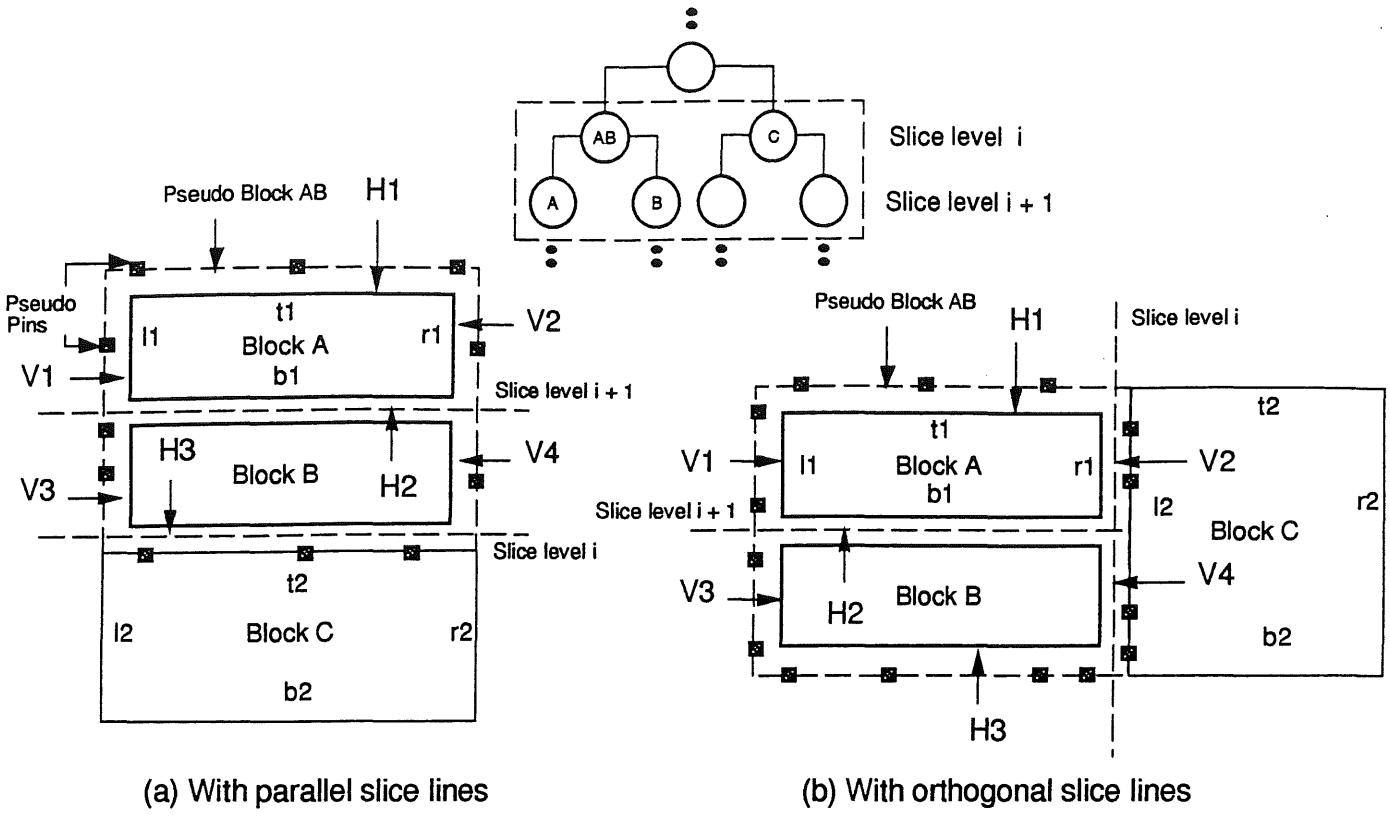

Figure 8: A Mixed design wiring estimation

A similar set of formulas can also be applied to block B to estimate $H3$, $V3$, and $V4$. Once the channel widths are known. They are added to the total estimated area of block AB as shown in Figure 8.

This process of building the composite blocks is performed in a post-order manner from the leaves of the slicing tree towards the root, thus determining the area of the entire design. This technique allows us to model the different types of blocks that constitute the various system components and to accounts for the effects of global routing. For each two sibling blocks, there exists two possible ways of generating the parent block depending on the orientation of the slice. It is easy to see that this would result in a very large number of combinations very rapidly, even if only the non-inferior points of the shape function are retained. To reduce the problem to manageable proportions, we use an approach similar to the one reported in [4]. Since only a prediction of the chip dimensions is desired, we need not perform an actual floor plan of the chip from the slicing tree. Therefore, we need not decide on the orientation of the slice line when traversing the slicing tree bottom up. For each two siblings, two shape functions of the parent block are generated: one assuming a horizontal slice and another assuming a vertical slice. The two curves are then superimposed and a "lower bound" curve is generated by keeping only the smaller of the two slice orientations at each x as shown in Figure 9. The resulting shape function is taken as the set of predicted dimension pairs for the optimal layout area of the parent block. Zimmerman [4] has proven that if carried out up to the root node of the slicing tree, this method will indeed result in a shape function of the whole chip containing only the

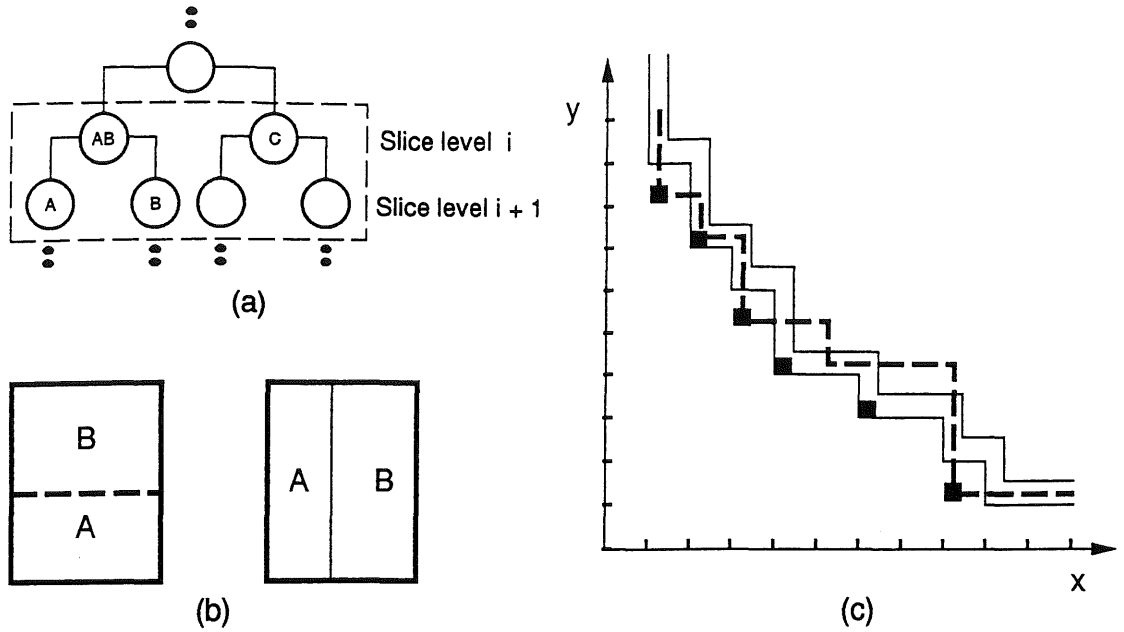

Figure 9: Constructive prediction: (a) the slicing tree, (b) possible AB decomposition, and (c) predicted shape function of AB (points shown as black squares)

non-inferior [4] layout configurations (see Figure 9).

At the end of this phase, we can estimate the area of the overall chip, and, more importantly, we have an approximate topology of the chip which can be used in the subsequent timing models described next.

6.2 *Chip level timing model*

In general, the netlist of a combinational block may consist of standard cell blocks (which are used to implement the datapath units, e.g. muxes, adders, as well as controllers) and of Macrocells which may implement more complex functions (such as combinational multipliers). We assume that all the timing information of these components is readily available from various standard cell and Macrocell libraries.

Given such a combinational network, our chip level area model described in Section 6.1 outputs an approximate floorplan which provides estimates of the relative locations of the constituent blocks and hence, estimates of the interblocks connection lengths. In order to estimate these connection length, we need to determine the pin locations of the connection terminals. Our model assumes three levels of information on pin locations as illustrated in Figure 10. For predesigned blocks (e.g. Block B in Figure 10), the pin locations can be exactly known. For other types of blocks, the pin location can be either completely unknown (Block A), or known up to the side location (Block C). In the first case, the pin coordinates are estimated at the center of the block. In the second case, the pin coordinates are estimated at the midpoint of the block side.

Given the terminal pin locations of a net n, we can use Equations 4 and 5 to compute its propagation

[4] A configuration is non-inferior if there does not exist any other configurations having one of its dimensions equal to the non-inferior one but has smaller area.

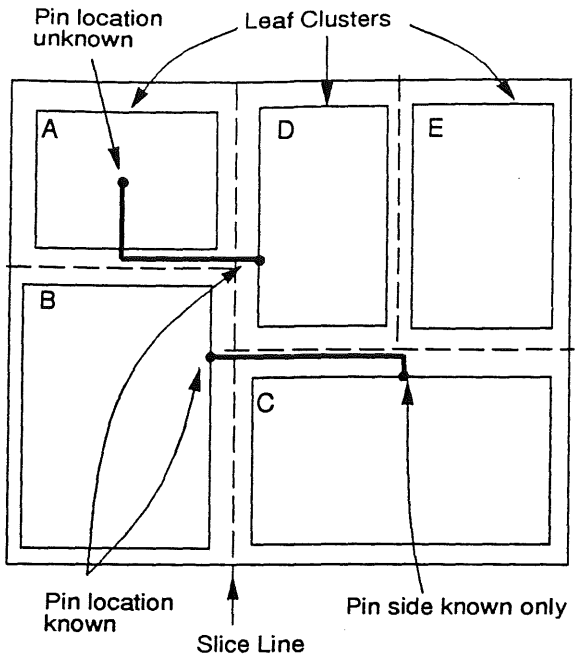

Figure 10: Interblock wire length estimation

delay. Let (A_x, A_y) and (B_x, B_y) be the coordinates of the pins connected by n. The length of n is estimated as $L(n) = |A_x - B_x| + |A_y - B_y|$. Thus, $R_w(n)$ and $C_w(n)$, the inter-block wiring resistance and capacitance, respectively, can now be estimated as described in Section 4.2. The propagation delay of n is then computed using Equations 4 and 5.

6.3 *Clock cycle model*

The clock cycle is determined by the worst register-to-register delay that includes the propagation delays in the control unit, in the datapath and between blocks. In our implementation, the clock computation is based on a simple Finite-State-Machine Datapath model (FSMD), as shown in Figure 11.

In Figure 11, the critical path (*Path1*) is from the *State register*, through the *Control logic*, the *Datapath*, the *Next-state logic* and back to the *State register*. Thus, the clock period is the sum of the propagation delays of the *State register* ($t_p(State\ register)$), the *Control logic* ($t_p(Control\ logic)$), the *Datapath* ($t_p(DP)$), the *Next-state logic* ($t_p(Next\text{-}state\ logic)$), and the set-up delay of the *State register* ($t_{setup}(State\ register)$), that is,

$$\begin{aligned} t_{clock} = \; & t_p(State\ register) + t_p(Control\ logic) \\ & + t_p(DP) + t_p(Next\text{-}state\ logic) \\ & + t_{setup}(State\ register) \end{aligned} \tag{26}$$

$t_p(Control\ logic)$ and $t_p(Next\text{-}state\ logic)$ are computed using one of the models described in Section 4 and include the wiring delay at the output of each block. $t_p(DP)$ is determined by the worst register-to-

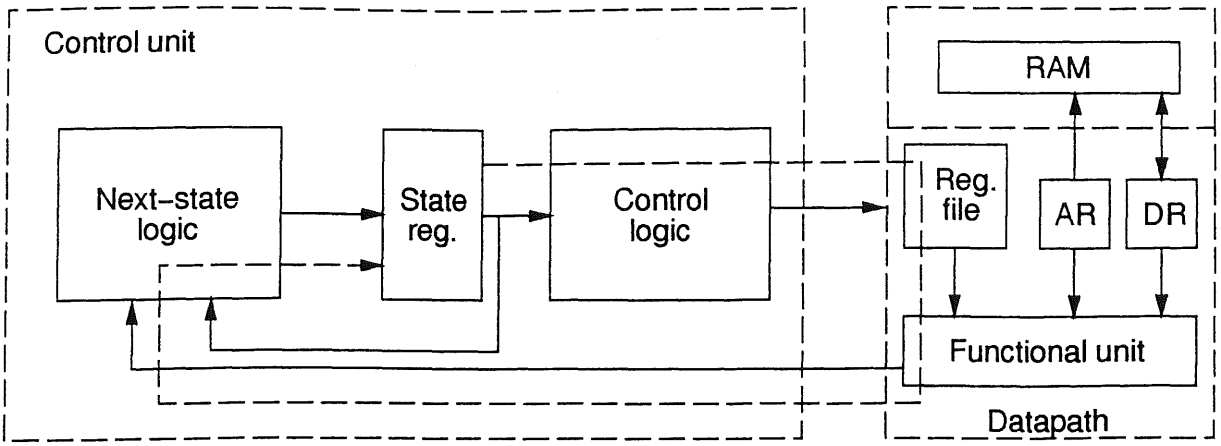

Figure 11: A typical FSMD clocking model

register delay in the datapath. Since the access time to a RAM is slow and often takes several clock cycles, we consider only the storage units as registers or a register file. For each operation *op*, implemented by a functional unit FU, let $R1$, and $R2$ be the input registers, and $R3$ be the output register, connected to *op* through $C1$, $C2$, and $C3$, respectively. In general $C1$, $C2$, and $C3$ are implemented as wires or interconnect units such as a muxes or a buses. Thus, for a single-cycle operation, or a single-cycle chaining operation, *op*, the register-to-register delay of operation *op* is computed as

$$\begin{aligned} t_p(op) = \quad & \max\{t_p(R1), t_p(R2)\} + t_p(FU) + t_{setup}(R3) \\ & + \max\{t_p(C1), t_p(C2)\} + t_p(C3) \end{aligned} \tag{27}$$

and the longest register-to-register delay for each clock-cycle among all operations op_i in the datapath, is $t_p(DP) = \max\{\forall_i t_p(op_i)\}$.

The effects of clock skew can also be accounted for: since our area model can provide approximate locations of the major blocks (and therefore estimate the length of the clock tree branches), the clock arrival times at these blocks can be estimated and used to adjust the clock cycle estimates accordingly. Modeling of multi-phase clocking schemes is accomplished assuming the assignment of phases to registers is determined. In this case, the timing estimation model estimates the delays of all the combinational logic blocks and can feed this information into more sophisticated clocking models (such as [18], for example) to determine a lower bound on the clock period and a clocking scheme. This flexibility allows the use of our models to explore the effect of different clocking schemes on layout performance.

7 Experimental Results

7.1 *Experimental procedure*

Design	Add	Mult	Reg	Mux	Cells	Time Steps
EF1	2†	2°	11	28	2040	19
EF2	3†	1°	10	32	2221	21
EF3	1†	1°	10	20	1757	28
EF4	1†	1°	10	23	1702	28
EF5	1‡	1°	12/5RF	14	948	29
EF6	2‡	1•	10	26	2213	21*
2901	2†	—	18	7	975	-
2910	2†	—	7	2	2450	-

†= Carry-lookahead, ‡= Ripple-carry, °= Macrocell, non-pipelined, •= Shift-add Bitslice Unit, RF=Register File

* Number of control steps. Shift-add multiplier takes 16 clock cycles to execute, number of clock cycles = 77

Table 1: Description of the benchmark designs

In order to benchmark the accuracy of our layout model, we used 8 layouts described in Table 1. These layouts were generated from three standard high level synthesis benchmarks[19]: (1) the Elliptic Filter [20] (EF1-EF6), (2) the AMD 2901 cpu [21], and (3) the AMD 2910 micro-sequencer [21]. All the EF designs have 16 bit datapaths and use muxed functional units and registers and were derived manually except for EF5 which is implemented using register files and described in [22]. The specifications of the AMD 2901 and 2910 were mostly structural and thus, only one RT level implementation of each was considered. The 2901 was designed with a bit width of 4 and the 2910 with a bit width of 12. Both used Carry-lookahead adders. Altogether, the RT-level implementations spanned a reasonably large set of design variations that are likely to be considered during high level design.

While one may comment that these design examples are small compared to real industrial designs, we argue the following reasons in favor of our choice: (1) large size industrial designs are extremely hard to obtain from industry for obvious proprietary reasons, (2) even when such designs are made available, they are usually described in non-standard format which requires substantial modifications in order to process, (3) most of the large real life examples have a fixed structure, and it is usually quite difficult to generate several variations of each design as is done in high level synthesis, (4) simulating large designs is too costly and sometimes cannot be handled, even by our "industrial strength" tools, and finally (5) two of our examples, the 2901 and the 2910 describe standard chips (albeit small ones) used extensively in real life designs.

All the RT-level implementations were written in VHDL and verified for correctness at the functional level. The Mentor Graphics GDT tools were used to generate and simulate the final layouts which were based on the layout methodology explained in Sections 3, 4, 5, and 6. The main difference was that the controller netlist was actually synthesized (instead of "estimated") using a combination of the Berkeley logic design tools and the GDT Autologic tools. For consistency, the same library of standard cells and Macrocells (SCMOS 3μ) was used across all the designs[5]. Each layout was functionally verified using GDT's simulator,

[5]Our choice of this technology was mainly due to is availability in the GDT tools. Clearly, the estimation models themselves

Benchmark design	Measured Area (μ^2)	Estimated Area (μ^2)	% error	Measured clock cycle (ns)	Estimated clock cycle (ns)	% error
EF1	11327×4928	11236×4944	0.4	394	354	10.1
EF2	9141×4798	9338×4650	0.4	413	373	9.6
EF3	7574×4943	8621×4794	9.4	274	258	5.8
EF4	7885×4297	8019×4304	1.8	290	274	5.5
EF5	4881×6143	4285×6409	6.8	419	404	3.2
EF6	6553×4049	6917×4144	7.4	260	242	6.9
2901	5340×1874	5289×1652	12.7	303	332	9.5
2910	3375×3815	3414×3980	5.1	246	231	6.0
Average error			5.5			7.1

Table 2: Estimation results

Lsim in switch mode, and the extracted netlist was backannotated and subsequently simulated in the *Lsim* ADEPT mode to obtain accurate timing waveforms. The worst case cycle delay time was measured from the Lsim output waveforms.

In order to assess the accuracy of our chip level mode, we produced estimates of the chip area and delay using the datapath area and delay models described in Sections 3.1 and 3.2. The controller netlist generated by the logic synthesis tools was input to the controller model described in Section 4. For designs with register files as macrocells, the dimensions of these macrocells were easily derived using simple equations based on the macrocell generators provided within the GDT tools. Once the area and delay information of all the blocks were estimated, we used the chip level area and timing models described in Sections 6.1, 6.2, and 6.3 to obtain estimates of the total chip area and a lower bound on the clock cycle time.

7.2 *Results*

The estimation results are shown in Table 2. First, we note that our area estimate are within 10% with the exception of the AMD2901 where our estimation error was 12.7% Figure 12 graphically compares the ordering of the actual and the estimated layout areas for the designs. We note that the ordering of the areas is the same except for EF5 and EF6, whose ranking was reversed by the estimator. However, given a 10% maximum area error margin, one can say that the two designs are almost equal in area, which can be concluded when the actual areas are compared. Another interesting comparison is between designs EF4 and EF3 which have the same number of control steps and the same number of functional units and similar register count, but different register and mux allocations.

Figure 12 also compares the estimates of area using our model, and other partial models including: functional unit (FU) area (Model B), (FU+register) area (Model C), (FU+register+mux) or datapath area (Model D), and (FU+register+mux+controller) area (Model E). The difference between the last model and the actual area clearly shows that the effect of global wiring and floorplanning is significant in all the

are technology independent and can accommodate newer technologies as well. For very fine technologies (0.5μ and below) and very high frequencies, however, the Elmore model described in Section 2.2 must be relpaced by a more accurate one.

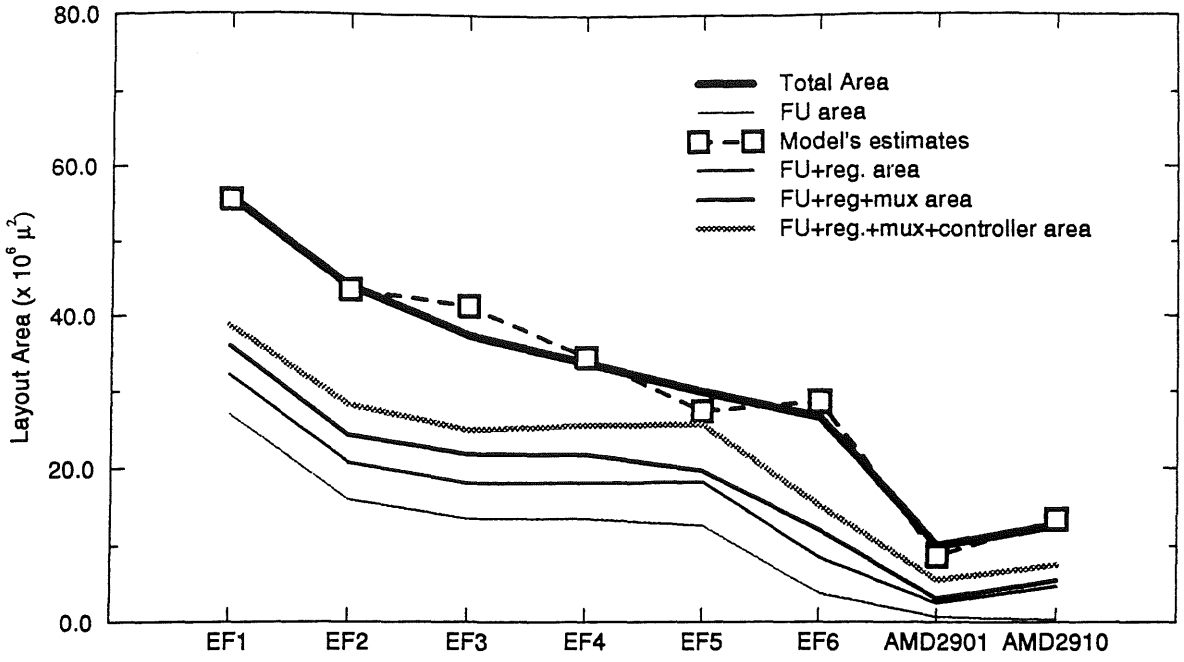

Figure 12: Area comparisons

examples. It is important to note that the FU, mux, register and controller area curves shown in Figure 12 were obtained using our block level models described in Sections 3.1, 4.1, and 5 which account for both cell area and wiring area inside each of the major blocks. When comparing our model to the other models in the graph, we observe that our model is by far the most accurate and the highert quality, with an average error of 5.5% (using Equation 1) while achieving a fidelity of about 93% (using Equation 2) over the Elliptic Filter examples. By comparison, the average errors for models B, C, D, and E are: 63%, 49%, 40%, and 29%, respectively on the same Elliptic Filter examples. The fidelities of these models are: 93%, 80%, 93%, and 80%, in the same order.

The clock cycle estimation results show a maximum error of 10%, and an average error of 7.1%. One interesting design is EF1 in which the long bitslice datapath wires clearly affected the clock cycle time adversely. The same effect is reflected in the model which shows a proportionately large estimate of EF1's cycle time. Another interesting tradeoff is between EF5, which uses register files outside the datapath and EF4 which uses muxed registers inside the data path bitslice. Both designs have one adder and one multiplier each. Note that the cycle time of EF5 is significantly larger than EF4 mainly because of the combined effects of the ripple carry adder delay and the delay in accessing registers inside the register files, which are located outside the bitslice datapath. These effects were also accurately reflected in the cycle time estimated by our model.

Many high level synthesis systems compute the clock cycle time as the cycle delay of the slowest operator among those used in the RT level design [23]. To assess the accuracy and fidelity of our estimation of this metric, we plotted in Figure 13 the actual and estimated clock cycle times for all 9 designs along with the delay of the slowest operator in each design. It can be clearly seen that the latter metric is inadequate

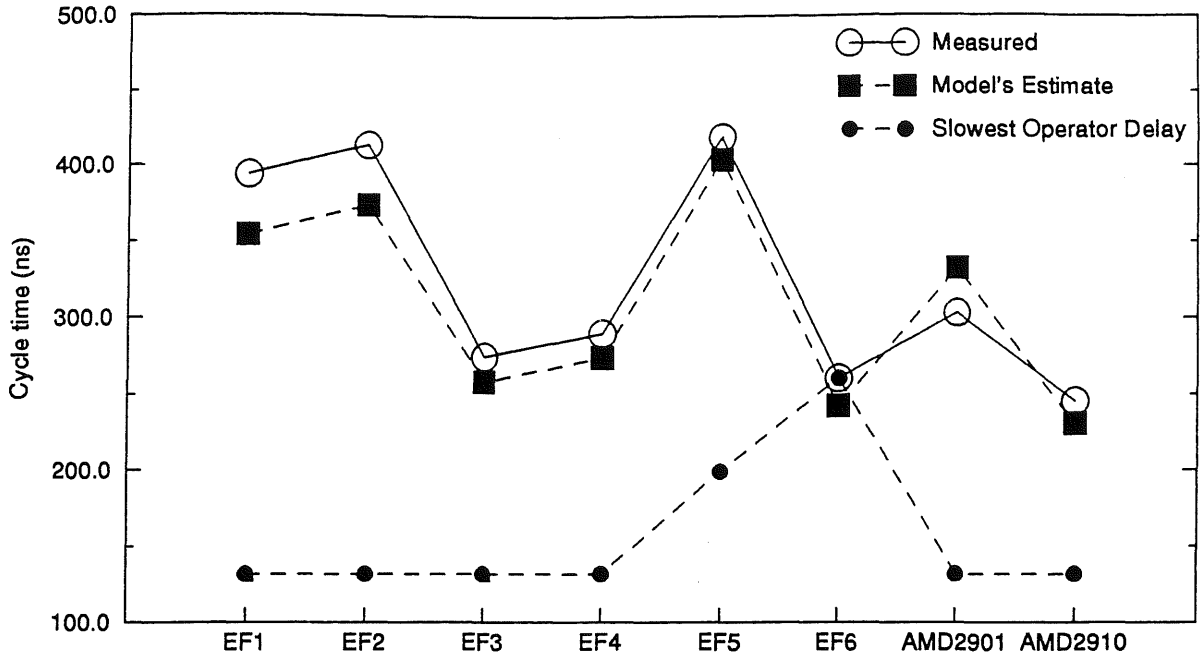

Figure 13: Comparison of measured, predicted clock cycle times, and slowest operator delay

because it does not take the register, mux, and wiring delays into consideration[6] and does indeed achieve low fidelity (26%) as defined by Equation 2. By contrast, our model shows an excellent tracking ability of the actual cycle delay values and achieves 100% fidelity.

Figure 14 is a plot of the area versus overall input-to-output delay (i.e. *cycle_time*$\times$ *#_clock_cycles*) for each EF design[7]. Clearly, the slowest Elliptic Filter design is EF6 which uses an add-shift multiplier whereas the fastest design is EF1 which uses two multiplier and two adders and is scheduled in 19 time steps. The tradeoffs between the other designs are estimated accurately by our model. For example, EF2 is correctly predicted as inferior by our model.

For all the benchmarks, the runtime of the model to estimate both the area and cycle times was under 10 seconds of SUN4 CPU time per design. This is expected, since the heuristics used in the model are either constant time, linear time, or pseudo-linear time. Of course, we need to run larger examples with many more RT level components and benchmark their runtime before making a general statement about the model's efficiency.

8 Conclusions

We presented a new layout predictive model for high level applications. This model accurately and efficiently accounts for a variety of RT level design styles on a chip as is normally required for digital systems layout. We tested our model on a variety of high level design benchmarks. The results show that this model can

[6]The only exception is EF6, where the add-shift multiplier cycle delay was dominant. However, the add-shift multiplier itself was implemented as a bitslice unit whose cycle delay was computed using our model.

[7]We did not include the delays of the AMD designs because they have a variable number of control steps.

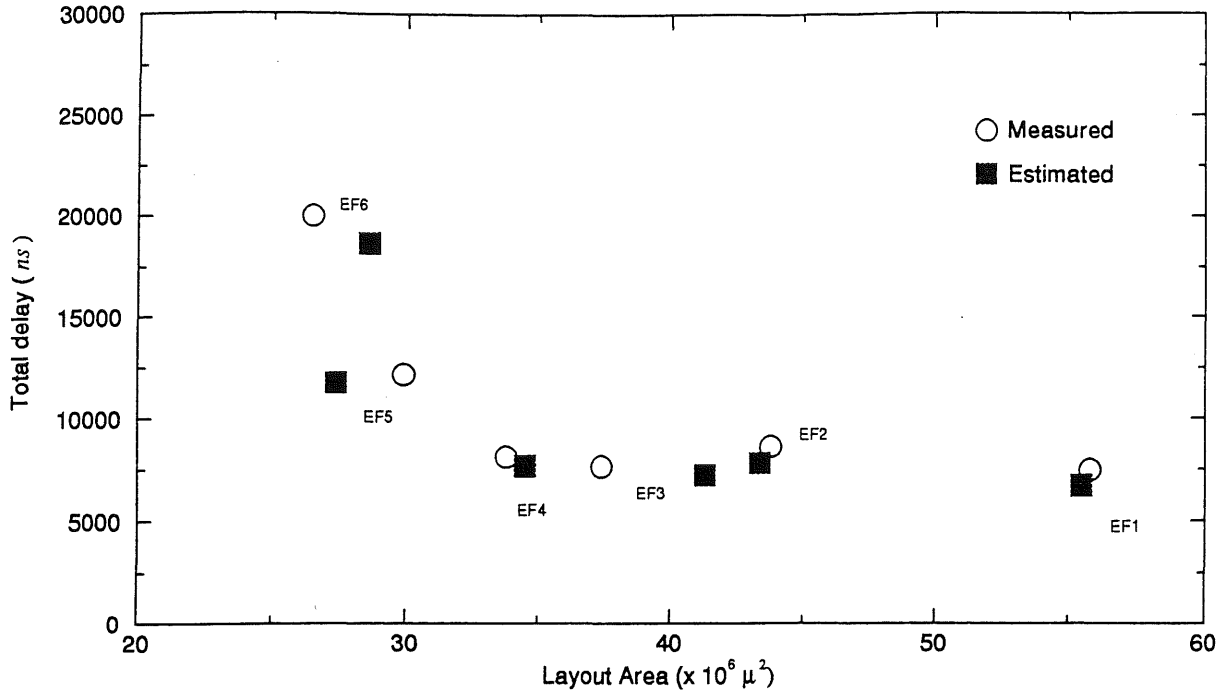

Figure 14: Area versus delay for the Elliptic Filter layouts

reflect the effects of layout on design area and performance much more accurately than typical high level synthesis cost models.

Clearly, more benchmarking on a wider range of design styles is needed before it is possible to make a more general assessment of the accuracy and efficiency of our models. Another issue of importance is to model the effects of control logic optimization on controller area and delay. Our current behavioral controller models can only account for a fraction of these effects. Large scale systems, especially control-dominated systems usually have a large controller contribution to both area and delay. In such cases, the simple modeling of logic synthesis may result in large prediction errors. Thus, we need to develop a more complete paradigm of the overall logic optimization procedure in order to abstract the level of modeling used for the controller. Finally, we need to look into means of incorporating these prediction models into the overall design process with the aim of generating better solutions. These issues and others are open problems and subjects of current research.

Acknowledgements

The authors would like to acknowledge the students in the VLSI design classes at UCI who provided the layouts used for benchmarking our model. This work was supported by NSF grants #MIP-8909677 and #MIP-8922851, and CAL-MICRO grants #90-046 and #91-080. We are grateful for their support.

References

[1] M. McFarland, "Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral

specifications," in *Proc. 23rd Design Automation Conf.*, pp. 474–480, IEEE/ACM, 1986.

[2] M. Pedram and B. Preas, "Interconnection length estimation for optimized standard cell layouts," in *Proc. ICCAD-89*, pp. 390–393, IEEE/ACM, 1989.

[3] M. Pedram and B. Preas, "Accurate prediction of physical design characteristics for random logic," in *Proc. ICCD 89*, IEEE/ACM, 1989.

[4] G. Zimmerman, "A new area and shape function estimation technique for VLSI layouts," in *Proc. 25th Design Automation Conf.*, pp. 60–65, IEEE/ACM, 1988.

[5] F. J. Kurdahi and C. Ramachandran, "LAST: A layout area and shape function estimator for high level applications," in *Proc. Second European Conf. on Design Automation*, Feb. 1991.

[6] C. Ramachandran and F. J. Kurdahi, "TELE: a timing evaluator using layout estimation for high level applications," in *Proc. EDAC-92*, 1992.

[7] A. C.-H. Wu, V. Chaiyakul, and D. D. Gajski, "Layout-area models for high-level synthesis," in *Proc. ICCAD-91*, pp. 34–37, Sept. 1991.

[8] V. Chaiyakul, A. Wu, and D. Gajski, "Timing models for high-level synthesis," in *Proc. EuroDAC-92*, 1992.

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

[10] S. Narayan, F. Vahid, and D. Gajski, "System specification and synthesis with the SpecChart language," in *Proc. ICCAD-91*, pp. 266–269, 1991.

[11] P. Penfield Jr. and J. Rubenstein, "Signal delay in RC tree networks," in *Proc. 18th DAC*, 1981.

[12] W. K. Luk and A. A. Dean, "Multi-stack optimization for data-path chip (microprocessor) layout," in *Proc. 26th DAC*, pp. 110–115, 1989.

[13] F. J. Kurdahi and A. C. Parker, "Techniques for area estimation of VLSI layouts," *IEEE Trans. CAD*, vol. 8, no. 1, pp. 81–92, 1989.

[14] M. Feuer, "Connectivity of random logic," *IEEE Transactions on computers*, vol. C-31, pp. 29–33, January 1982.

[15] P. C. McGeer, *On the Interaction of Functional and Timing Behavior of Combinational Logic Circuits*. PhD thesis, Dept. of EECS, Univ. of California, Berkeley, 1989.

[16] C. Ramachandran and F. J. Kurdahi, "A combined topological and functionality based delay estimation using a layout-driven approach for high level applications," in *Proc. Euro-DAC 92*, Sept. 1992.

[17] M. Buric and T. Matheson, "Silicon compilation environments," in *Proc. IEEE CICC*, 1985.

[18] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Analysis and design of latch-controlled synchronous digital circuits," in *Proc. 27th Design Automation Conf.*, pp. 111–117, June 1990.

[19] N. Dutt, "Status of hlsw92 benchmarks," in *6th International Workshop on High Level Synthesis*, IEEE/ACM, November 1992.

[20] S.-Y. Kung, H. J. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing*. Prentice Hall, 1985.

[21] "Amd 2900 series databook." Advanced Micro Devices.

[22] B. Rouzeyre and G. Sagnes, "Memory area minimization by hierarchical clustering in high-level synthesis," in *Fifth International Workshop on High-Level Synthesis*, Mar. 1991.

[23] R. Jain et. al., "Experience with the ADAM synthesis system," in *Proc. 26th DAC*, pp. 56-61, 1989.